デベロッパーガイド

Amazon DynamoDB

API バージョン 2012-08-10

Amazon DynamoDB: デベロッパーガイド

Table of Contents

Amazon DynamoDB とは

Amazon DynamoDB は、フルマネージド NoSQL データベースサービスであり、シームレスなスケーラビリティを備えた高速で予測可能なパフォーマンスを提供します。DynamoDB を使用すると、ディストリビューションデータベースの運用とスケーリングに伴う管理作業をまかせることができるため、ハードウェアのプロビジョニング、設定と構成、レプリケーション、ソフトウェアパッチ適用、クラスタースケーリングなどを自分で行う必要はなくなります。また、DynamoDB も保管時の暗号化を提供し、機密データの保護における負担と複雑な作業を解消します。詳細については、「保管時の DynamoDB 暗号化」を参照してください。

DynamoDB を使用して、任意の量のデータを保存および取得できるデータベーステーブルを作成し、任意のレベルのリクエストトラフィックを処理できます。ダウンタイムやパフォーマンスが低下することなく、テーブルのスループット容量をスケールアップまたはスケールダウンできます。AWS Management Console を使用して、リソースの使用率とパフォーマンスメトリクスをモニタリングできます。

DynamoDB では、オンデマンドバックアップ機能を使用できます。この機能により、テーブルの完全なバックアップを作成して、規制やコンプライアンス要件を満たすために長期間の保存とアーカイブを行うことができます。詳細については、「DynamoDB のオンデマンドバックアップおよび復元の使用」を参照してください。

オンデマンドバックアップを作成し、Amazon DynamoDB テーブルのポイントインタイムリカバリを有効にすることもできます。ポイントインタイムリカバリを使用することで、オペレーションによってテーブルが誤って上書きされたり削除されたりしないようにできます。ポイントインタイムリカバリを使用すれば、過去 35 日間の任意の時点にテーブルを復元することができます。詳細については、「ポイントインタイムリカバリ: 仕組み」を参照してください。

DynamoDB では、テーブルから有効期限切れの項目を自動的に削除できるため、ストレージの使用量と、関連性がなくなったデータの保存コストを削減できます。詳細については、「Time to Live (TTL)」を参照してください。

高い可用性と耐久性

DynamoDB では、一貫性のある高速パフォーマンスを維持しながら、スループットとストレージの要件を処理できるように、テーブルのデータとトラフィックが十分な数のサーバーに自動的に分散されます。また、すべてのデータをソリッドステートディスク (SSD) に保存し、AWS リージョン内の複数のアベイラビリティーゾーン間で自動的にレプリケートするため、組み込みの高い可用性とデータ堅牢性が実現します。グローバルテーブルを使用して、DynamoDB テーブルを AWS リージョン

間で同期させることができます。詳細については、「グローバルテーブル – DynamoDB の複数リージョンレプリケーション」を参照してください。

DynamoDB の使用開始

最初に以下のセクションを読むことをお勧めします。

- Amazon DynamoDB: 仕組み — DynamoDB の基本的な概念について説明します。
- DynamoDB のセットアップ — DynamoDB (ダウンロード可能なバージョンまたはウェブサービス) のセットアップ方法を説明します。
- DynamoDB にアクセスする — コンソール、AWS CLI、API を使用して DynamoDB にアクセスする方法を説明します。

その後で、2 つのオプションを使用して DynamoDB をすぐに使い始めることができます。

- DynamoDB の使用開始
- DynamoDB および AWS SDK の使用開始

アプリケーション開発の詳細については、以下を参照してください。

- DynamoDB と AWS SDK を使用したプログラミング
- テーブル、項目、クエリ、スキャン、およびインデックスの使用

パフォーマンスを最大にしてスループットコストを最小にするための推奨事項をすばやく確認するには、「DynamoDB を使用した設計とアーキテクチャの設計に関するベストプラクティス」を参照してください。DynamoDB リソースにタグを付ける方法については、「リソースへのタグとラベルの追加」を参照してください。

ベストプラクティス、ハウツーガイド、およびツールについては、Amazon DynamoDB のリソースを参照してください。

AWS Database Migration Service (AWS DMS) を使用して、リレーショナルデータベースまたは MongoDB から DynamoDB テーブルにデータを移行できます。詳細については、AWS Database Migration Service ユーザーガイドを参照してください。

MongoDB を移行ソースとして使用する方法については、「MongoDB を AWS Database Migration Service のソースとして使用する」を参照してください。DynamoDB を移行ターゲットとして使用す

る方法については、「Amazon DynamoDB データベースを AWS Database Migration Service のターゲットとして使用する」を参照してください。

DynamoDB のチュートリアル

次のチュートリアルでは、DynamoDB を理解するためのエンドツーエンドの一連の手順について説明します。これらのチュートリアルは AWS の無料利用枠で完了でき、DynamoDB の実務経験を積むことができます。

- Build an Application Using a NoSQL Key-Value Data Store (NoSQL キー値データストアを使ってアプリケーションを構築する)
- Amazon DynamoDB で NoSQL テーブルを作成してクエリを実行する

Amazon DynamoDB: 仕組み

以下のセクションでは、Amazon DynamoDB サービスコンポーネントと、それらの対話方法の概要を示します。

この概要を読んだ後は、「DynamoDB でのコード例用のテーブルの作成とデータのロード」セクションの操作をお試しください。このセクションでは、サンプルテーブルを作成し、データをアップロードして、いくつかの基本的なデータベース操作を実行します。

言語固有のチュートリアルとサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

トピック

DynamoDB のチートシート

このチートシートは、Amazon DynamoDB とそのさまざまな AWS SDK を操作するためのクイックリファレンスを提供します。

初期セットアップ

1. AWS にサインアップします。
2. プログラムで DynamoDB にアクセスするための AWS アクセスキーを取得します。
3. DynamoDB 認証情報を設定します。

以下も参照してください。

- DynamoDB (ウェブサービス) の設定
- DynamoDB の使用開始
- コアコンポーネントの基本概要

SDK または CLI

任意の SDK を選択するか、AWS CLI を設定します。

> **Note**
>
> Windows で AWS CLI を使用すると、引用符の中にないバックスラッシュ (\) はキャリッジリターンとして扱われます。また、他の引用符の中にある引用符や中かっこはエスケープする必要があります。例については、次のセクションの「テーブルの作成」の [Windows] タブを参照してください。

以下も参照してください。

- DynamoDB でのAWS CLI
- DynamoDB の使用開始 – ステップ 2

基本アクション

このセクションでは、基本的な DynamoDB タスクのコードを提供します。これらのタスクの詳細については、「DynamoDB とAWS SDK の使用開始」を参照してください。

テーブルを作成する

Default

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

Windows

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema ^
        AttributeName=Artist,KeyType=HASH ^
        AttributeName=SongTitle,KeyType=RANGE ^
    --provisioned-throughput ^
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

テーブルへの項目の書き込み

```
aws dynamodb put-item \ --table-name Music \ --item file://item.json
```

テーブルから項目の読み込み

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

テーブルから項目の削除

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

テーブルに対してクエリを実行する

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

テーブルを削除する

```
aws dynamodb delete-table --table-name Music
```

テーブル名のリスト化

```
aws dynamodb list-tables
```

名前付けルール

- すべての名前は UTF-8 を使用してエンコードする必要があり、大文字と小文字が区別されます。
- テーブル名とインデックス名の長さは 3～255 文字で、次の文字のみを含めることができます。
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (下線)
  - `-` (ダッシュ)
  - `.` (ドット)
- 属性名は 1 文字以上、64 KB 未満のサイズである必要があります。

詳細については、「命名規則」を参照してください。

サービスクォータの基本

読み込みユニットと書き込みユニット

- 読み込みキャパシティユニット (RCU) – 最大サイズが 4 KB の項目について、読み込みキャパシティユニット = 1 秒あたり 1 回の強力な整合性のある読み込み、あるいは 1 秒あたり 2 回の結果整合性のある読み込み。
- 書き込みキャパシティユニット (WCU) – 最大サイズが 1 KB の項目について、書き込みキャパシティユニット = 1 秒あたり 1 回の書き込み。

テーブルリミット

- テーブルサイズ – テーブルのサイズには実用的な制限はありません。テーブルは項目数やバイト数について制限がありません。
- テーブルの数 – AWS アカウントについては、AWS リージョンごとに 2,500 個のテーブルという初期クォータがあります。
- クエリとスキャンのページサイズの制限 – クエリまたはスキャンごとに、1 ページあたり 1 MB に制限されています。クエリパラメータまたはテーブルでのスキャン操作の結果データが 1 MB を超える場合、DynamoDB は最初に一致した項目を返します。また、新しいリクエストで次のページを読むときに使用できる `LastEvaluatedKey` プロパティも返されます。

インデックス

- ローカルセカンダリインデックス (LSI) – 最大 5 つのローカルセカンダリインデックスを定義できます。LSI は主に、インデックスがベーステーブルと強い一貫性を持たなければならない場合に役立ちます。
- グローバルセカンダリインデックス (GSI) – デフォルトクォータとして、テーブルごとに 20 個のグローバルセカンダリインデックスがあります。
- テーブル毎のセカンダリインデックス属性の射影 – 合計最大 100 の属性を、1 つのテーブルのすべてのグローバルセカンダリインデックスに射影することができます。これは、ユーザー指定の射影された属性だけに適用されます。

パーティションキー

- パーティションキーと値の最小長は 1 バイトです。最大長は 2048 バイトです
- テーブルまたはセカンダリインデックスについて、パーティションキー値の明確な数に関する実質的な制限はありません。
- ソートキーと値の最小長は 1 バイトです。最大長は 1024 バイトです

- 一般的に、パーティションキーの値ごとのソートキーの値の数について、実質的に制限はありません。セカンダリインデックスを持つテーブルは例外です。

セカンダリインデックス、パーティションキー設計、ソートキー設計の詳細については、「ベストプラクティス」を参照してください。

一般的に使用されるデータ型の制限

- 文字列 – 文字列の長さは、項目の最大サイズである 400 KB に制約されます。文字列は、UTF-8 バイナリエンコードの Unicode です。
- 数値 – 数値は、最大 38 桁の精度であり、正、負、または 0 のいずれかです。
- バイナリ – バイナリの長さは、項目の最大サイズである 400 KB に制約されます。バイナリ属性を操作するアプリケーションは、データを DynamoDB に送信する前に、base64 でエンコードする必要があります。

サポートされているデータ型のリストについては、「データ型」を参照してください。詳細については、「Service Quotas」(サービスクォータ) を参照してください。

項目、属性、式パラメータ

DynamoDB の項目の最大サイズは 400 KB で、属性名のバイナリの長さ (UTF-8 の長さ) と属性値の長さ (UTF-8 の長さ) を含みます。属性名はサイズ制限に反映されます。

値を含む項目が 400 KB の制限内である限り、リスト、マップ、またはセットにおける値の最大数の制限はありません。

式パラメータでは、式文字列の最大長は 4 KB です。

項目サイズ、属性、および式パラメータの詳細については、「Service Quotas」(サービスクォータ) を参照してください。

詳細情報

- セキュリティ
- モニタリングとログ記録
- ストリームの使用
- バックアップとポイントインタイムリカバリ

- 他の AWS のサービスとの統合
- API リファレンス
- アーキテクチャセンター: データベースのベストプラクティス
- ビデオチュートリアル
- DynamoDB フォーラム

Amazon DynamoDB のコアコンポーネント

DynamoDB では、テーブル、項目、および属性が、操作するコアコンポーネントです。テーブルは項目の集合であり、各項目は属性の集合です。DynamoDB は、プライマリキーを使用してテーブルの各項目を一意に識別し、セカンダリインデックスを使用してクエリの柔軟性を高めます。DynamoDB Streams を使用して、DynamoDB テーブルのデータ変更イベントをキャプチャできます。

DynamoDB には制限があります。詳細については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。

次の動画では、テーブル、項目、および属性の概要を説明します。

テーブル、項目、属性

テーブル、項目、属性

基本的な DynamoDB コンポーネントは以下のとおりです。

- テーブル – 他のデータベースシステムと同様、DynamoDB はデータをテーブルに保存します。テーブルは、データのコレクションです。たとえば、テーブルの例 (People) を参照してください。このテーブルは、友人、家族、関心のある人に関する個人の連絡先情報を保存するのに使用できます。また、その人たちが運転する車に関する情報を保存する Cars テーブルを作成することもできます。
- 項目 – 各テーブルにはゼロ以上の項目が含まれています。項目は、他のすべての項目間で一意に識別可能な属性のグループです。People テーブルの各項目は、人を表します。Cars テーブルの各項目は 1 台の車を表します。DynamoDB の項目は、多くの点で他のデータベースシステムの行、レコード、またはタプルに似ています。DynamoDB では、テーブルに保存できる項目数に制限はありません。
- 属性 – 各項目は 1 つ以上の属性で構成されます。属性は、基盤となるデータ要素であり、それ以上分割する必要がないものです。例えば、People テーブルの項目に

は、PersonID、LastName、FirstName といった名前の属性が含まれます。Department テーブルでは、項目が DepartmentID、Name、Manager などの属性を設定することができます。DynamoDB 内の属性は、多くの点で他のデータベースシステムのフィールドや列に似ています。

次の図は、いくつかの項目と属性の例を含む、People という名前のテーブルを示しています。

```
People

{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}

{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}

{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
                "Street": "123 Main",
                "City": "London",
                "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

People テーブルについて、以下の点に注意してください。

- テーブルの各項目には一意の識別子があります。これは、テーブルの他のすべての項目からその項目を区別するプライマリキーです。People テーブルで、プライマリキーは 1 つの属性 (PersonID) で構成されます。
- プライマリキー以外、People テーブルはスキーマレスです。つまり、属性またはデータ型を事前に定義する必要はありません。各項目は、独自の固有の属性を持つことができます。
- 属性のほとんどはスカラーです。つまり、1 つの値のみを持つことができます。文字列と数値はスカラーの一般的な例です。
- 一部の項目には、ネストされた属性 (アドレス) があります。DynamoDB は深さが最大 32 レベルの入れ子の属性をサポートします。

以下は、音楽コレクションを追跡するために使用できる、Music という名前の別のサンプルテーブルです。

```
Music

{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}

{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Music テーブルについて、以下の点に注意してください。

- Music のプライマリキーは 2 つの属性 (Artist および SongTitle) で構成されます。テーブルの各項目にはこれら 2 つの属性が必要です。Artist および SongTitle の組み合わせにより、テーブルの各項目が他のすべての項目から区別されます。
- プライマリキー以外、Music テーブルはスキーマレスです。つまり、属性またはデータ型を事前に定義する必要はありません。各項目は、独自の固有の属性を持つことができます。
- 項目の 1 つに、入れ子の属性 (PromotionInfo) があります。これには、入れ子の他の属性が含まれます。DynamoDB は深さが最大 32 レベルの入れ子の属性をサポートします。

詳細については、「DynamoDB でのテーブルとデータの操作」を参照してください。

プライマリキー

テーブルを作成する場合には、テーブル名に加えて、テーブルのプライマリキーを指定する必要があります。プライマリキーはテーブルの各項目を一意に識別するため、テーブル内の 2 つの項目が同じキーを持つことはありません。

DynamoDB は 2 種類の異なるプライマリキーをサポートします。

- パーティションキー – パーティションキーという 1 つの属性で構成されたシンプルなプライマリキー。

  DynamoDB は、パーティションキーの値を内部ハッシュ関数への入力として使用します。ハッシュ関数からの出力により、項目が保存されるパーティション (DynamoDB 内部の物理ストレージ) が決まります。

  パーティションキーのみを含むテーブルでは、2 つの項目が同じパーティションキー値を持つことはできません。

  テーブル、項目、属性 で説明されている People テーブルは、単純なプライマリキー (PersonID) を持つテーブルの例です。People テーブル内の任意の項目に直接アクセスするには、その項目の PersonId 値を指定します。
- パーティションとソートキー – 複合プライマリキーと呼ばれるこのキーのタイプは、2 つの属性で構成されます。最初の属性はパーティションキーであり、2 番目の属性はソートキーです。

  DynamoDB は、パーティションキーバリューを内部ハッシュ関数への入力として使用します。ハッシュ関数からの出力により、項目が保存されるパーティション (DynamoDB 内部の物理ストレージ) が決まります。同じパーティションキー値を持つすべての項目は、ソートキー値でソートされてまとめて保存されます。

  パーティションキーとソートキーが存在するテーブルでは、同じパーティションのキーバリューが複数の項目に割り当てられることがあります。ただし、ソートキー値は複数の項目で異なる必要があります。

  テーブル、項目、属性 で説明されている Music テーブルは、複合プライマリキー (Artist および SongTitle) を持つテーブルの例です。その項目に Artist と SongTitle の値を指定すると、Music テーブルの任意の項目に直接アクセスできます。

  複合プライマリキーは、データのクエリを実行するときに柔軟性を高めます。たとえば、Artist の値のみを指定した場合、DynamoDB はそのアーティストのすべての曲を取得します。特定のアーティストの曲のサブセットのみを取得するには、Artist の値と SongTitle の値範囲を指定します。

> ⓘ Note
>
> 項目のパーティションキーは、そのハッシュ属性とも呼ばれます。ハッシュ属性という用語は、DynamoDB が内部のハッシュ関数を使用し、パーティションキーバリューに基づいてパーティション間でデータ項目を均等に分散することに由来しています。
> 項目のソートキーは、範囲属性とも呼ばれます。範囲属性という用語は、ソートキーバリューで並べ替えられた順に、DynamoDB が同じパーティションキーを持つ項目同士を物理的に近くに保存する方法に由来しています。

各プライマリキー属性はスカラー値 (単一値のみを保持できる) である必要があります。プライマリキー属性に許可される唯一のデータ型は、文字列、数値、またはバイナリです。他のキー以外の属性では、このような制限はありません。

セカンダリインデックス

テーブルで 1 つ以上のセカンダリインデックスを作成できます。セカンダリインデックスでは、プライマリキーに対するクエリに加えて、代替キーを使用して、テーブル内のデータのクエリを行うことができます。DynamoDB では、インデックスを使用する必要はありませんが、インデックスを使用すると、データのクエリを行う際にアプリケーションの柔軟性が高まります。テーブルにグローバルセカンダリインデックスを作成すると、テーブルから行う場合とほぼ同じ方法でインデックスからデータを読み取ることができます。

DynamoDB では、次の 2 種類のインデックスをサポートしています。

- グローバルセカンダリインデックス – パーティションキーおよびソートキーを持つインデックス。テーブルのものとは異なる場合があります。
- ローカルセカンダリインデックス – パーティションキーはテーブルと同じですが、ソートキーが異なるインデックスです。

DynamoDB では、グローバルセカンダリインデックス (GSI) はテーブル全体にまたがるインデックスであり、すべてのパーティションキーをクエリできます。ローカルセカンダリインデックス (LSI) は、ベーステーブルとパーティションキーは同じで、ソートキーが異なるインデックスです。

DynamoDB の各テーブルには、20 個のグローバルセカンダリーインデックス (デフォルトのクォータ) と、5 個のローカルセカンダリーインデックスのクォータがあります。

前に示した Music サンプルテーブルでは、Artist (パーティションキー) または Artist および SongTitle (パーティションキーとソートキー) によってデータ項目にクエリを実行できます。Genre および

AlbumTitle によってデータにクエリを実行する場合はどうでしょうか。これを行うには、Genre および AlbumTitle にインデックスを作成し、Music テーブルのクエリと同様に、インデックスにクエリを実行できます。

次の図表は、GenreAlbumTitle という新しいインデックスを持つ Music テーブルの例を示しています。このインデックスでは、Genre がパーティションキーで、AlbumTitle がソートキーです。

Music テーブル	GenreAlbumTitle

{
"Artist": "No One You Know",
"SongTitle": "My Dog Spot",
"AlbumTitle": "Hey Now",
"Price": 1.98,
"Genre": "Country",
"CriticRating": 8.4
}
	{
	"Genre": "Country",
	"AlbumTitle": "Hey Now",
	"Artist": "No One You Know",
	"SongTitle": "My Dog Spot"
	}

{
"Artist": "No One You Know",
"SongTitle": "Somewhere Down The Road",
"AlbumTitle": "Somewhat Famous",
"Genre": "Country",
"CriticRating": 8.4,
"Year": 1984
}
	{
	"Genre": "Country",
	"AlbumTitle": "Somewhat Famous",
	"Artist": "No One You Know",
	"SongTitle": "Somewhere Down The Road"
	}

{
"Artist": "The Acme Band",
"SongTitle": "Still in Love",
"AlbumTitle": "The Buck Starts Here",
"Price": 2.47,
"Genre": "Rock",
	{
	"Genre": "Rock",
	"AlbumTitle": "The Buck Starts Here",
	"Artist": "The Acme Band",
	"SongTitle": "Still In Love"
	}

Music テーブル	GenreAlbumTitle

`    "PromotionInfo": {`
`        "RadioStationsPlaying": {`
`            "KHCR",`
`            "KQBX",`
`            "WTNR",`
`            "WJJH"`
`        },`
`        "TourDates": {`
`            "Seattle": "20150622",`
`            "Cleveland": "20150630"`
`        },`
`        "Rotation": "Heavy"`
`    }`
`}`	

`{`
`    "Artist": "The Acme Band",`
`    "SongTitle": "Look Out, World",`
`    "AlbumTitle": "The Buck Starts`
` Here",`
`    "Price": 0.99,`
`    "Genre": "Rock"`
`}`
	`{`
	`    "Genre": "Rock",`
	`    "AlbumTitle": "The Buck Starts`
	` Here",`
	`    "Artist": "The Acme Band",`
	`    "SongTitle": "Look Out, World"`
	`}`

GenreAlbumTitle インデックスについて、以下の点に注意してください。

- 各インデックスはテーブルに属します。これをインデックスの基本テーブルと呼びます。前述の例では、Music が GenreAlbumTitle インデックスの基本テーブルです。
- DynamoDB はインデックスを自動的に維持します。基本テーブルの項目を追加、更新、または削除すると、DynamoDB はそのテーブルに属するすべてのインデックスの対応する項目を追加、更新、または削除します。
- インデックスを作成するときは、基本テーブルからインデックスにコピーまたは射影される属性を指定します。少なくとも、DynamoDB は基本テーブルからインデックスにキー属性を射影しま

す。これは `GenreAlbumTitle` のケースで、`Music` テーブルのキー属性のみがインデックスに射影されます。

GenreAlbumTitle インデックスにクエリを実行し、特定のジャンルのすべてのアルバム (たとえば、すべての Rock アルバム) を検索できます。また、インデックスにクエリを実行して、特定のジャンル内のすべてのアルバムのうち、特定のアルバムタイトル (たとえば、タイトルが文字 H で始まるすべての Country アルバム) のみを検索することもできます。

詳細については、「セカンダリインデックスを使用したデータアクセス性の向上」を参照してください。

DynamoDB Streams

DynamoDB Streams は、DynamoDB テーブルのデータ変更イベントをキャプチャするオプションの特徴です。これらのイベントに関するデータは、ほとんどリアルタイムに、イベントの発生順にストリームに表示されます。

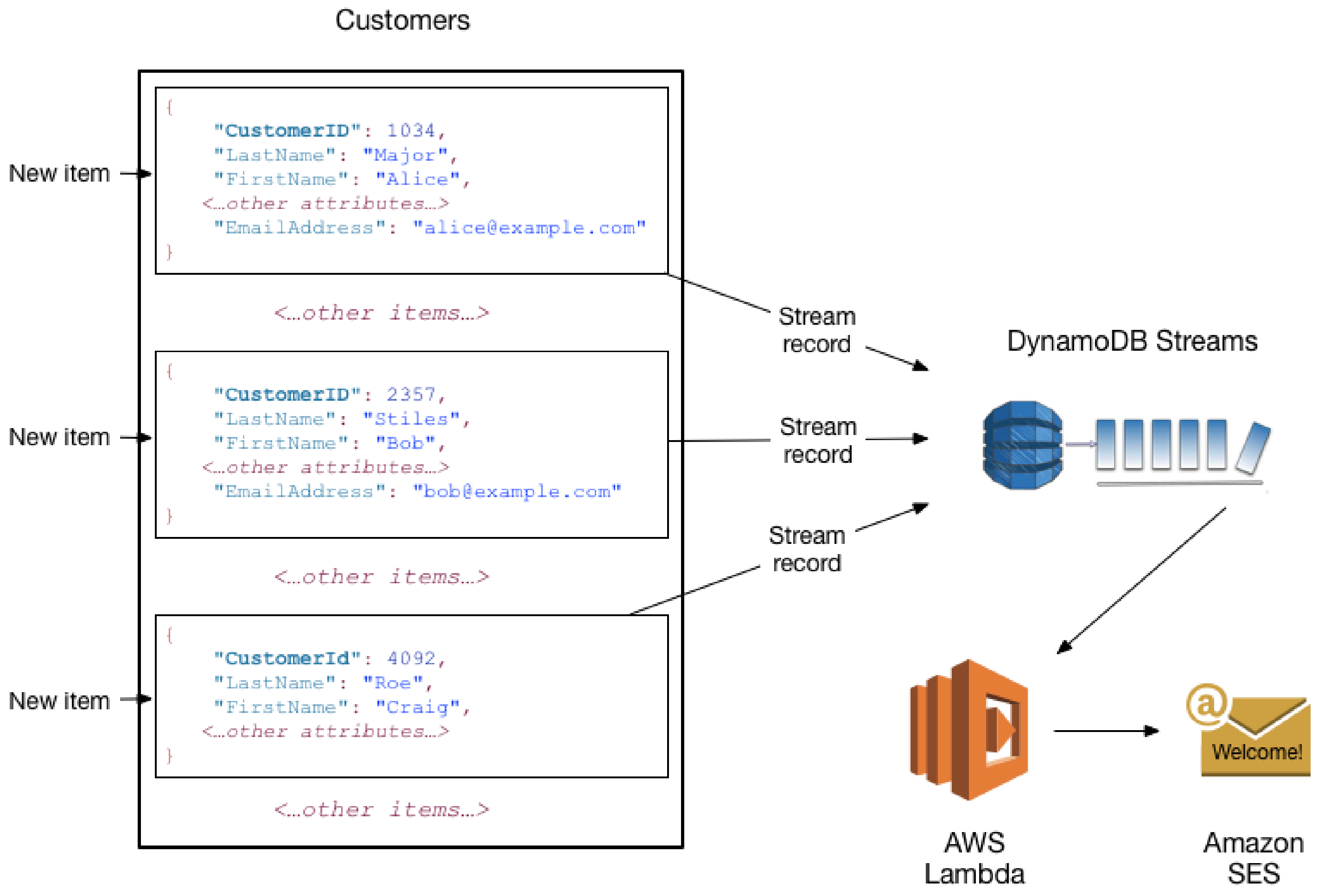
各イベントはストリームレコードによって表されます。テーブルでストリーミングを有効にすると、DynamoDB Streams は次のいずれかのイベントが発生するたびに、ストリーミングレコードを書き込みます。

- 新しい項目がテーブルに追加された場合: ストリームは、すべての属性を含む項目全体のイメージをキャプチャします。
- 項目が更新された場合: ストリームは、項目で変更された属性について、「前」と「後」のイメージをキャプチャします。
- テーブルから項目が削除された場合: ストリームは、項目が削除される前に項目全体のイメージをキャプチャします。

各ストリームレコードには、テーブルの名前、イベントのタイムスタンプ、およびその他のメタデータも含まれます。ストリームレコードには 24 時間の有効期間があり、その後はストリームから自動的に削除されます。

DynamoDB Streams を AWS Lambda と共に使用して、トリガーを作成できます。トリガーは、対象イベントがストリーミングに表示されるたびに自動的に実行されます。たとえば、会社の顧客情報を含む Customers テーブルがあるとします。新規の各顧客に、「ようこそ」 E メールを送信するとします。そのテーブルでストリーミングを有効にし、そのストリーミングを Lambda 関数に関連付けます。Lambda 関数は、新しいストリーミングレコードが表示されるたびに実行されます

が、Customers テーブルに追加された新しい項目のみを処理します。EmailAddress 属性を持つ項目について、Lambda 関数は Amazon Simple Email Service (Amazon SES) をコールしてそのアドレスに E メールを送信します。



```
{
  "CustomerID": 1034,
  "LastName": "Major",
  "FirstName": "Alice",
 <…other attributes…>
  "EmailAddress": "alice@example.com"
}
```

```
{
  "CustomerID": 2357,
  "LastName": "Stiles",
  "FirstName": "Bob",
 <…other attributes…>
  "EmailAddress": "bob@example.com"
}
```

```
{
  "CustomerId": 4092,
  "LastName": "Roe",
  "FirstName": "Craig",
 <…other attributes…>
}
```

> **Note**
>
> この例で、最後の顧客 Craig Roe には EmailAddress がないため E メールを受信することはありません。

トリガーに加えて、DynamoDB Streams は AWS リージョン内全体のデータレプリケーション、DynamoDB テーブル内のデータのマテリアライズドビュー、Kinesis のマテリアライズドビューを使用したデータ分析など、数多くの強力なソリューションを可能にします。

詳細については、「DynamoDB Streams の変更データキャプチャ」を参照してください。

DynamoDB API

Amazon DynamoDB を使用するには、アプリケーションでいくつかの簡単な API オペレーションを使用する必要があります。以下に、カテゴリー別にこれらのオペレーションの概要を示します。

> ⓘ Note
>
> API オペレーションの完全なリストについては、「Amazon DynamoDB API リファレンス」を参照してください。

トピック

コントロールプレーン

コントロールプレーンのオペレーションでは、DynamoDB テーブルを作成および管理できます。また、インデックス、ストリーム、およびテーブルに依存する他のオブジェクトを操作できます。

- `CreateTable` – 新しいテーブルを作成します。オプションで、1 つ以上のセカンダリインデックスを作成し、テーブルに対して DynamoDB Streams を有効にできます。
- `DescribeTable` – プライマリキーのスキーマ、スループット設定、インデックス情報など、テーブルに関する情報を返します。
- `ListTables` – リストのすべてのテーブルの名前を返します。
- `UpdateTable` – テーブルまたはそのインデックスの設定を変更、テーブルの新しいインデックスを作成または削除、またはテーブルの DynamoDB Streams 設定を変更します。
- `DeleteTable` – テーブルとそのすべての依存オブジェクトを DynamoDB から削除します。

データプレーン

データプレーンオペレーションでは、テーブルのデータで、作成、読み込み、更新、および削除 (CRUD とも呼ばれる) アクションを実行できます。一部のデータプレーンオペレーションでも、セカンダリインデックスからデータを読み込むことができます。

PartiQL: Amazon DynamoDB 用の SQL 互換クエリ言語 を使用してこれらの CRUD オペレーションを実行するか、各オペレーションを個別の API コールに分離する DynamoDB の従来の CRUD API を使用できます。

PartiQL - SQL 互換クエリ言語

- `ExecuteStatement` – テーブルから複数の項目を読み込みます。テーブルから単一の項目を書き込むか、更新することもできます。単一の項目を書き込むか、更新する場合は、プライマリキー属性を指定する必要があります。
- `BatchExecuteStatement` – テーブルから複数の項目を書き込み、更新または読み込みます。これは、`ExecuteStatement` よりも効率的です。アプリケーションで項目を書き込んだり読み込んだりするために、1 回のネットワークラウンドトリップのみで済むためです。

Classic API

データの作成

- `PutItem` – テーブルに単一の項目を書き込みます。プライマリキー属性を指定する必要がありますが、その他の属性を指定する必要はありません。
- `BatchWriteItem` – 最大 25 個の項目をテーブルに書き込みます。これは、`PutItem` を複数回呼び出すよりも効率的です。アプリケーションで項目を書き込むために、1 回のネットワークラウンドトリップのみで済むためです。

データの読み込み

- `GetItem` – テーブルから単一の項目を取り出します。目的の項目のプライマリキーを指定する必要があります。項目全体またはその属性のサブセットのみを取り出すことができます。
- `BatchGetItem` – 1 つ以上のテーブルから最大 100 個の項目を取り出します。これは、`GetItem` を複数回呼び出すよりも効率的です。アプリケーションで項目を読み込むために、1 回のネットワークラウンドトリップのみで済むためです。
- `Query` – 特定のパーティションキーがあるすべての項目を取り出します。パーティションキーの値を指定する必要があります。項目全体またはその属性のサブセットのみを取り出すことができます。オプションで、ソートキーの値に条件を適用し、同じパーティションキーがあるデータのサブセットだけを取り出すことができます。テーブルにパーティションキーとソートキーの両方を持つテーブルがある場合、テーブルでこのオペレーションを使用できます。また、インデックスにパーティションキーとソートキーの両方がある場合、インデックスでこのオペレーションを使用できます。

- Scan – 指定されたテーブルまたはインデックスのすべての項目を取り出します。項目全体またはその属性のサブセットのみを取り出すことができます。オプションでフィルタリング条件を適用すると、関心のある値のみを返し、残りは破棄できます。

データの更新

- UpdateItem – 項目の 1 つ以上の属性を変更します。変更する項目のプライマリキーを指定する必要があります。新しい属性を追加したり、既存の属性を変更または削除したりできます。ユーザー定義の条件を満たす場合にのみ更新が成功するように、条件付きの更新を実行できます。オプションで、アトミックカウンターを実装できます。このカウンタは、他の書き込みリクエストを妨害することなく、数値属性をインクリメントまたはデクリメントします。

データの削除

- DeleteItem – テーブルから単一の項目を削除します。削除する項目のプライマリキーを指定する必要があります。
- BatchWriteItem – 1 つ以上のテーブルから最大 25 個の項目を削除します。これは、DeleteItem を複数回呼び出すよりも効率的です。アプリケーションで項目を削除するために、1 回のネットワークラウンドトリップのみで済むためです。

> Note
>
> BatchWriteItem は、データの作成とデータの削除の両方に使用できます。

DynamoDB Streams

DynamoDB Streams オペレーションでは、テーブルのストリーミングを有効または無効にし、ストリーミングに含まれるデータ変更レコードにアクセスするように許可します。

- ListStreams – すべてのストリーミングのリスト、または特定のテーブルのストリーミングのみを返します。
- DescribeStream – Amazon リソースネーム (ARN) およびアプリケーションが最初のいくつかのストリーミングレコードの読み込みを開始できる場所など、ストリーミングに関する情報を返します。
- GetShardIterator – シャードイテレーターを返します。これは、ストリーミングからレコードを取得するためにアプリケーションが使用するデータ構造です。

- `GetRecords` – 特定のシャードイテレーターを使用して 1 つ以上のストリーミングレコードを取得します。

トランザクション

トランザクションによって不可分性、一貫性、分離性、耐久性 (ACID) が実現されるため、アプリケーション内でのデータの精度を維持することがさらに容易になります。

PartiQL: Amazon DynamoDB 用の SQL 互換クエリ言語 を使用してこれらのトランザクションオペレーションを実行するか、各オペレーションを個別の API コールに分離する DynamoDB の従来の CRUD API を使用できます。

PartiQL - SQL 互換クエリ言語

- `ExecuteTransaction` – テーブル内または複数のテーブル間の複数の項目に対して、オールオアナッシングの結果が保証された CRUD オペレーションを実行できるバッチ操作です。

Classic API

- `TransactWriteItems` – テーブル内または複数のテーブル間の複数の項目に対して、オールオアナッシングの結果が保証された `Put`、`Update` および `Delete` オペレーションを実行できるバッチオペレーションです。
- `TransactGetItems` – 1 つ以上のテーブルから複数の項目を取得する `Get` オペレーションを実行できるバッチ操作です。

Amazon DynamoDB でサポートされるデータ型と命名規則

このセクションでは、Amazon DynamoDB の命名規則と、DynamoDB がサポートするさまざまなデータ型について説明します。データタイプに適用される制限があります。詳細については、「データ型」を参照してください。

トピック

名前付けルール

DynamoDB のテーブル、属性、および他のオブジェクトには名前が必要です。名前には意味があり、簡潔でなければなりません。たとえば、製品、ブック、および著者などの名前は一目瞭然です。

DynamoDB の命名規則は次のとおりです。

- すべての名前は UTF-8 を使用してエンコードする必要があり、大文字と小文字が区別されます。
- テーブル名とインデックス名の長さは 3～255 文字で、次の文字のみを含めることができます。
  - a-z
  - A-Z
  - 0-9
  - _ (下線)
  - - (ダッシュ)
  - . (ドット)
- 属性名は 1 文字以上の長さ、64 KB 未満のサイズにする必要があります。属性名はできるだけ短くすることがベストプラクティスであると考えられています。これにより、属性名がストレージとスループットの使用量の測定に含まれるため、消費される読み取りリクエストユニットを減らすことができます。

  以下の例外があります。これらの属性名は 255 文字以下である必要があります。

  - セカンダリインデックスのパーティションキー名
  - セカンダリインデックスのソートキー名
  - ユーザー指定の射影された属性の名前 (ローカルセカンダリインデックスにのみ適用)

予約語と特殊文字

DynamoDB には予約語と特殊文字のリストもあります。詳細な一覧については、「DynamoDB の予約語」を参照してください。DynamoDB では、# (ハッシュ) および : (コロン) に特別な意味があります。

DynamoDB では、命名目的でこれらの予約語と特殊文字を使用することができますが、お勧めしません。これは、式でこれらの名前を使用するたびに、プレースホルダー変数を定義する必要があるためです。詳細については、「DynamoDB の式の属性名」を参照してください。

データ型

DynamoDB では、テーブル内の属性に対してさまざまなデータ型がサポートされています。データ型は次のように分類できます。

- スカラー型 – スカラー型は 1 つの値を表すことができます。スカラー型は、数値、文字列、バイナリ、ブール、および null です。
- ドキュメント型 – JSON ドキュメントで見られるように、入れ子の属性を持つ複雑な構造を表すことができます。ドキュメント型は、リストとマップです。
- セット型 – セット型は複数のスカラー値を表すことができます。セット型は、文字セット、数値セット、およびバイナリセットです。

テーブルまたはセカンダリインデックスを作成するときは、各プライマリキー属性 (パーティションキーとソートキー) の名前とデータ型を指定する必要があります。さらに、各プライマリキー属性は、文字列、数値、またはバイナリとして定義する必要があります。

DynamoDB は NoSQL データベースであり、スキーマレスです。つまり、テーブルを作成する場合、プライマリキー属性を除き、属性やデータ型を定義する必要はありません。これに対して、リレーショナルデータベースでは、テーブルの作成時に各列の名前とデータ型を定義する必要があります。

ここでは、各データ型についてJSON 形式の例を挙げて説明します。

スカラー型

スカラー型は、数値、文字列、バイナリ、ブール、および null です。

数

数値は、正、負、または 0 とすることができます。数値は最大 38 桁の精度を持つことができます。これを超えると例外が発生します。38 桁を超える精度が必要な場合は、文字列を使用できます。

- 正の範囲: 1E-130 ~ 9.9999999999999999999999999999999999999E+125
- 負の範囲: -9.9999999999999999999999999999999999999E+125 ~ -1E-130

DynamoDB では、数値は可変長で表されます。先頭と末尾の 0 は切り捨てられます。

すべての数値はネットワーク経由で文字列として DynamoDB に送信され、言語やライブラリ間の互換性が最大になります。ただし、DynamoDB は算術演算では数値を数値型属性として扱います。

数値データ型を使用して、日付またはタイムスタンプを表すことができます。これを行うための方法の 1 つは、1970 年 1 月 1 日 00:00:00 UTC から経過した秒数であるエポックタイムを使用することです。たとえば、エポック時間 `1437136300` は、2015 年 7 月 17 日の 12:31:40 PM UTC を表します。

詳細については、http://en.wikipedia.org/wiki/Unix_time を参照してください。

文字列

文字列は、UTF-8 バイナリエンコードの Unicode です。属性がインデックスまたはテーブルのキーとして使用されない場合、文字列の最小長は 0 になります。また、DynamoDB 項目の最大サイズ上限である 400 KB の制約があります。

文字列型として定義されているプライマリキー属性には、さらに以下の制約が適用されます。

- シンプルなプライマリキーの場合、最初の属性値 (パーティションキー) の最大長は 2,048 バイトです。
- 複合プライマリキーの場合、2 番目の属性値 (ソートキー) の最大長は 1,024 バイトです。

DynamoDB は、基礎となる UTF-8 文字列エンコードのバイトを使用して文字列を照合し、比較します。たとえば、「`a`」 (0x61) は「`A`」 (0x41) より大きく、「`¿`」 (0xC2BF) は 「`z`」 (0x7A) より大きいです。

文字列データ型を使用して、日付またはタイムスタンプを表すことができます。これを行う 1 つの方法は、これらの例に示すように、ISO 8601 文字列を使用することです。

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

詳細については、http://en.wikipedia.org/wiki/ISO_8601 を参照してください。

> **Note**
>
> 従来のリレーショナルデータベースとは異なり、DynamoDB は日付と時刻のデータ型をネイティブにサポートしていません。代わりに、Unix エポック時間を使用して、日付と時刻のデータを数値データ型として保存すると便利です。

バイナリ

バイナリ型の属性には、圧縮テキスト、暗号化データ、イメージなど、任意のバイナリデータが保存されます。DynamoDB は、バイナリ値を比較するたびに、各バイナリデータを符号なしとして扱います。

属性がインデックスまたはテーブルのキーとして使用されず、DynamoDB 項目の最大サイズ上限が 400 KB に制約されている場合、バイナリ属性の長さは 0 になります。

プライマリキー属性をバイナリ型属性として定義する場合、以下の制約がさらに適用されます。

- シンプルなプライマリキーの場合、最初の属性値 (パーティションキー) の最大長は 2,048 バイトです。
- 複合プライマリキーの場合、2 番目の属性値 (ソートキー) の最大長は 1,024 バイトです。

アプリケーションは、DynamoDB に送信する前に、base64 エンコード形式のバイナリ値をエンコードする必要があります。DynamoDB は、これらの値を受信すると、データを符号なしバイト配列にデコードし、それをバイナリ属性の長さとして使用します。

次に示しているのはバイナリ属性の例であり、Base64 でエンコードされたテキストを使用しています。

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

ブール値

ブール型の属性には、`true` または `false` が格納されます。

Null

Null は不明または未定義の状態の属性を表します。

ドキュメント型

ドキュメント型は、リストとマップです。これらのデータ型は、相互に入れ子にして、最大 32 レベルの深さまで複雑なデータ構造を表すことができます。

値を含む項目が DynamoDB のサイズ制限 (400 KB) 内である限り、リストまたはマップの値の最大数の制限はありません。

属性値がテーブルまたはインデックスキーに使用されていない場合、空の文字列または空のバイナリ値にすることができます。属性値は空の文字列または空のセット (文字列セット、数値セット、また

はバイナリセット) にすることはできませんが、空のリストとマップは許可されます。リストとマップ内では、空の文字列とバイナリ値が許可されます。詳細については、「属性」を参照してください。

リスト

リスト型の属性には、順序付きの値のコレクションを保存できます。リストは角括弧で囲まれます: `[ ... ]`

リストは JSON 配列に似ています。リスト要素に保存できるデータ型に制限はなく、リスト要素の要素が同じ型である必要はありません。

次の例は、2 つの文字列と数が含まれるリストを示します。

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> Note
>
> DynamoDB では、要素が深い入れ子になっていても、リスト内の個々の要素を操作できます。詳細については、「DynamoDB での式の使用」を参照してください。

マッピング

マップ型属性は、順序なしの名前と値のペアのコレクションを保存できます。マップは中括弧で囲まれます: `{ ... }`

マップは、JSON オブジェクトと同様です。マップの要素に保存できるデータ型に制限はなく、マップの要素が同じ型である必要はありません。

マップは DynamoDB で JSON ドキュメントを保存するのに最適です。以下の例は、文字列、数値、別のマップを含む入れ子のリストを要素とするマップを示しています。

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

Note

DynamoDB では、要素が深い入れ子になっていても、マップ内の個々の要素を操作できます。詳細については、「DynamoDB での式の使用」を参照してください。

セット

DynamoDB は、数値、文字列、またはバイナリ値のセットを表す型をサポートします。セット内の要素はすべて、同じ型である必要があります。例えば、数値セットには数値のみを含めることができ、文字列セットには文字列のみを含めることができます。

値を含む項目が DynamoDB のサイズ制限 (400 KB) 内である限り、セットの値の最大数の制限はありません。

設定内の各値は一意である必要があります。設定内の値の順序は保持されません。したがって、アプリケーションは、設定内の要素の特定の順序に依存することはできません。DynamoDB は空のセットをサポートしていませんが、セット内の空の文字列とバイナリ値は許可されます。

次の例では、文字列セット、設定された numbet とバイナリ設定を示します:

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

データ型記述子

低レベルの DynamoDB API プロトコルは、各属性を解釈する方法を DynamoDB に伝えるトークンとして、データ型記述子を使用します。

DynamoDB データ型記述子の一覧を次に示します。

- **S** — 文字列

- **N** — 数値
- **B** — バイナリ
- **BOOL** — ブール
- **NULL** — Null
- **M** — マップ
- **L** — リスト
- **SS** — 文字列セット
- **NS** — 数値セット
- **BS** — バイナリセット

テーブルクラス

DynamoDB には、コストの最適化に役立つように設計された 2 つのテーブルクラスが用意されています。DynamoDB 標準テーブルクラスがデフォルトで、大半のワークロードで推奨されています。DynamoDB Standard-Infrequent Access (DynamoDB 標準-IA) テーブルクラスは、ストレージが主要なコストとなるテーブル用に最適化されています。例えば、アプリケーションログ、古いソーシャルメディアの投稿、e コマースの注文履歴、過去のゲーム実績など、アクセス頻度の低いデータを格納するテーブルは、標準-IA テーブルクラスの候補として適しています。料金の詳細については、「Amazon DynamoDB の料金表」を参照してください。

すべての DynamoDB テーブルは、テーブルクラス (デフォルトでは DynamoDB Standard) に関連付けられます 。テーブルに関連付けられたすべてのセカンダリインデックスは、同じテーブルクラスを使用します。各テーブルクラスで、データストレージと読み込み/書き込みリクエストに対して異なる料金が適用されます。ストレージとスループットの使用パターンに基づいて、テーブルに対して最も費用対効果の高いテーブルクラスを選択できます。

テーブルクラスの選択は永続的ではありません。この設定は、AWS Management Console、AWS CLI、または AWS SDK を使用して変更できます。DynamoDB は、シングルリージョンテーブルとグローバルテーブルに対して AWS CloudFormation を使用したテーブルクラスの管理もサポートしています。テーブルクラスの選択の詳細については、「テーブルクラスを選択する場合の考慮事項」を参照してください。

パーティションとデータ分散

Amazon DynamoDB は、データをパーティションに保存します。パーティションは、ソリッドステートドライブ (SSD) によってバックアップされ、AWS リージョン内の複数のアベイラビリティー

ゾーン間で自動的にレプリケートされる、テーブル用のストレージの割り当てです。パーティション管理は DynamoDB によって完全に処理されます。パーティションをご自身が管理する必要はありません。

テーブルを作成するときに、テーブルの最初のステータスは CREATING になります。このフェーズの間に、DynamoDB はテーブルに十分なパーティションを割り当て、プロビジョンされたスループット要件に対応できるようにします。テーブルのステータスが ACTIVE に変わったらテーブルデータの書き込みと読み取りを開始できます。

DynamoDB は次の状況でテーブルに追加のパーティションを割り当てます。

- テーブルのプロビジョニングされたスループット設定を、既存のパーティションがサポートできる以上に増やした。
- 既存のパーティションが容量いっぱいになり、より多くのストレージ領域が必要になった。

パーティション管理は自動的にバックグラウンドで自動的に発生し、アプリケーションに対して透過的です。テーブルは利用可能な状態のままで、プロビジョニングされたスループット要件を完全にサポートします。

詳細については、パーティションキーの設計を参照してください。

DynamoDB のグローバルセカンダリインデックスもパーティションで構成されます。グローバルセカンダリインデックスのデータは、基本テーブルのデータとは別に保存されますが、インデックスパーティションはテーブルパーティションと同様に動作します。

データ分散: パーティションキー

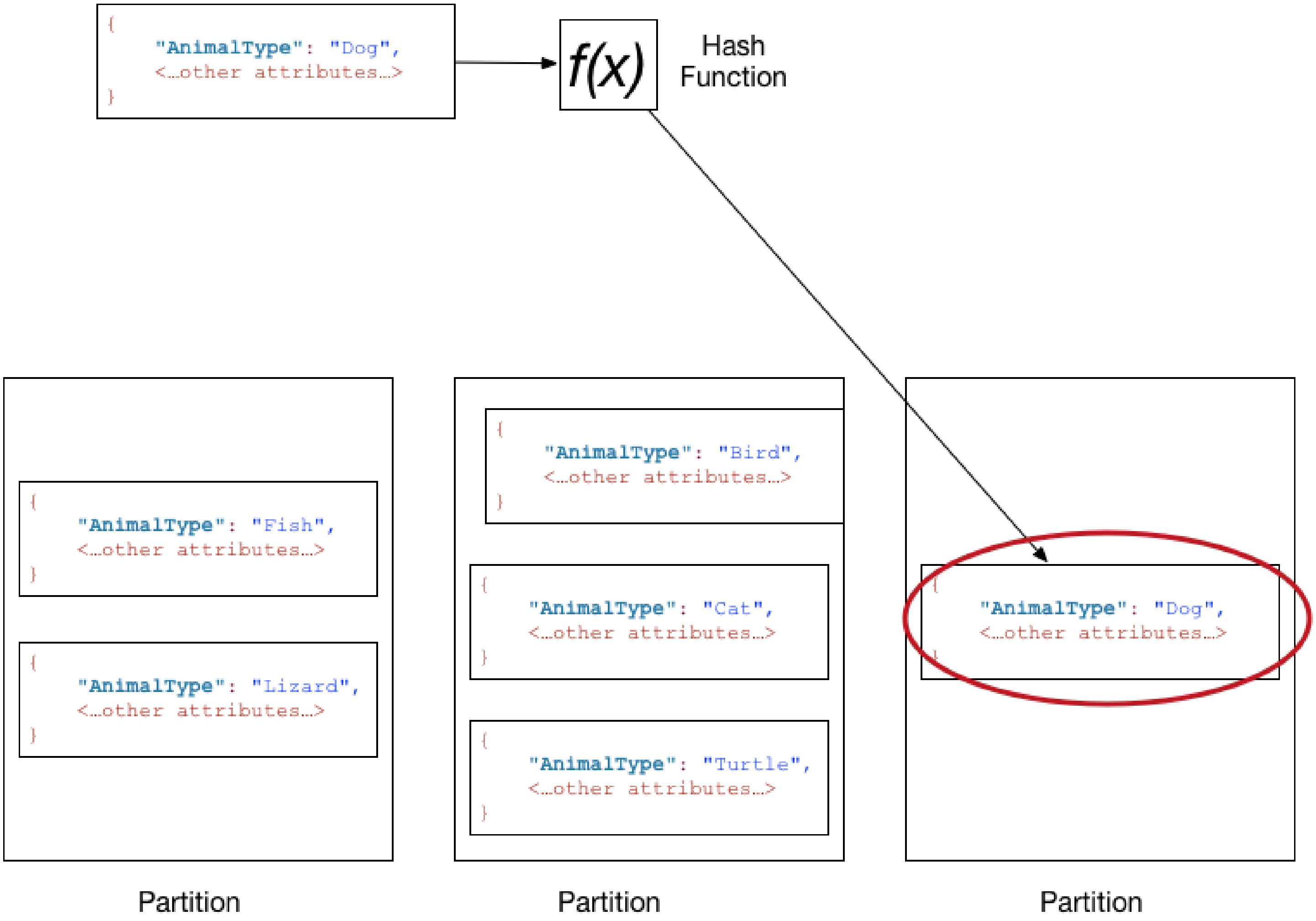
テーブルにシンプルなプライマリキー (パーティションキーのみ) がある場合、DynamoDB はパーティションキーバリューに基づいて、各項目を保存および取得します。

DynamoDB は項目をテーブルに書き込むため、パーティションキーバリューを内部ハッシュ関数への入力として使用します。ハッシュ関数からの出力値によって、項目が保存されるパーティションが決まります。

テーブルから項目を読み込むには、項目のパーティションキーバリューを指定する必要があります。DynamoDB はこの値をハッシュ関数への入力として使用し、項目が見つかるパーティションを提供します。

次の図は、複数のパーティションにまたがる Pets という名前のテーブルを示しています。テーブルのプライマリキーは AnimalType (このキー属性のみが表示されます。) DynamoDB は、ハッシュ関

数を使用して、新しい項目を保存する場所を決定します。この場合は、文字列 Dog のハッシュ値に基づいています。項目はソート順に保存されないことに注意してください。各アイテムの場所は、そのパーティションキーのハッシュ値によって決まります。



> **Note**
>
> DynamoDB は、パーティション数にかかわらず、項目がテーブルのパーティション全体に渡って均一にディストリビューションされている状態に対して最適化されています。テーブルの項目数に対して大きな個別の値を持つことができるパーティションキーを選択することをお勧めします。

データ分散: パーティションキーおよびソートキー

テーブルに複合プライマリキー (パーティションキーとソートキー) がある場合、DynamoDB は データ分散: パーティションキー で説明したのと同じ方法でパーティションキーのハッシュ値を計算しま

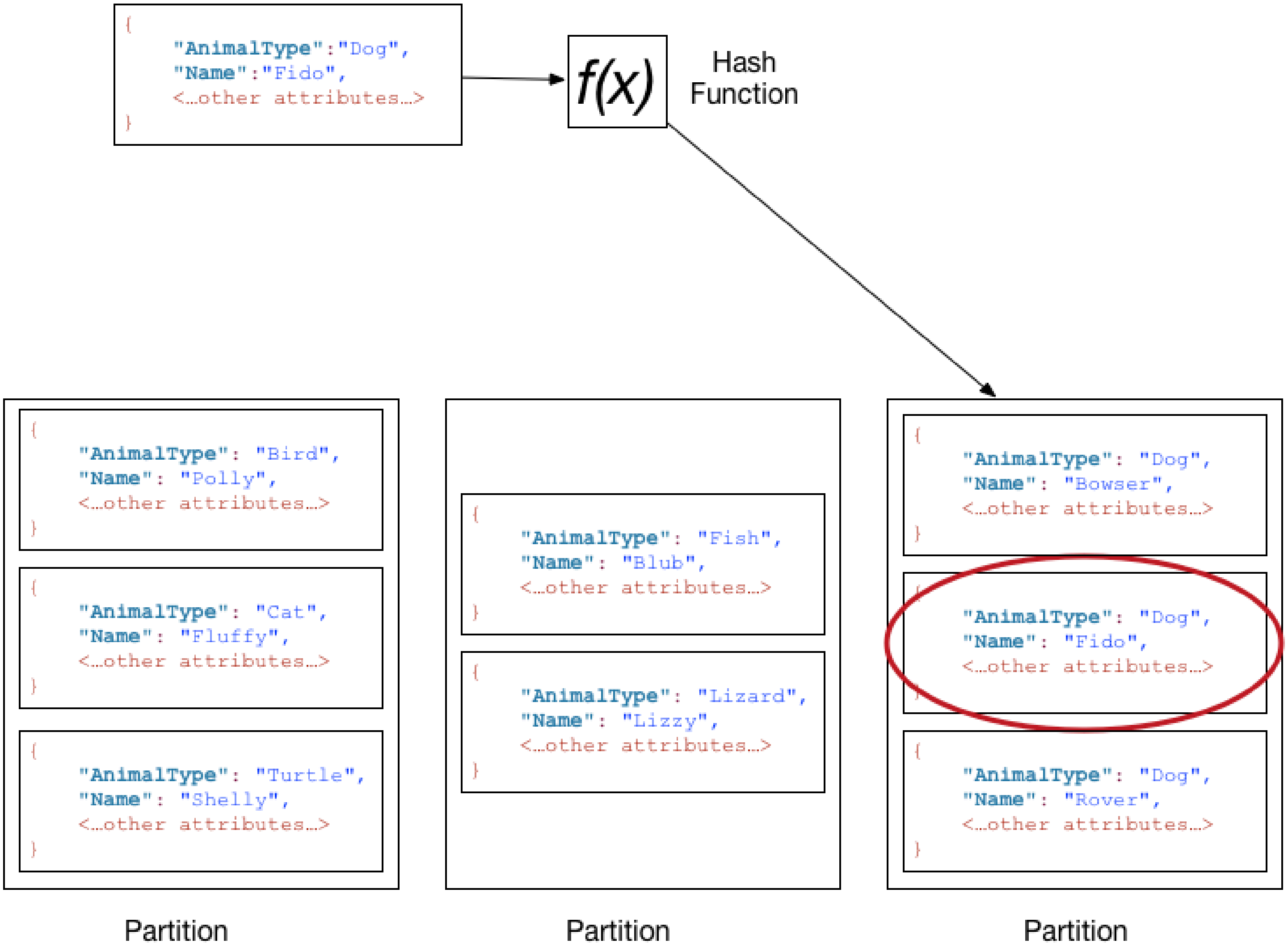
す。ただし、パーティションキーの値が同じ項目は互いに近く、ソートキー属性の値によってソートされた順序になる傾向があります。パーティションキー属性の値が同じ項目のセットは、項目コレクションと呼ばれます。項目コレクションは、コレクション内の項目の範囲を効率的に取得できるように最適化されています。テーブルにローカルセカンダリインデックスがない場合、DynamoDB は、データを保存し、読み取りと書き込みのスループットを実現するために必要な数のパーティションに自動的に項目コレクションを分割します。

テーブルに項目を書き込むため、DynamoDB はパーティションキーのハッシュ値を計算し、項目を含めるパーティションを決定します。そのパーティションでは、いくつかの項目で同じパーティションキー値を持つことができます。そのため、DynamoDB には、同じパーティションキーを持つ他の項目とソートキーの昇順で項目が保存されます。

テーブルから項目を読み込むには、パーティションのキーバリューとソートキーのキーバリューを指定する必要があります。DynamoDB は、パーティションキーのハッシュ値を計算し、項目が見つかるパーティションを提供します。

目的の項目に同じパーティションキーバリューがある場合、単一のオペレーション (`Query`) でテーブルから複数の項目を読み取ることができます。DynamoDB は、そのパーティションのキーバリューを持つすべての項目を返します。オプションでソートキーに条件を適用し、特定範囲内の値を持つ項目だけを返すことができます。

Pets テーブルに、AnimalType (パーティションキー) と Name (ソートキー) で構成される複合プライマリキーがあるとします。次の図表は、DynamoDB がパーティションのキーバリューが Dog でソートキー値が Fido の項目を書き込んでいるところを示しています。



Pets テーブルから同じ項目を読み込むために、DynamoDB は、ハッシュ値 Dog を計算し、これらの項目が保存されたパーティションを提供します。次に、DynamoDB は、Fido が見つかるまで、ソートキーの属性値をスキャンします。

AnimalType が Dog のすべての項目を読み込むには、ソートキーの条件を指定しないで `Query` オペレーションを発行できます。デフォルトでは、項目は保存されている順序 (つまり、ソートキーによって昇順でソート) で返されます。オプションで、代わりに降順をリクエストできます。

一部の Dog 項目のみをクエリするには、ソートキーに条件を適用できます (たとえば、Name が A から K の範囲内の文字で始まる Dog 項目のみ)。

> Note
>
> DynamoDB テーブルには、パーティションキーバリューごとに個別のソートキーバリューの数に上限はありません。何十億もの Dog 項目を Pets テーブルに保存する必要がある場合、DynamoDB はこの要件を自動的に処理するのに十分なストレージを割り当てます。

SQL から NoSQL へ

アプリケーションデベロッパーなら、リレーショナルデータベース管理システム (RDBMS) および構造化クエリ言語 (SQL) を使用した経験があるかもしれません。Amazon DynamoDB を使い始めると、多くの類似点があると同時に、異なる点も多くあることに気づきます。NoSQL は、可用性が高く、スケーラブルで、高パフォーマンス用に最適化された、非リレーショナルデータベースシステムについて説明するのに使用される用語です。NoSQL データベース (DynamoDB など) は、リレーショナルモデルの代わりに、キーバリューのペアやドキュメントストレージなど、データ管理のための代替モデルを使用します。詳細については、「NoSQL とは」を参照してください。

Amazon DynamoDB は PartiQL をサポートしています。PartiQL は、オープンソースの SQL 互換のクエリ言語で、データの格納場所や形式に関係なく、データへの効率的なクエリの実行が簡単に行えます。PartiQL を使用すると、リレーショナルデータベースの構造化データ、オープンデータ形式の半構造化データおよびネストされたデータ、さらには行ごとに異なる属性を許可する NoSQL またはドキュメントデータベース内のスキーマレスデータを簡単に処理できます。詳細については、「PartiQL Query Language」(PartiQL クエリ言語) を参照してください。

次のセクションでは、SQL ステートメントを同等の DynamoDB オペレーションと比較および対比しながら、一般的なデータベースタスクについて説明します。

> Note
>
> このセクションの SQL の例は、MySQL の RDBMS と互換性があります。
> このセクションの DynamoDB 例では、JSON 形式のオペレーションのパラメータと共に DynamoDB オペレーションの名前を表示します。これらのオペレーションを使用するコード例については、「DynamoDB および AWS SDK の使用開始」を参照してください。

トピック

- データベースの特徴
- テーブルの作成
- テーブルに関する情報の取得
- テーブルへのデータの書き込み
- テーブルからデータを読み込む場合の、SQL と DynamoDB の主な相違点
- インデックスの管理
- テーブルのデータ変更
- テーブルからデータを削除する
- テーブルの削除

リレーショナル (SQL) または NoSQL?

現在のアプリケーションには今までになく厳しい要件があります。たとえば、あるオンラインゲームを、小数のユーザーおよび非常に小さいデータ量で開始するかもしれません。しかし、ゲームが成功すれば、それは基盤となるデータベース管理システムのリソースを簡単に上回ります。ウェブベースのアプリケーションに、数百、数千、数百万の同時ユーザーがいて、テラバイトあるいはそれ以上の新しいデータが毎日生成される、というのはよくあることです。そのようなアプリケーションのデータベースの場合は、1 秒あたり数万 (あるいは数十万) の読み取り/書き込みの処理が必要です。

Amazon DynamoDB は、これらのワークロードに適しています。デベロッパーとして、アプリケーションを小さく開始し、人気が出るにつれて徐々に増加させることができます。DynamoDB は、大量のデータや多数のユーザーの処理をシームレスにスケーリングします。

従来のリレーショナルデータベースモデリングと DynamoDB に適応させる方法の詳細については、「DynamoDB でリレーショナルデータをモデル化するためのベストプラクティス」を参照してください。

次の表に、リレーショナルデータベース管理システム (RDBMS) と DynamoDB の高度な相違点を示します。

特徴	リレーショナルデータベース管理システム (RDBMS)	Amazon DynamoDB
最適なワークロード	アドホッククエリ、データウェアハウス、OLAP (オンライン分析処理)。	ソーシャルネットワーク、ゲーム、メディア共有、Internet of Things (IoT) を含む、

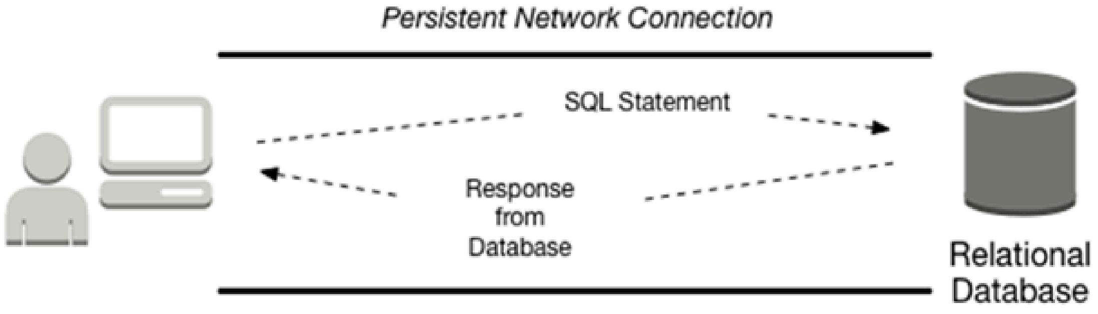
特徴	リレーショナルデータベース管理システム (RDBMS)	Amazon DynamoDB
		ウェブスケールアプリケーションです。
データモデル	リレーショナルモデルには、データを、テーブル、行、列に正規化する、明確に定義されたスキーマが必要です。さらに、関係のすべては、テーブル、列、インデックスおよび他のデータベースのエレメント間で定義されます。	DynamoDB はスキーマレスです。すべてのテーブルに、各データ項目を一意に識別するプライマリキーが必要ですが、他の非キー属性のような制約はありません。DynamoDB では、JSON ドキュメントを含む構造化データまたは半構造化データを管理できます。
データアクセス	SQL は、データを保存および取得するための基準です。リレーショナルデータベースにはデータベース駆動型アプリケーションの開発を簡素化するためのツールが豊富ですが、これらのツールはすべて、SQL を使用します。	AWS Management Console、AWS CLI、または NoSQL WorkBench を使用して DynamoDB を操作し、アドホックタスクを実行できます。PartiQL は SQL 互換のクエリ言語であり、DynamoDB でデータの選択、挿入、更新、および削除を行うことができます。アプリケーションでは、オブジェクトベースでドキュメント中心、または低レベルのインターフェイスを使用しながら、AWS ソフトウェア開発キット (SDK) を活用し、DynamoDB を操作できます。

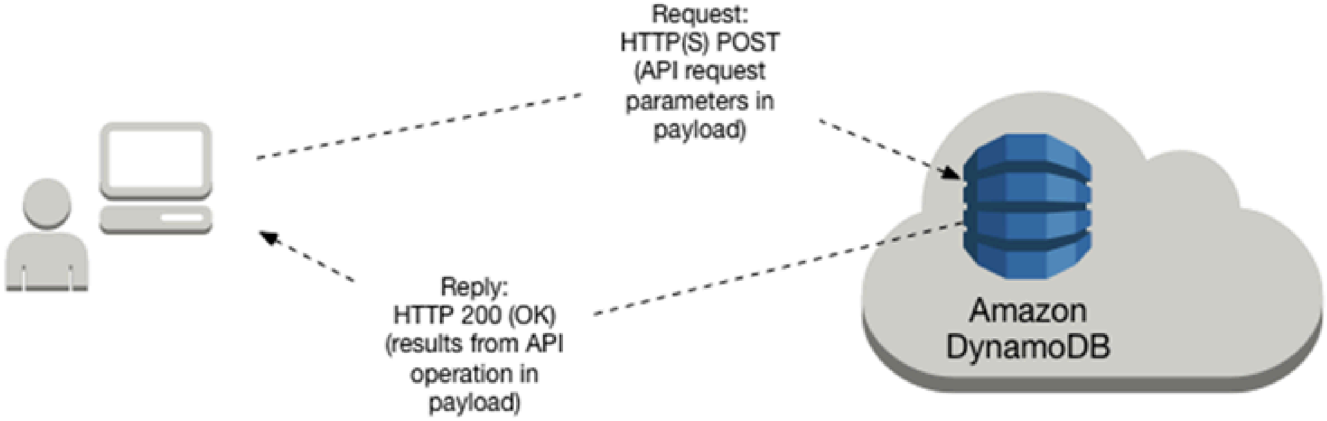
特徴	リレーショナルデータベース管理システム (RDBMS)	Amazon DynamoDB
パフォーマンス	リレーショナルデータベースは、ストレージに最適化されていますので、パフォーマンスは通常、ディスクサブシステムによって異なります。デベロッパーとデータベース管理者は、ピークパフォーマンスを達成するために、クエリ、インデックスおよびテーブルの構造を最適化する必要があります。	DynamoDB は、コンピューティングに最適化されていますので、パフォーマンスは主に、基盤となるハードウェアとネットワークレイテンシーの機能です。マネージドサービスとして、DynamoDB は、これらの実装の詳細からアプリケーションを隔離し、堅牢で高パフォーマンスなアプリケーションの設計と構築に集中できるようにします。
スケーリング	より高速なハードウェアでスケールアップするのが最も簡単です。データベーステーブルが分散システムの複数のホストにまたがることは可能ですが、この場合、追加の投資が必要です。リレーショナルデータベースは、スケーラビリティに上限を課すファイルの数とサイズが最大サイズです。	DynamoDB は、ハードウェアの分散型クラスターを使用してスケールアウトできるように設計されています。この設計により、レイテンシーの増加なしでスループットを強化することができます。顧客がスループット要件を指定すると、DynamoDB は、その要件に対応するために十分なリソースを割り当てます。テーブル単位の項目数、およびそのテーブルの合計サイズに上限はありません。

データベースの特徴

アプリケーションがデータベースにアクセスする前に、アプリケーションがデータベースの使用を許可されるように認証する必要があります。アプリケーションで、許可されているアクションのみ実行できるように、許可する必要があります。

次の図表は、クライアントのリレーショナルデータベースおよび Amazon DynamoDB とのやり取りを示します。





次の表にクライアントのやり取りのタスクについての詳細があります。

特徴	リレーショナルデータベース管理システム (RDBMS)	Amazon DynamoDB
データベースにアクセスするためのツール	ほとんどのリレーショナルデータベースは、コマンドラインインターフェイス (CLI) を提供しており、アドホックな SQL ステートメントを入力して、結果をすぐに見ることができます。	ほとんどの場合、アプリケーションコードを書き込みます。AWS Management Console、AWS Command Line Interface (AWS CLI)、または NoSQL Workbench を使用して、アドホックリクエストを DynamoDB に送信し、結果を表示することもできま

特徴	リレーショナルデータベース管理システム (RDBMS)	Amazon DynamoDB
		す。PartiQL は SQL 互換のクエリ言語であり、DynamoDB でデータの選択、挿入、更新、および削除を行うことができます。
データベースに接続	アプリケーションプログラムは、データベースを使用したネットワーク接続を確立し、維持します。アプリケーションが終了すると、接続を終了します。	DynamoDB は、ウェブサービスで、その操作はステートレスです。アプリケーションは永続的ネットワーク接続を維持する必要はありません。代わりに、DynamoDB の操作は HTTP(S) リクエストおよびレスポンスを使用して行われます。
認証	アプリケーションが認証されるまでデータベースに接続できません。RDBMS は認証自体を実行できますし、ホストのオペレーティングシステムやディレクトリサービスにこのタスクをオフロードすることもできます。	DynamoDB に対するすべてのリクエストは、その特定のリクエストを認証する暗号署名と共に使用しなければなりません。AWS SDK は、署名の作成とリクエストの署名に必要なすべての論理を提供します。詳細については、「AWS 全般のリファレンス.」の「AWS API リクエストの署名」を参照してください。

特徴	リレーショナルデータベース管理システム (RDBMS)	Amazon DynamoDB
認証	アプリケーションは承認されたアクションのみ実行できます。データベース管理者またはアプリケーション所有者は、SQL GRANT および REVOKE ステートメントを使用して、データベースオブジェクト (テーブルなど)、データ (テーブル内の行など)、特定の SQL ステートメントを発行する機能へのアクセスを制御できます。	DynamoDB では、AWS Identity and Access Management (IAM) が承認を行います。DynamoDB リソース (テーブルなど) へのアクセス許可を付与する IAM ポリシーを記述し、ユーザーとロールがそのポリシーを使用できるようにします。IAM では、DynamoDB テーブルの個々のデータ項目に対する詳細なアクセス制御といった特徴も備えています。詳細については、「Amazon DynamoDB の Identity and Access Management」を参照してください。
リクエストを送信	アプリケーションは、実行するすべてのデータベース操作に対する SQL ステートメントを発行します。SQL ステートメントを受信すると、RDBMS は構文を確認し、オペレーションを実行するための計画を作成してから、計画を実行します。	アプリケーションは、HTTP(S) リクエストを DynamoDB に送信します。リクエストには、パラメータとともに、実行する DynamoDB オペレーションの名前が含まれます。DynamoDB はリクエストを直ちに実行します。

特徴	リレーショナルデータベース管理システム (RDBMS)	Amazon DynamoDB
レスポンスを受信	RDBMS は SQL ステートメントから結果を返します。エラーがある場合は、RDBMS はエラー状況とエラーメッセージを返します。	DynamoDB は、オペレーションの結果を含む HTTP(S) レスポンスを返します。エラーがあると、DynamoDB は、HTTP エラー状況およびエラーメッセージを返します。

テーブルの作成

テーブルは、リレーショナルデータベースおよび Amazon DynamoDB の基本的なデータ構造です。リレーショナルデータベース管理システム (RDBMS) では、作成時に、テーブルのスキーマを定義する必要があります。対照的に、DynamoDB テーブルはスキーマレスであるため、プライマリキーを除いて、テーブルを作成する際に、追加の属性やデータ型を定義する必要はありません。

次のセクションでは、SQL を使用してテーブルを作成する方法と、DynamoDB を使用してテーブルを作成する方法を比較します。

トピック

- SQL を使用してテーブルを作成する
- DynamoDB を使用してテーブルを作成する

SQL を使用してテーブルを作成する

SQL では、次の例に示すように、CREATE TABLE ステートメントを使用して、テーブルを作成します。

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

このテーブルのプライマリキーは、Artist および SongTitle で構成されます。

テーブルの列とデータ型すべて、およびテーブルのプライマリキーを定義する必要があります。(これらの定義は、ALTER TABLE ステートメントを使用して、必要に応じて後で変更することができます。)

多くの SQL 実装では、CREATE TABLE ステートメントの一部として、テーブルのストレージ仕様を定義することができます。他に明記されていない限り、テーブルはデフォルトのストレージ設定で作成されます。本稼働環境では、データベース管理者は最適なストレージパラメータを特定することもできます。

DynamoDB を使用してテーブルを作成する

CreateTable オペレーションを使用してプロビジョニングモードのテーブルを作成し、次に示すように、パラメータを指定します。

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH" //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
    ],
    ProvisionedThroughput: {       // Only specified if using provisioned mode
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

このテーブルのプライマリキーは、Artist (パーティションキー) および SongTitle (ソートキー) で構成されています。

CreateTable に以下のパラメータを提供する必要があります。

- TableName – テーブルの名前。
- KeySchema – プライマリキーに使用する属性。詳細については、「テーブル、項目、属性」および「プライマリキー」を参照してください。
- AttributeDefinitions – キースキーマ属性のデータ型。
- ProvisionedThroughput (for provisioned tables) – このテーブルに必要な 1 秒あたりの読み取り/書き込み数。DynamoDB は、スループット要件を常に満たすように、十分なストレージとシステムリソースを確保しています。これらは、UpdateTable オペレーションを使用し、必要に応じて後で変更できます。DynamoDB がストレージ割り当てを完全に管理しているため、テーブルのストレージ要件を指定する必要はありません。

> Note
>
> CreateTable を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

テーブルに関する情報の取得

テーブルが仕様に従って作成されたことを確認できます。リレーショナルデータベースでは、テーブルのスキーマがすべて表示されます。Amazon DynamoDB テーブルはスキーマレスであるため、プライマリキー属性のみが表示されます。

トピック

SQL を使ってテーブルに関する情報を取得する

ほとんどのリレーショナルデータベース管理システム (RDBMS) では、テーブルの構造 (列、データ型、プライマリキー定義など) を記述できます。SQL にはこれを行うための標準的な方法はありません。ただし、データベースシステムの多くが DESCRIBE コマンドを提供しています。MySQL の例を以下に示します。

```
DESCRIBE Music;
```

すべての列名、データ型、サイズを含むテーブルの構造が返ります。

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Artist     | varchar(20) | NO   | PRI | NULL    |       |
| SongTitle  | varchar(30) | NO   | PRI | NULL    |       |
| AlbumTitle | varchar(25) | YES  |     | NULL    |       |
| Year       | int(11)     | YES  |     | NULL    |       |
| Price      | float       | YES  |     | NULL    |       |
| Genre      | varchar(10) | YES  |     | NULL    |       |
| Tags       | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

このテーブルのプライマリキーは、Artist および SongTitle で構成されます。

DynamoDB でテーブルに関する情報を取得する

DynamoDB には、同様の DescribeTable オペレーションがあります。パラメータは、テーブル名のみです。

```
{
    TableName : "Music"
}
```

DescribeTable からの返信は、次のようになります。

```
{
  "Table": {
    "AttributeDefinitions": [
      {
            "AttributeName": "Artist",
            "AttributeType": "S"
        },
        {
            "AttributeName": "SongTitle",
            "AttributeType": "S"
        }
    ],
    "TableName": "Music",
    "KeySchema": [
        {
            "AttributeName": "Artist",
            "KeyType": "HASH"  //Partition key
        },
        {
            "AttributeName": "SongTitle",
            "KeyType": "RANGE"  //Sort key
        }
    ],

    ...
```

DescribeTable は、テーブルのインデックス、プロビジョニングされたスループット設定、概算項目数、その他メタデータに関する情報を返します。

テーブルへのデータの書き込み

リレーショナルデータベースのテーブルには、データの行が含まれます。行は列で構成されます。Amazon DynamoDB テーブルには、項目が含まれています。項目は属性で構成されます。

このセクションでは、テーブルに 1 つの行 (または項目) を書き込む方法を説明します。

トピック

- SQL を使ってテーブルにデータを書き込む
- DynamoDB のテーブルにデータを書き込む

SQL を使ってテーブルにデータを書き込む

リレーショナルデータベースのテーブルは、行と列で構成される、2 つのディメンションのデータ構造です。一部のデータベース管理システムは、通常、ネイティブ JSON または XML データ型を使用する半構造化データのサポートを提供します。ただし、実装の詳細はベンダー間で変わります。

SQL では、INSERT ステートメントを使用して、テーブルに行を追加します。

```
INSERT INTO Music
    (Artist, SongTitle, AlbumTitle,
    Year, Price, Genre,
    Tags)
VALUES(
    'No One You Know', 'Call Me Today', 'Somewhat Famous',
    2015, 2.14, 'Country',
    '{"Composers": ["Smith", "Jones", "Davis"],"LengthInSeconds": 214}'
);
```

このテーブルのプライマリキーは、Artist と SongTitle で構成されます。これらの列の値を指定する必要があります。

> **Note**
>
> この例では、Tags 列を使用して、Music テーブル内の曲に関する半構造化データを保存します。Tags 列は TEXT 型として定義され、MySQL で最大 65,535 文字を保存できます。

DynamoDB のテーブルにデータを書き込む

Amazon DynamoDB では、DynamoDB API または PartiQL (SQL 互換のクエリ言語) を使用して、テーブルに項目を追加できます。

DynamoDB API

DynamoDB API では、PutItem オペレーションを使用して、テーブルに項目を追加します。

```
{
    TableName: "Music",
    Item: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today",
        "AlbumTitle":"Somewhat Famous",
        "Year": 2015,
        "Price": 2.14,
        "Genre": "Country",
        "Tags": {
            "Composers": [
                    "Smith",
                    "Jones",
                    "Davis"
                ],
                "LengthInSeconds": 214
            }
        }
}
```

このテーブルのプライマリキーは、Artist および SongTitle で構成されます。これらの属性の値を指定する必要があります。

この PutItem 例に関する主要事項を示します。

- DynamoDB では、JSON を使用して、ドキュメントのネイティブサポートを提供しています。これにより DynamoDB は Tags などの半構造化データを保存する場合に最適になります。また、JSON ドキュメント内からデータを取得および操作できます。
- Musicテーブルには、プライマリキー (Artist および SongTitle) 以外には事前定義された属性はありません。
- ほとんどの SQL データベースはトランザクション指向です。INSERT ステートメントを発行すると、データ変更は、COMMIT ステートメントを発行するまで永続的ではありません。Amazon DynamoDB によって、PutItem オペレーションの効果は、DynamoDB が HTTP 200 ステータスコード (OK) で応答する場合、永続的になります。

> Note
>
> PutItem を使用したコード例については、「DynamoDB および AWS SDK の使用開始」を参照してください。

次に、PutItem の他の例をいくつか示します。

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "My Dog Spot",
        "AlbumTitle":"Hey Now",
        "Price": 1.98,
        "Genre": "Country",
        "CriticRating": 8.4
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "Somewhere Down The Road",
        "AlbumTitle":"Somewhat Famous",
        "Genre": "Country",
        "CriticRating": 8.4,
        "Year": 1984
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Still In Love",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 2.47,
        "Genre": "Rock",
        "PromotionInfo": {
            "RadioStationsPlaying":[
                 "KHCR", "KBQX", "WTNR", "WJJH"
            ],
            "TourDates": {
                "Seattle": "20150625",
                "Cleveland": "20150630"
            },
            "Rotation": "Heavy"
        }
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Look Out, World",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 0.99,
        "Genre": "Rock"
    }
}
```

ⓘ Note

PutItem に加えて、DynamoDB は、同時に複数の項目に書き込むための BatchWriteItem オペレーションもサポートします。

PartiQL for DynamoDB

PartiQL では、PartiQL ExecuteStatement ステートメントを利用する Insert オペレーションを使用して、テーブルに項目を追加します。

```
INSERT into Music value {
    'Artist': 'No One You Know',
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

このテーブルのプライマリキーは、Artist および SongTitle で構成されます。これらの属性の値を指定する必要があります。

ⓘ Note

Insert と ExecuteStatement を使用したコード例については、DynamoDB 用の PartiQL 挿入ステートメント を参照してください。

テーブルからデータを読み込む場合の、SQL と DynamoDB の主な相違点

SQL では、SELECT ステートメントを使用して、テーブルから 1 つ以上の行を取得します。WHERE 句を使用して、返されたデータを調べます。

これは、Amazon DynamoDB を使用する場合とは異なります。Amazon DynamoDb は、データの読み取りに次のオペレーションを提供します。

- `ExecuteStatement` がテーブルから 1 つまたは複数の項目を取得します。`BatchExecuteStatement` が 1 回のオペレーションで異なるテーブルから複数の項目を取得します。どちらのオペレーションでも、SQL 互換のクエリ言語である PartiQL が使用されます。
- `GetItem` – テーブルから単一の項目を取り出します。これは、項目の物理的な場所に直接アクセスできるため、単一の項目を読み込むうえで最も効率的な方法です。(DynamoDB では、`BatchGetItem` オペレーションを提供し、1 回のオペレーションで最大 100 回の `GetItem` コールを実行できます。)
- `Query` – 特定のパーティションキーがあるすべての項目を取得します。これらの項目内では、ソートキーに条件を適用し、データのサブセットのみ取得できます。`Query` は、データが格納されているパーティションに、すばやく効率的にアクセスできます。（詳しくは、パーティションとデータ分散 を参照してください）。
- `Scan` – 指定されたテーブルで、すべての項目を取得します。(大量のリソースシステムを消費するため、このオペレーションは大きなテーブルでは使用しないでください。)

> Note
>
> リレーショナルデータベースでは、SELECT ステートメントを使用して、複数のテーブルからデータを結合し、その結果を返すことができます。結合は、リレーショナルモデルにおいて必須です。結合を効率的に実行するために、データベースおよびアプリケーションは、継続的にパフォーマンスを調整する必要があります。DynamoDB は、テーブル結合をサポートしない非リレーショナル NoSQL データベースです。その代わり、アプリケーションは一度に 1 つのテーブルからデータを読み込みます。

次のセクションでは、データを読み込むための異なるユースケースおよびこれらのタスクをリレーショナルデータベースと DynamoDB を使って実行する方法を説明します。

トピック

プライマリキーを使用して項目を読み込む

データベースの一般的なアクセスパターンの 1 つは、テーブルから単一項目を読み取ることです。目的の項目のプライマリキーを指定する必要があります。

トピック

- SQL にプライマリキーを使用して項目を読み込む
- DynamoDB でプライマリキーを使用して項目を読み込む

SQL にプライマリキーを使用して項目を読み込む

SQL では、SELECT ステートメントを使用して、テーブルからデータを取得します。結果の 1 つ以上の列 (* オペレーター を使用すれば、すべて) をリクエストできます。WHERE 句は返る行を判別します。

以下は、Music テーブルから単一の行を取得するための SELECT ステートメントです。WHERE 句はプライマリキー値を指定します。

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

列のサブセットのみを取得するようにこのクエリを変更できます。

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

このテーブルのプライマリキーが、Artist および SongTitle で構成されていることに注意してください。

DynamoDB でプライマリキーを使用して項目を読み込む

Amazon DynamoDB では、DynamoDB API または PartiQL (SQL 互換のクエリ言語) を使用して、テーブルから項目を読み込むことができます。

DynamoDB API

DynamoDB API では、PutItem オペレーションを使用して、テーブルに項目を追加します。

DynamoDB は、プライマリキーにより項目を取得するための GetItem オペレーションを提供します。GetItem は、項目の物理的な場所への直接アクセスを提供するため非常に効率的です。（詳しくは、パーティションとデータ分散 を参照してください）。

デフォルトでは、GetItem は、すべての属性を含む項目全体を返します。

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

一部の属性のみが返されるように、ProjectionExpression パラメータを追加できます。

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

このテーブルのプライマリキーが、Artist および SongTitle で構成されていることに注意してください。

DynamoDB GetItem オペレーションは非常に効率的です。プライマリキー値を使用して、該当する項目の正確な格納場所を特定し、そこから直接取得します。SQL SELECT ステートメントは、プライマリキー値によって項目を取得する場合、同様に効率的です。

SQL SELECT ステートメントは、さまざまな種類のクエリとテーブルスキャンをサポートしています。DynamoDB は、テーブルに対するクエリの実行 および 表のスキャン で説明されている Query および Scan オペレーションと同様の機能を提供します。

SQL SELECT ステートメントは、テーブルの結合を実行でき、同時に複数のテーブルからデータを取得できます。データベーステーブルが正規化され、テーブル間の関係が明確である場合、結合は最も効果的です。ただし、1 つの SELECTステートメントであまりに多くのテーブルを結合すると、アプリケーションパフォーマンスが影響を受けます。データベースレプリケーション、

マテリアライズドビュー、またはクエリの書き換えを使用して、このような問題を回避できます。

DynamoDB は、非リレーショナルデータベースのため、テーブルの結合はサポートされません。リレーショナルデータベースから既存のアプリケーションを DynamoDB に移行する場合、結合の必要を排除するためデータモデルを非正規化する必要があります。

> **Note**
>
> GetItem を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

PartiQL for DynamoDB

PartiQL では、PartiQL Select ステートメントを利用する ExecuteStatement オペレーションを使用して、テーブルから項目を読みます。

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

このテーブルのプライマリキーが、Artist および SongTitle で構成されていることに注意してください。

> **Note**
>
> 選択 PartiQL ステートメントは、DynamoDB テーブルにクエリやスキャンを実行する場合にも使用できます

Select と ExecuteStatement を使用したコード例については、DynamoDB 用の PartiQL select ステートメント を参照してください。

テーブルに対するクエリの実行

もう 1 つの一般的なアクセスパターンは、クエリ条件に基づき、テーブルから複数の項目を読み込むことです。

トピック

- SQL を使用してテーブルのクエリを実行する
- DynamoDB でテーブルのクエリを実行する

SQL を使用してテーブルのクエリを実行する

SQL を使用すると、SELECT ステートメントは、キー列、非キー列、または任意の組み合わせに対してクエリを実行できます。WHERE 句は、次の例に示すように、返る行を決定します。

```
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
/* Return all of the songs by an artist */

SELECT * FROM Music
WHERE Artist='No One You Know';
```

```
/* Return all of the songs by an artist, matching first part of title */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE 'Call%';
```

```
/* Return all of the songs by an artist, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

このテーブルのプライマリキーが、Artist および SongTitle で構成されていることに注意してください。

DynamoDB でテーブルのクエリを実行する

Amazon DynamoDB では、DynamoDB API または PartiQL (SQL 互換のクエリ言語) を使用して、テーブルの項目にクエリを実行できます。

DynamoDB API

Amazon DynamoDB では、Query オペレーションを使用すると、同様の方法でデータを取得できます。Query オペレーションは、データが保存されている物理的な場所にすばやく効率的にアクセスすることができます。詳細については、「パーティションとデータ分散」を参照してください。

Query は任意のテーブルまたはセカンダリインデックスでを使用できます。パーティションキーの値に対して等価条件を指定する必要があります。ソートキー属性が定義されている場合は、必要に応じて別の条件を指定できます。

KeyConditionExpression パラメータは、クエリを実行するキー値を指定します。オプションの FilterExpression を使用して、結果が返される前に、そこから特定の項目を削除できます。

DynamoDB では、式パラメータ (KeyConditionExpression や FilterExpression など) で ExpressionAttributeValues をプレースホルダーとして使用する必要があります。これは、ランタイム時に実際の値を SELECT ステートメントに代入する、リレーショナルデータベースでのバインド変数の使用に類似しています。

このテーブルのプライマリキーが、Artist および SongTitle で構成されていることに注意してください。

次に、DynamoDB Query の例をいくつか示します。

```
// Return a single song, by primary key

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

```
// Return all of the songs by an artist

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
    ExpressionAttributeValues: {
        ":a": "No One You Know"
    }
}
```

```
// Return all of the songs by an artist, matching first part of title

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call"
    }
}
```

> Note
>
> Query を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

PartiQL for DynamoDB

PartiQL では、パーティションキーの ExecuteStatement オペレーションと Select ステートメントを使用してクエリを実行できます。

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

このように SELECT ステートメントを使用すると、この特定の Artist に関連付けられているすべての曲が返されます。

Select と ExecuteStatement を使用したコード例については、DynamoDB 用の PartiQL select ステートメント を参照してください。

表のスキャン

SQL では、SELECT 句なしの WHERE ステートメントは、テーブルのすべての行を返します。Amazon DynamoDB では、Scan オペレーションで同様のアクションを行います。どちらの場合も、すべて、または一部の項目を取得できます。

SQL または NoSQL データベースのいずれを使用するにしても、スキャンは大量のシステムリソースを消費するので、控え目に使用する必要があります。スキャンが適切 (小さなテーブルをスキャンするなど) または不可避 (データの一括エクスポートの実行など) な場合はあります。しかし、一般的なルールとして、スキャンを実行しないようアプリケーションを設計する必要があります。詳細については、「DynamoDB のクエリオペレーション」を参照してください。

> **Note**
>
> 一括エクスポートを実行すると、パーティションごとに少なくとも 1 つのファイルも作成されます。各ファイル内のすべての項目は、その特定のパーティションのハッシュされたキースペースからのものです。

トピック

- SQL を使用してテーブルをスキャンする
- DynamoDB でテーブルをスキャンする

SQL を使用してテーブルをスキャンする

SQL を使用すると、SELECT 句を指定せずに、WHERE ステートメントを使用してテーブルをスキャンし、データのすべてを取得することができます。結果の 1 つ以上の列をリクエストできます。または、ワイルドカード文字「*」を使用する場合は、すべての項目をリクエストできます。

以下は SELECT ステートメントの使用例です。

```
/* Return all of the data in the table */
SELECT * FROM Music;
```

```
/* Return all of the values for Artist and Title */
SELECT Artist, Title FROM Music;
```

DynamoDB でテーブルをスキャンする

Amazon DynamoDB では、DynamoDB API または PartiQL (SQL 互換のクエリ言語) を使用して、テーブルに対してスキャンを実行します。

DynamoDB API

DynamoDB API では、テーブルまたはセカンダリインデックスの各項目にアクセスして、1 つまたは複数の項目または項目属性を返す Scan オペレーションを使用します。

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

Scan オペレーションでは、FilterExpression パラメータも指定できます。これを使用して、結果に表示しない項目を破棄することができます。FilterExpression は、スキャンの実行後、結果が返されるまでの間に適用されます。(これは、大きなテーブルではお勧めしません。返される一致項目がごく少数でも、Scan 全体に対して料金が請求されます。)

> **Note**
>
> Scan を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

PartiQL for DynamoDB

PartiQL では、Select ステートメントを使用するテーブルに対してすべての内容を返す ExecuteStatement オペレーションを使用してスキャンを実行します。

```
SELECT AlbumTitle, Year, Price
FROM Music
```

このステートメントは、Music テーブルのすべての項目を返すことに注意してください。

`Select` と `ExecuteStatement` を使用したコード例については、DynamoDB 用の PartiQL select ステートメント を参照してください。

インデックスの管理

インデックスは代替のクエリパターンへのアクセスを付与し、クエリを高速化できます。このセクションでは、SQL と Amazon DynamoDB におけるインデックスの作成と使用を比較します。

リレーショナルデータベースまたは DynamoDB のいずれを使用するにしても、インデックスの作成は慎重に判断する必要があります。テーブルに書き込みが発生するたびに、テーブルのインデックスはすべて、更新する必要があります。大きなテーブルでの書き込み量が多い環境では、大量のシステムリソースを消費する可能性があります。読み取り専用または読み取りがほとんどの環境では、これは大きな問題にはなりません。ただし、インデックスがアプリケーションによって実際に使用されており、ただ容量を取っていることがないよう確認する必要があります。

トピック

- インデックスの作成
- インデックスのクエリの実行およびスキャン

インデックスの作成

SQL の `CREATE INDEX` ステートメントを Amazon DynamoDB の `UpdateTable` オペレーションと比較します。

トピック

- SQL を使ってインデックスを作成する
- DynamoDB でインデックスを作成する

SQL を使ってインデックスを作成する

リレーショナルデータベースのインデックスは、テーブルの異なる列に対して高速にクエリを実行できるデータ構造です。`CREATE INDEX` SQL ステートメントを使用して、既存のテーブルにインデックスが追加し、インデックスを作成する列を指定します。インデックスの作成後は、通常どおりテーブルのデータにクエリを実行できますが、テーブル全体をスキャンする代わりに、テーブルの指定された行を迅速に検索するため、データベースがインデックスを使用することができます。

インデックスの作成後は、データベースが維持します。テーブルのデータを変更した場合は、インデックスが自動的に変更され、テーブル内の変更を反映します。

MySQL では、次のようなインデックスを作成します。

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

DynamoDB でインデックスを作成する

DynamoDB では、同様の目的のために、セカンダリインデックスを作成して使用できます。

DynamoDB のインデックスはリレーショナルな対応物とは異なります。セカンダリインデックスを作成する際、そのキー属性 (パーティションキーとソートキー) を指定する必要があります。セカンダリインデックスを作成したら、テーブルと同じように Query または Scan を実行できます。DynamoDB にはクエリオプティマイザーがないため、セカンダリインデックスは Query または Scan の場合にのみ使用されます。

DynamoDB では、次の 2 種類のインデックスをサポートしています。

- グローバルセカンダリインデックス – インデックスのプライマリキーは、テーブルからの任意の 2 つの属性になります。
- ローカルセカンダリインデックス – インデックスのパーティションキーは、テーブルのパーティションキーと同じである必要があります。ただし、ソートキーは、他の任意の属性にすることができます。

DynamoDB を使えば、セカンダリインデックスのデータは結果的にテーブルと整合性が取れます。テーブルまたはローカルセカンダリインデックスでの強い整合性を持つ Query または Scan オペレーションをリクエストできます。ただし、グローバルセカンダリインデックスは結果整合性のみをサポートします。

UpdateTable オペレーションを使用し、GlobalSecondaryIndexUpdates を指定して、既存のテーブルにグローバルセカンダリインデックスを追加できます。

```
{
    TableName: "Music",
    AttributeDefinitions:[
        {AttributeName: "Genre", AttributeType: "S"},
        {AttributeName: "Price", AttributeType: "N"}
    ],
    GlobalSecondaryIndexUpdates: [
        {
            Create: {
                IndexName: "GenreAndPriceIndex",
                KeySchema: [
                    {AttributeName: "Genre", KeyType: "HASH"}, //Partition key
                    {AttributeName: "Price", KeyType: "RANGE"}, //Sort key
                ],
                Projection: {
                    "ProjectionType": "ALL"
                },
                ProvisionedThroughput: {                                // Only
 specified if using provisioned mode
                    "ReadCapacityUnits": 1,"WriteCapacityUnits": 1
                }
            }
        }
    ]
}
```

UpdateTable に以下のパラメータを提供する必要があります。

- TableName – インデックスが関連付けられるテーブル。
- AttributeDefinitions – インデックスのキースキーマ属性用のデータ型。
- GlobalSecondaryIndexUpdates – 作成するインデックスに関する詳細。
  - IndexName – インデックスの名前。
  - KeySchema – インデックスのプライマリキーに使用する属性。
  - Projection – テーブルからインデックスにコピーされる属性。この場合、ALL は、すべての属性がコピーされることを意味します。
  - ProvisionedThroughput (for provisioned tables) – このインデックスに必要な 1 秒あたりの読み取り/書き込み数。(これは、テーブルのプロビジョニングされたスループット設定とは異なります。)

このオペレーションの一部は、テーブルから新しいインデックスにデータをバックフィリングすることを含みます。バックフィリング中、テーブルは使用可能なままになります。ただし、インデックスは Backfilling 属性が true から false に変わるまで、準備ができていません。DescribeTable オペレーションを使用して、この属性を表示できます。

Note

UpdateTable を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

インデックスのクエリの実行およびスキャン

SQL の SELECT ステートメントを使用したインデックスのクエリとスキャンを Amazon DynamoDB の Query および Scan オペレーションと比較します。

トピック

- SQL を使ってインデックスのクエリを実行してスキャンする
- DynamoDB でインデックスのクエリを実行してスキャンする

SQL を使ってインデックスのクエリを実行してスキャンする

リレーショナルデータベースでは、インデックスを直接使用しません。代わりに、SELECT ステートメントの発行により、テーブルでクエリを実行し、クエリオプティマイザは任意のインデックスを利用できます。

クエリオプティマイザは、使用できるインデックスを評価し、クエリを高速化するためにそれらを使用できるかを決定するリレーショナルデータベース管理システム (RDBMS) のコンポーネントです。クエリを高速化するためにインデックスが使用できる場合、RDBMS は最初にインデックスにアクセスし、それを使用してテーブルのデータを特定します。

GenreAndPriceIndex を使用してパフォーマンスを向上できる SQL ステートメントを次に示します。ここでは、Music テーブルに十分なデータがあり、クエリオプティマイザが、テーブル全体にスキャンするのではなく、このインデックスを使用することを前提にします。

```
/* All of the rock songs */

SELECT * FROM Music
WHERE Genre = 'Rock';
```

```
/* All of the cheap country songs */

SELECT Artist, SongTitle, Price FROM Music
```

```
WHERE Genre = 'Country' AND Price < 0.50;
```

DynamoDB でインデックスのクエリを実行してスキャンする

DynamoDB では、テーブルに対して実行するのと同じように、インデックスに対して直接 `Query` および `Scan` オペレーションを実行します。DynamoDB API または [PartiQL](SQL 互換のクエリ言語) を使用して、インデックスにクエリやスキャンを実行できます。`TableName` と `IndexName` の両方を指定する必要があります。

以下は、DynamoDB の GenreAndPriceIndex に対するいくつかのクエリです。(このインデックスのキースキーマは、ジャンルと価格で構成されています。)

DynamoDB API

```
// All of the rock songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

この例では `ProjectionExpression` を使用して、属性すべてではなく、一部のみを結果に表示することを示します。

```
// All of the cheap country songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre and Price < :price",
    ExpressionAttributeValues: {
        ":genre": "Country",
        ":price": 0.50
    },
    ProjectionExpression: "Artist, SongTitle, Price"
};
```

次に示すのは、GenreAndPriceIndex に対するスキャンです。

```
// Return all of the data in the index

{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

PartiQL では、Select ステートメントを使用して、インデックスに対してクエリとスキャンを実行します。

```
// All of the rock songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock'
```

```
// All of the cheap country songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock' AND Price < 0.50
```

次に示すのは、GenreAndPriceIndex に対するスキャンです。

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> Note
>
> Select を使用したコード例については、「DynamoDB 用の PartiQL select ステートメント」を参照してください。

テーブルのデータ変更

SQL 言語は、データを変更するための UPDATE ステートメントを提供します。Amazon DynamoDB は、この UpdateItem オペレーションを使用して同様のタスクを実行します。

トピック

- SQL を使用してテーブルのデータを変更する
- DynamoDB でテーブルのデータを変更する

SQL を使用してテーブルのデータを変更する

SQL では、UPDATE ステートメントを使用して、1 つ以上の行を変更します。SET 句は、1 つ以上の行に新しい値を指定し、WHERE 句は変更する行を決定します。以下はその例です。

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

WHERE 句に行が一致しない場合、UPDATE ステートメントは何も実行しません。

DynamoDB でテーブルのデータを変更する

DynamoDB では、クラシック API または PartiQL (SQL 互換のクエリ言語) を使用して、単一の項目を変更できます。複数の項目を変更する場合は、複数のオペレーションを使用する必要があります。

DynamoDB API

DynamoDB API では、UpdateItem オペレーションを使用して、単一の項目を変更します。

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

属性値を指定するには、変更する項目の Key 属性および UpdateExpression を指定する必要があります。UpdateItem は、「アップサート」オペレーションのように動作します。テーブルに項目が存在する場合は更新され、存在しない場合は新しい項目が追加 (挿入) されます。

UpdateItem は、条件付き書き込みをサポートしており、特定の ConditionExpression が true と評価された場合のみ、オペレーションが成功します。たとえば、次の UpdateItem オペレーションは、曲の価格が 2.00 以上でない限り、更新を実行しません。

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ConditionExpression: "Price >= :p",
    ExpressionAttributeValues: {
        ":label": "Global Records",
        ":p": 2.00
    }
}
```

また、UpdateItem はアトミックカウンター、または増加減少する Number 型の属性をサポートしています。アトミックカウンターは多くの点で SQL データベースのシーケンスジェネレーター、IDENTITY 列、または自動インクリメントフィールドと似ています。

UpdateItem オペレーションを使用して、新しい属性 (Plays) を初期化し、曲の再生回数を追跡する例を次に示します。

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
}
```

ReturnValues パラメータは、更新された任意の属性の新しい値を返す、UPDATED_NEW に設定されています。この場合は、0 (ゼロ) を返します。

ユーザーがこの曲を再生するたびに、次の UpdateItem オペレーションを使用して Plays を 1 ずつ増分できます。

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = Plays + :incr",
    ExpressionAttributeValues: {
        ":incr": 1
    },
    ReturnValues: "UPDATED_NEW"
}
```

> **Note**
>
> UpdateItem を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

PartiQL for DynamoDB

PartiQL では、Update ステートメントを利用する ExecuteStatement オペレーションを使用して、テーブル内の項目を変更します。

このテーブルのプライマリキーは、Artist および SongTitle で構成されます。これらの属性の値を指定する必要があります。

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

また、次の例のように、複数のフィールドを一度に変更することもできます。

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

また、Update はアトミックカウンター、または増加減少する Number 型の属性をサポートしています。アトミックカウンターは多くの点で SQL データベースのシーケンスジェネレーター、IDENTITY 列、または自動インクリメントフィールドと似ています。

次は、曲が再生された回数を追跡するための新しい属性 (Plays) を初期化する Update ステートメントの例です。

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

ユーザーがこの曲を再生するたびに、次の Update ステートメントを使用して Plays を 1 ずつ増分できます。

```
UPDATE Music
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> ⓘ Note
>
> Update と ExecuteStatement を使用したコード例については、DynamoDB 用の PartiQL 更新ステートメント を参照してください。

テーブルからデータを削除する

SQL では、DELETE ステートメントを使用してテーブルから 1 つ以上の行を削除します。Amazon DynamoDB では、この DeleteItem オペレーションを使用して一度に 1 つの項目を削除します。

トピック

- SQL を使ってテーブルからデータを削除する
- DynamoDB でテーブルからデータを削除する

SQL を使ってテーブルからデータを削除する

SQL では、DELETE ステートメントを使用して、1 つ以上の行を削除します。WHERE 句は、変更する行を決定します。以下はその例です。

```
DELETE FROM Music
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

WHERE 句を変更して、複数行を削除できます。たとえば、次の例に示すように、特定のアーティストのすべての曲を削除できます。

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

ⓘ Note

WHERE 句を省略すると、データベースはテーブルからすべての行を削除しようとします。

DynamoDB でテーブルからデータを削除する

DynamoDB では、クラシック API または PartiQL (SQL 互換のクエリ言語) を使用して、単一の項目を削除できます。複数の項目を変更する場合は、複数のオペレーションを使用する必要があります。

DynamoDB API

DynamoDB API では、DeleteItem オペレーションを使用して、テーブルからデータを一度に 1 項目ずつ削除します。項目のプライマリキー値を指定する必要があります。

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> Note
>
> DeleteItem に加えて、Amazon DynamoDB は、同時に複数の項目を削除するための BatchWriteItem オペレーションをサポートしています。

DeleteItem は、条件付き書き込みをサポートしており、特定の ConditionExpression が true と評価された場合のみ、オペレーションが成功します。たとえば、次の DeleteItem オペレーションは RecordLabel 属性がある場合のみ項目を削除します。

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

> Note
>
> DeleteItem を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

PartiQL for DynamoDB

PartiQL では、ExecuteStatement オペレーションでの Deleteステートメントを使用して、テーブルからデータを一度に 1 項目ずつ削除します。項目のプライマリキー値を指定する必要があります。

このテーブルのプライマリキーは、Artist および SongTitle で構成されます。これらの属性の値を指定する必要があります。

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

オペレーションには、追加の条件を指定することもできます。次の DELETE オペレーションは、11 を超える Awards が項目に含まれる場合にのみ、その項目を削除します。

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> DELETE と ExecuteStatement を使用したコード例については、DynamoDB 用の PartiQL 削除ステートメント を参照してください。

テーブルの削除

SQL では、DROP TABLE ステートメントを使用して、テーブルを削除します。Amazon DynamoDB では、DeleteTable オペレーションを使用します。

トピック

- SQL を使用してテーブルを削除する
- DynamoDB でテーブルを削除する

SQL を使用してテーブルを削除する

不要になったテーブルを完全に廃棄する場合、SQL の DROP TABLE ステートメントを使用します。

```
DROP TABLE Music;
```

テーブルは削除された後、復元できません。(一部のリレーショナルデータベースは DROP TABLE オペレーションを取消すことができますが、これはベンダー固有の機能であり、一般的には実装されていません。)

DynamoDB でテーブルを削除する

DynamoDB には、DeleteTable が同様のオペレーションです。次の例では、テーブルは完全に削除されます。

```
{
    TableName: "Music"
}
```

Note

DeleteTable を使用したサンプルコードについては、「DynamoDB および AWS SDK の使用開始」を参照してください。

Amazon DynamoDB のその他のリソース

以下に示すその他のリソースを使用して、DynamoDB について理解し、作業を行うことができます。

トピック

- コーディングと可視化のツール
- 「規範的ガイダンス」の記事
- ナレッジセンターの記事
- ブログ投稿、リポジトリ、ガイド
- データモデリングと設計パターンのプレゼンテーション
- トレーニングコース

コーディングと可視化のツール

次のコーディングおよび視覚化ツールを使用して DynamoDB を操作できます。

- Amazon DynamoDB 用の NoSQL Workbench – DynamoDB テーブルの設計、作成、クエリ、管理に役立つ、統合されたビジュアルツールです。このツールは、データモデリング、データの視覚化、クエリ開発機能を備えています。
- Dynobase – DynamoDB テーブルの表示と操作、アプリケーションコードの作成、リアルタイム検証を伴うレコードの編集を簡単に行うデスクトップツール。
- DynamoDB Toolbox – データモデリング、JavaScript および Node.js での作業に役立つユーティリティを提供する Jeremy Daly 氏のプロジェクト。
- DynamoDB Streams Processor – DynamoDB Streams の操作を簡単にするために使用できるシンプルなツール。

「規範的ガイダンス」の記事

「AWS の規範的ガイダンス」には、プロジェクトを加速させるのに役立つ、実績のある戦略、ガイド、パターンが記載されています。これらのリソースは、お客様のビジネス目標の達成を支援してきた長年の経験に基づいて、AWS テクノロジーのエキスパートと AWS パートナーのグローバルコミュニティによって開発されました。

データモデリングと移行

- DynamoDB の階層型データモデル
- DynamoDB によるデータのモデリング
- AWS DMS を使用して Oracle データベースを DynamoDB に移行する

グローバルテーブル

- Amazon DynamoDB グローバルテーブルを使用する

サーバーレス

- AWS Step Functions を使用してサーバーレス Saga パターンを実装する

SaaS アーキテクチャ

- 複数の SaaS 製品間のテナントを単一のコントロールプレーンで管理する
- C# と AWS CDK を使用するサイロモデル用の SaaS アーキテクチャでのテナントオンボーディング

データ保護とデータ移動

- Amazon DynamoDB へのクロスアカウントアクセスを設定する
- DynamoDB のフルテーブルコピーオプション
- AWS でのデータベースのディザスタリカバリ戦略

Miscellaneous

- DynamoDB でタグ付けを適用するのに役立ちます

規範ガイダンスの動画チュートリアル

- サーバーレスアーキテクチャを使用したデータパイプラインの作成
- Novartis - 購買エンジン: AI を活用した調達ポータル
- Veritiv: インサイトを活用して AWS データレイクで売上需要を予測
- mimik: AWS を活用してエッジマイクロサービスメッシュをサポートするハイブリッドエッジクラウド
- Amazon DynamoDB を使用した変更データキャプチャ

DynamoDB に関するその他の規範的ガイダンスの記事と動画については、「規範的ガイダンス」を参照してください。

ナレッジセンターの記事

AWS ナレッジセンターの記事と動画には、AWS のお客様から寄せられるよくある質問やリクエストが網羅されています。DynamoDB に関連した特定のタスクに関する最新のナレッジセンターの記事を、いくつか以下に示します。

コスト最適化

- Amazon DynamoDB を使用してコストを最適化する方法を教えてください。

スロットリングとレイテンシー

- 平均レイテンシーが正常なのに、DynamoDB 最大レイテンシーメトリクスが高いのはなぜですか?
- DynamoDB テーブルがスロットリングされているのはなぜですか?
- オンデマンド DynamoDB テーブルがスロットリングされているのはなぜですか?

ページ分割

- DynamoDB にページネーションを実装するにはどうすればよいですか?

トランザクション

- DynamoDB で `TransactWriteItems` API コールが失敗するのはなぜですか?

トラブルシューティング

- DynamoDB 自動スケーリングの問題を解決する方法を教えてください。
- DynamoDB の HTTP 4XX エラーをトラブルシューティングする方法を教えてください。

DynamoDB に関するその他の記事や動画については、ナレッジセンターの記事を参照してください。

ブログ投稿、リポジトリ、ガイド

「DynamoDB デベロッパーガイド」の他に、DynamoDB を使用するうえで役立つリソースが多数あります。DynamoDB を使用するためのブログ投稿、リポジトリ、ガイドをいくつか紹介します。

- さまざまな AWS SDK 言語での DynamoDB コード例のAWSリポジトリ: Node.js、Java、Python、.Net、Go、Rust。
- DynamoDB ブック – Alex DeBrie 氏からの包括的なガイドで、DynamoDB を使用したデータモデリングに対する戦略主導型のアプローチについて説明します。
- DynamoDB ガイド – Alex DeBrie 氏からのオープンガイドで、DynamoDB NoSQL データベースの基本的な概念と高度な機能について説明します。
- How to switch from RDBMS to DynamoDB in 20 easy steps – Jeremy Daly 氏からデータモデリングを学ぶために役立つステップのリスト。
- DynamoDB JavaScript DocumentClient cheat sheet – Node.js または JavaScript 環境で DynamoDB を使用してアプリケーションの構築を開始するのに役立つデータシート。
- DynamoDB Core Concept Videos – このプレイリストでは、DynamoDB の多くの重要な概念が説明されています。

データモデリングと設計パターンのプレゼンテーション

DynamoDB を最大限に活用するには、データモデリングと設計パターンに関する次のリソースを使用できます。

- AWSre:Invent 2019: DynamoDB を使用したデータモデリング
  - DynamoDB データモデリングの原則の使用開始に役立つ、Alex DeBrie 氏の講演
- AWSre:Invent 2020: DynamoDB を使用したデータモデリング – パート 1
- AWSre:Invent 2020: DynamoDB を使用したデータモデリング – パート 2

- AWS re:Invent 2017: Advanced design patterns
- AWS re:Invent 2018: Advanced design patterns
- AWS re:Invent 2019: Advanced design patterns
  - このセッションでは、Jeremy Daly が 12 の重要ポイントを説明しています。
- AWS re:Invent 2020: DynamoDB advanced design patterns – Part 1
- AWS re:Invent 2020: DynamoDB advanced design patterns – Part 2
- Twitch での DynamoDB の営業時間

> Note
>
> 各セッションでは、さまざまなユースケースと例について説明しています。

トレーニングコース

DynamoDB についてさらに学ぶためのさまざまなトレーニングコースや教育オプションがあります。現在の例をいくつかご紹介します。

- Developing with Amazon DynamoDB – AWS による設計で、Amazon DynamoDB のデータモデリングを使用して実際のアプリケーションを開発する初心者をエキスパートに育成します。
- DynamoDB Deep-Dive Course – Cloud Guru のコース。
- Amazon DynamoDB: Building NoSQL database-driven applications – edX でホストされる AWS トレーニングと認定チームのコース。

Amazon DynamoDB の読み込みと書き込み

DynamoDB の読み込みと書き込みは、テーブルからのデータの取得 (読み込み) と、テーブル内のデータの挿入、更新、削除 (書き込み) に関するオペレーションを指します。DynamoDB を使用する場合は、読み込みと書き込みの概念を理解することが不可欠です。読み込みと書き込みは、アプリケーションのパフォーマンスとコストに直接影響します。

このトピックでは、DynamoDB に適用されるさまざまなタイプの読み込み整合性について詳しく説明します。また、実行するさまざまな読み込みおよび書き込みオペレーションのユニット消費量についても説明します。

トピック

読み込み整合性

Amazon DynamoDB は、テーブル、ローカルセカンダリインデックス (LSI)、グローバルセカンダリインデックス (GSI)、およびストリームからデータを読み取ります。詳細については、「Amazon DynamoDB のコアコンポーネント」を参照してください。テーブルと LSI のどちらにも、結果整合性のある読み込み (デフォルト) と強力な整合性のある読み込みという 2 つの読み込み整合性オプションが用意されています。GSI とストリームからのすべての読み取りは、結果整合性のある読み込みです。

アプリケーションが DynamoDB テーブルにデータを書き込み、HTTP 200 応答 (OK) を受信した場合、これは書き込みが正常に完了し、永続的に保持されたことを意味します。DynamoDB は、コミット済み読み取り分離を提供し、読み取りオペレーションが常に項目のコミット済みの値を返すようにします。読み取りでは、最終的に成功しなかった書き込みの項目に対してビューが表示されることはありません。コミット済み読み取り分離では、読み取りオペレーションの直後に項目の変更が防止されません。

結果整合性のある読み込み

結果整合性のある読み込みは、すべての読み取りオペレーションのデフォルトの読み込み整合性モデルです。DynamoDB テーブルまたはインデックスに対して結果整合性のある読み込みを発行すると、最近完了した書き込みオペレーションの結果が応答に反映されない場合があります。少し時間が経ってから読み取りリクエストを繰り返すと、最終的に、より最新の項目が応答で返されるはずで

す。結果整合性のある読み込みは、テーブル、ローカルセカンダリインデックス、およびグローバルセカンダリインデックスでサポートされています。また、DynamoDB ストリームからのすべての読み取りも、結果整合性のある読み込みであることに注意してください。

結果整合性のある読み込みは、強力な整合性のある読み込みの半分のコストで済みます。詳細については、「Amazon DynamoDB 料金」を参照してください。

強力な整合性のある読み込み

読み取りオペレーション (`GetItem`、`Query`、`Scan` など) には、オプションの `ConsistentRead` パラメータがあります。`ConsistentRead` を true に設定すると、DynamoDB は、成功した以前のすべての書き込みオペレーションからの更新を反映した、最新データを含む応答を返します。強力な整合性のある読み込みは、テーブルとセカンダリインデックスでのみサポートされています。グローバルセカンダリインデックスまたは DynamoDB ストリームからの強力な整合性のある読み込みはサポートされていません。

グローバルテーブルの読み込み整合性

DynamoDB は、マルチアクティブおよびマルチリージョンのレプリケーション用のグローバルテーブルもサポートしています。グローバルテーブルは、異なる AWS リージョン間の複数のレプリカテーブルで構成されています。レプリカテーブル内の任意の項目に加えられた変更は、通常は 1 秒以内に同じグローバルテーブル内の他のすべてのレプリカにレプリケートされ、結果的に整合します。詳細については、「整合性と競合の解決」を参照してください。

読み込みと書き込みのオペレーション

DynamoDB の読み込みオペレーションでは、パーティションキー値とオプションのソートキー値を指定して、テーブルから 1 つ以上の項目を取得できます。DynamoDB の書き込みオペレーションでは、テーブル内の項目を挿入、更新、または削除できます。このトピックでは、これら 2 つのオペレーションのキャパシティユニット消費量について説明します。

トピック

読み込みオペレーションのキャパシティユニット消費量

DynamoDB 読み込みリクエストは、強力な整合性、結果整合性、またはトランザクションのいずれかになります。

- 4 KB 以下の項目の強力な整合性のある読み込みリクエストには、1 つの読み込みユニットが必要です。
- 4 KB 以下の項目の結果整合性のある読み込みリクエストには、2 分の 1 の読み込みユニットが必要です。
- 4 KB 以下の項目のトランザクション読み込みリクエストには、2 つの読み込みユニットが必要です。

DynamoDB 読み込み整合性モデルの詳細については「読み込み整合性」を参照してください。

読み込みの項目サイズは、次の 4 KB の倍数に切り上げられます。たとえば、3,500 バイトの項目の読み込みは、4 KB の項目の読み込みと同じスループットを消費します。

4 KB より大きい項目を読み込む必要がある場合、DynamoDB には追加の読み込みユニットが必要です。必要な読み込みユニットの合計数は、項目のサイズと、結果整合性のある読み込みまたは強力な整合性のある読み込みが必要かどうかによって異なります。例えば、項目のサイズが 8 KB の場合、1 つの強力な整合性のある読み込みを維持するには 2 つの読み込みユニットが必要です。結果整合性のある読み込みを選択した場合は 1 つの読み込みユニットが必要です。トランザクション読み込みリクエストを選択した場合は 4 つの読み込みユニットが必要です。

DynamoDB の読み込みオペレーションが読み込みユニットをどのように消費するかを以下に示します。

- GetItem: テーブルから単一の項目を読み込みます。`GetItem` が消費する読み込みユニットの数を決定するには、項目のサイズを次の 4 KB の倍数まで切り上げます。強力な整合性のある読み込みを指定した場合は、この数値が必要な読み込みユニットの数になります。結果整合性のある読み込み (デフォルト) の場合は、この数値を 2 で割ります。

  たとえば、3.5 KB の項目を読み取ると、DynamoDB は項目サイズを 4 KB まで切り上げます。10 KB の項目を読み取ると、DynamoDB は項目サイズを 12 KB まで切り上げます。
- BatchGetItem: 1 つ以上のテーブルから最大 100 個の項目を読み込みます。DynamoDB は、バッチ内の各項目を個別の `GetItem` リクエストとして処理します。DynamoDB は、まず各項目のサイズを次の 4 KB の倍数に切り上げてから合計サイズを計算します。この結果は、すべての項目の合計サイズと必ずしも一致しません。例えば、`BatchGetItem` がサイズ 1.5 KB と 6.5 KB の 2 つの項目を読み込むと、DynamoDB はサイズを 12 KB (4 KB + 8 KB) として計算します。DynamoDB はサイズを 8 KB (1.5 KB + 6.5 KB) として計算しません。
- Query: 同じパーティションキー値を持つ複数の項目を読み込みます。返されるすべての項目は単一の読み込みオペレーションとして扱われ、DynamoDB はすべての項目の合計サイズを計算しま

す。次に、DynamoDB はサイズを次の 4 KB の倍数に切り上げます。たとえば、クエリの結果、合計サイズが 40.8 KB になる 10 項目が返されるとします。DynamoDB は、オペレーションの項目サイズを 44 KB に切り上げます。クエリの結果、64 バイトの項目が 1,500 項目返されると、累積サイズは 96 KB になります。

- Scan: テーブル内のすべての項目を読み込みます。DynamoDB は、スキャンにより返される項目のサイズではなく、評価される項目のサイズを考慮します。Scan オペレーションの詳細については、「DynamoDB でのスキャンの使用」を参照してください。

> Important
>
> 存在しない項目に対して読み込みオペレーションを実行した場合でも、DynamoDB は、上で説明したとおりに読み込みスループットを消費します。`Query`/`Scan` オペレーションでは、データが存在しない場合でも、読み込み整合性のタイプとリクエストを処理するために検索したパーティションの数に基づいて、追加の読み込みスループットの料金が発生します。

項目を返す任意のオペレーションで、属性のサブセットを取得するようリクエストできます。ただし、そうすることで項目サイズの計算に影響は生じません。また、`Query` と `Scan` は、属性値の代わりに項目数を返します。項目数の取得では、項目ごとに読み込みユニット数が同じで、項目サイズの計算方法も同じとします。これは、DynamoDB では、項目数を増加させるために項目単位で読み込む必要があるためです。

書き込みオペレーションのキャパシティユニットの消費量

1 つの書き込みユニットは、最大サイズが 1 KB の項目について 1 回の書き込みを表します。1 KB より大きい項目を書き込む必要がある場合、DynamoDB は追加の書き込みユニットを消費する必要があります。トランザクション書き込みリクエストでは、最大 1 KB の項目を 1 回書き込むのに書き込みユニットが 2 個必要です。必要な書き込みリクエストユニットの合計数は、項目サイズに応じて異なります。例えば、項目のサイズが 2 KB の場合、1 回の書き込みリクエストを維持するには書き込みユニットが 2 個必要です。トランザクション書き込みリクエストには書き込みユニットが 4 個必要です。

書き込みの項目サイズは、次の 1 KB の倍数に切り上げられます。たとえば、500 バイトの項目の書き込みは、1 KB の項目の書き込みと同じスループットを消費します。

DynamoDB の書き込みオペレーションが書き込みユニットをどのように消費するかを以下に示します。

- PutItem: テーブルに 1 つの項目を書き込みます。同じプライマリキーを持つ項目がテーブル内に存在する場合、このオペレーションによって項目が置き換えられます。プロビジョニングされたスループットの消費量を算出する場合、対象となる項目サイズは 2 つのうち大きい方となります。
- UpdateItem: テーブル内の 1 つの項目を変更します。DynamoDB は更新の前後で実際の項目のサイズを考慮します。プロビジョニングされたスループットの消費は、これらの項目サイズの大きい方を反映しています。項目の属性の一部を更新する場合でも、UpdateItem は、プロビジョニングされたスループットの総量 (「前」の項目サイズと「後」の項目サイズで、より大きい方) を消費します。
- DeleteItem: テーブルから 1 つの項目を削除します。プロビジョニングされたスループットの消費量は、削除された項目のサイズに基づいています。
- BatchWriteItem: 1 つ以上のテーブルに最大 25 個の項目を書き込みます。DynamoDB は、バッチ内の各項目を個別の PutItem または DeleteItem リクエストとして処理します (更新はサポートされません)。DynamoDB は、まず各項目のサイズを次の 1 KB の倍数に切り上げてから合計サイズを計算します。この結果は、すべての項目の合計サイズと必ずしも一致しません。例えば、BatchWriteItem がサイズ 500 バイトと 3.5 KB の 2 つの項目を書き込むと、DynamoDB はサイズを 5 KB (1 KB + 4 KB) として計算します。DynamoDB はサイズを 4 KB (500 バイト + 3.5 KB) として計算しません。

PutItem、UpdateItem、および DeleteItem の各オペレーションでは、DynamoDB は項目のサイズを次の 1 KB に切り上げます。たとえば、1.6 KB の項目を入力または削除すると、DynamoDB は項目サイズを 2 KB まで切り上げます。

PutItem、UpdateItem、DeleteItem の各オペレーションでは、条件付き書き込みが許可されます。この場合、オペレーションが成功するためには true と必ず評価される式を指定します。式が false と評価された場合でも、DynamoDB はテーブルの書き込みキャパシティユニットを消費します。消費される書き込みキャパシティユニットの数は、項目のサイズによって異なります この項目は、テーブル内の既存の項目になるか、作成または更新しようとしている新しい項目になる場合があります。例えば、既存の項目が 300 KB であるとします。作成または更新しようとしている新しい項目が 310 KB であるとします。この場合、消費される書き込みキャパシティユニットは、新しい項目の 310 KB に基づきます。

DynamoDB のスループットキャパシティ

テーブルのスループットキャパシティモードによって、テーブルのキャパシティの管理方法が決まります。また、スループットキャパシティによって、テーブルの読み込みおよび書き込みオペレーションに対する課金方法も決まります。Amazon DynamoDB では、さまざまなワークロード要件に対応するために、テーブルのオンデマンドモードとプロビジョンドモードを使い分けることができます。

トピック

DynamoDB のキャパシティモードの概要

このセクションでは、DynamoDB テーブルで使用できる 2 つのキャパシティモードの概要と、アプリケーションに適したキャパシティモードを選択する際の考慮事項について説明します。2 つのモードにより、応答性の要件と使用状況の管理方法に応じて、さまざまなニーズを満たすことができます。

オンデマンドモード

Amazon DynamoDB オンデマンドは、キャパシティプランニングなしで 1 秒あたり数百万ものリクエストを処理できるサーバーレス請求オプションです。DynamoDB オンデマンドは、読み込みおよび書き込みリクエストの料金が従量制であるため、使用した分だけを支払います。オンデマンドモードのテーブルでは、アプリケーションで実行することが予測される読み込みおよび書き込みスループットを指定する必要はありません。

オンデマンドモードでは、DynamoDB がスループットを全面的に管理します。ユーザーはテーブルのスループットキャパシティを管理することなく、必要に応じて API コールを実行できます。

次のいずれかに当てはまる場合は、オンデマンドキャパシティモードが最適です。

- Amazon DynamoDB の使用を単に開始する場合
- トラフィックパターンが不明な本番稼働用の新しいアプリケーションで開発、テスト、プロトタイピング、実行を行う場合

- アプリケーションのトラフィックがバースト、断続的、または不透明で、予測が難しい場合
- わかりやすい従量課金制の支払いを希望します。

詳細については、「オンデマンドキャパシティモード」を参照してください。

プロビジョニングモード

プロビジョンドモードでは、アプリケーションに必要な 1 秒あたりの読み込みと書き込みの回数を指定します。プロビジョンドキャパシティを十分に活用していない場合でも、スループットキャパシティに対して課金されます。プロビジョニングした時間ごとの読み込みおよび書き込みキャパシティに基づいて課金されます。自動スケーリングを使用すると、トラフィックの変更に応じて、テーブルのプロビジョンドキャパシティを自動的に調整できます。これにより、コストの予測可能性を得るため、定義されたリクエストレート以下に維持されるように DynamoDB を制御することができます。

次のいずれかに当てはまる場合は、プロビジョンドキャパシティモードが最適です。

- アプリケーションのトラフィックが予測可能または周期的である場合
- トラフィックが一定であるか、徐々に増加するアプリケーションを実行する場合
- キャパシティーの要件を予測してコストを管理できる。
- トラフィックの急増が限定的かつ短期である場合

詳細については、「プロビジョンドキャパシティモード」を参照してください。

次の動画では、テーブルのスループットキャパシティの概要を示します。この動画では、要件に基づいてキャパシティモードを選択する方法についても説明します。

オンデマンドキャパシティモード

Amazon DynamoDB オンデマンドは、キャパシティプランニングなしで 1 秒あたり数百万ものリクエストを処理できるサーバーレス請求オプションです。DynamoDB オンデマンドは、読み込みおよび書き込みリクエストの料金が従量制であるため、使用した分だけを支払います。

オンデマンドモードを選択すると、DynamoDB は、前に到達したトラフィックレベルまで拡張または縮小して、ワークロードを即座に受け入れることができるようにします。ワークロードのトラフィックレベルが新しいピークに達すると、DynamoDB はワークロードに対応するように迅速に対応します。オンデマンドモードのスケーリングプロパティの詳細については、「初期スループットとスケーリングのプロパティ」を参照してください。

オンデマンドモードを使用するテーブルは、同じ 1 桁ミリ秒のレイテンシー、サービスレベルアグリーメント (SLA) のコミットメント、DynamoDB が既に実現しているセキュリティを提供します。オンデマンドは、新しいテーブルと既存のテーブルの両方に選択できるだけでなく、コードを変更せずに既存の DynamoDB API を引き続き使用することができます。

オンデマンドのスループットレートは、アカウントのすべてのテーブルに適用されるテーブルレベルのスループットクォータによって制限されます。このクォータの引き上げをリクエストできます。詳細については、「スループットのデフォルトクォータ」を参照してください。

オプションで、個別のオンデマンドテーブルやグローバルセカンダリインデックスに対して 1 秒あたりの読み込み/書き込み (またはその両方) の最大スループットを設定することもできます。スループットを設定することで、テーブルレベルの使用量とコストを制限し、不注意によるリソース消費量の急増を防ぎ、過剰な使用をなくして予測可能なコスト管理を行うことができます。テーブルの最大スループットを超えるスループットリクエストはスロットリングされます。テーブル別の最大スループットは、アプリケーションの要件に基づいていつでも変更できます。詳細については、「オンデマンドテーブルの最大スループット」を参照してください。

開始するには、オンデマンドモードを使用するテーブルを作成または更新します。詳細については、「DynamoDB テーブルの基本的なオペレーション」を参照してください。

テーブルは、オンデマンドモードからプロビジョンドキャパシティモードにいつでも切り替えることができます。キャパシティモード間で複数の切り替えを行う場合は、次の条件が適用されます。

- オンデマンドモードで新しく作成したテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、テーブルの作成タイムスタンプから 24 時間後のみです。
- オンデマンドモードの既存のテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、オンデマンドへの切り替えを示す最後のタイムスタンプから 24 時間後のみです。

読み込みおよび書き込みキャパシティモード間の切り替えの詳細については、「キャパシティモードを切り替える際の考慮事項」を参照してください。

トピック

読み取りリクエスト単位と書き込みリクエスト単位

DynamoDB では、アプリケーションがテーブルに対して実行する読み込みと書き込みに対して、読み込みリクエストユニットと書き込みリクエストユニットに基づいて課金します。

1 つの読み込みリクエストユニットは、最大サイズが 4 KB の項目に対する 1 秒あたり 1 回の強力な整合性のある読み込み、または 1 秒あたり 2 回の結果整合性のある読み込みを表します。DynamoDB の読み込み整合性モデルの詳細については「読み込み整合性」を参照してください。

1 つの書き込みリクエストユニットは、最大サイズが 1 KB の項目に対する 1 秒あたり 1 回の書き込みオペレーションを表します。

読み込みユニットと書き込みユニットの消費方法の詳細については、「読み込みと書き込みのオペレーション」を参照してください。

初期スループットとスケーリングのプロパティ

オンデマンドキャパシティーモードを使用する DynamoDB テーブルは、アプリケーションのトラフィックボリュームに自動的に対応します。新しいオンデマンドテーブルでは、1 秒あたり最大 4,000 回の書き込みと 1 秒あたり最大 12,000 回の読み込みを維持できます。オンデマンドキャパシティーモードは、テーブルにおける前のピークトラフィックの最大 2 倍まで瞬時に対応します。例えば、アプリケーションのトラフィックパターンが 1 秒あたり 25,000～50,000 回の強力な整合性のある読み込みの範囲で変動し、1 秒あたり 50,000 回の読み込みが前回のトラフィックのピークであったとします。オンデマンドキャパシティモードは、1 秒あたり最大 100,000 回の読み込みの持続的なトラフィックに即座に対応します。アプリケーションが 1 秒あたり 100,000 回の読み込みのトラフィックを維持する場合、このピークが新たに前回のピークとなります。この前回のピークにより、後続のトラフィックは 1 秒あたり最大 200,000 回の読み込みに到達できます。

1 つのテーブルで前回のピークの 2 倍を超えるピークがワークロードで発生した場合、DynamoDB は、トラフィックボリュームの増加に合わせて、キャパシティの割り当てを自動的に増加します。このキャパシティの割り当てにより、ワークロードでスロットリングが発生しなくなります。ただし、30 分以内に前のピークの 2 倍を超えた場合、スロットリングが発生する可能性があります。例えば、アプリケーションのトラフィックパターンが 1 秒あたり 25,000～50,000 回の強力な整合性のある読み込みの範囲で変動し、1 秒あたり 50,000 回の読み込みが前回到達したトラフィックのピークであったとします。1 秒あたり 100,000 回を超える読み込みを駆動する前に、テーブルを事前ウォーミングするか、少なくとも 30 分間にわたってトラフィックの増加の間隔を空けることをお勧めします。事前ウォーミングの詳細については、「オンデマンドキャパシティモードのテーブルの事前ウォーミング」を参照してください。

オンデマンドテーブルの最大スループット

オンデマンドテーブルの場合、オプションで、個々のテーブルおよび関連するグローバルセカンダリインデックス (GSI) の 1 秒あたりの読み込みまたは書き込み (または両方) の最大スループットを指定できます。オンデマンドの最大スループットを指定すると、テーブルレベルの使用量とコストを制限できます。デフォルトでは、最大スループットの設定は適用されません。オンデマンドのスループットレートは、すべてのテーブルまたはテーブル内のすべての GSI に対する AWS のサービスクォータによって制限されます。必要に応じて、サービスクォータの引き上げをリクエストできます。

オンデマンドテーブルの最大スループットを設定すると、指定した最大量を超えるスループットリクエストはスロットリングされます。テーブルレベルのスループット設定は、アプリケーションの要件に応じていつでも変更できます。

オンデマンドテーブルの最大スループットを使用することでメリットを得られる一般的なユースケースの例は以下のとおりです。

- スループットのコスト最適化 — オンデマンドテーブルの最大スループットを使用すると、コスト予測可能性と管理可能性を高めることができます。さらに、トラフィックパターンと予算が異なるワークロードをサポートするために、オンデマンドモードを使用する柔軟性も高まります。
- 過剰な使用からの保護 – 最大スループットを設定することで、オンデマンドテーブルに対する最適化されていないコードや不正なプロセスに伴う読み込みや書き込みの消費量の偶発的な急増を防止できます。このテーブルレベルの設定により、組織は特定の期間にわたる過剰なリソースの消費を防止できます。
- ダウンストリームサービスの保護 — 顧客アプリケーションには、サーバーレステクノロジーと非サーバーレステクノロジーを含めることができます。サーバーレスアーキテクチャは、需要に合わせて迅速にスケールできます。ただし、固定キャパシティを持つダウンストリームコンポーネントでは、対応できなくなる場合があります。オンデマンドテーブルに最大スループット設定を実装すると、複数のダウンストリームコンポーネントに対する大量のイベントの伝播に伴う予期しない副作用を防止できます。

オンデマンドモードの最大スループットは、新規および既存の単一リージョンのテーブル、グローバルテーブル、GSI に対して設定できます。最大スループットは、Amazon S3 ワークフローからのテーブルの復元時とデータのインポート時にも設定できます。

オンデマンドテーブルの最大スループット設定を指定するには、DynamoDB コンソール、AWS CLI、AWS CloudFormation、または DynamoDB API を使用できます。

> **Note**
>
> オンデマンドテーブルの最大スループットは、ベストエフォートベースで適用されるため、保証されたリクエストの上限としてではなく、目標として考える必要があります。バーストキャパシティにより、ワークロードが、指定した最大スループットを一時的に超える場合があります。場合によっては、DynamoDB がバーストキャパシティを使用して、テーブルの最大スループット設定を超える読み込みや書き込みに対応します。バーストキャパシティーにより、スロットリングされていた可能性のある読み込みまたは書き込みリクエストが成功します。

トピック

- オンデマンドモードの最大スループットを使用する際の考慮事項
- リクエストのスロットリングと CloudWatch メトリクス

オンデマンドモードの最大スループットを使用する際の考慮事項

オンデマンドモードでテーブルの最大スループットを使用する場合、次の考慮事項が適用されます。

- オンデマンドテーブルまたはテーブル内のグローバルセカンダリインデックスごとに読み込みと書き込みの最大スループットを個別に設定し、特定の要件に基づいてアプローチを微調整できます。
- Amazon CloudWatch を使用して、DynamoDB テーブルレベルの使用状況メトリクスをモニタリングおよび把握し、オンデマンドモードの適切な最大スループット設定を決定できます。詳細については、「DynamoDB のメトリクスとディメンション」を参照してください。
- 1 つのグローバルテーブルレプリカで読み込みや書き込み (または両方) の最大スループット設定を指定すると、同じ最大スループット設定がすべてのレプリカテーブルに自動的に適用されます。データが適切にレプリケートされるように、グローバルテーブル内のレプリカテーブルとセカンダリインデックスの書き込みスループット設定が同じであることが重要です。詳細については、「グローバルテーブルを管理するためのベストプラクティスと要件」を参照してください。
- 読み込みまたは書き込みの最大スループットとして指定できる最小値は、1 秒あたり 1 つのリクエストユニットです。
- 指定する最大スループットは、オンデマンドテーブルまたはテーブル内の個別のグローバルセカンダリインデックスで使用できるデフォルトのスループットクォータより小さくする必要があります。

リクエストのスロットリングと CloudWatch メトリクス

アプリケーションがオンデマンドテーブルに設定した読み込みまたは書き込みの最大スループットを超えると、DynamoDB はリクエストのスロットリングを開始します。DynamoDB は、読み込みや書き込みをスロットリングすると、発信者に `ThrottlingException` を返します。その後、必要に応じて適切なアクションを実行できます。例えば、テーブルの最大スループット設定を増加または無効化したり、リクエストを再試行する前に短い間待ったりできます。

テーブルまたはグローバルセカンダリインデックスに設定した最大スループットのモニタリングを簡素化するために、CloudWatch はメトリクスとして OnDemandMaxReadRequestUnits と OnDemandMaxWriteRequestUnits を提供しています。

オンデマンドキャパシティモードのテーブルの事前ウォーミング

オンデマンドテーブルの場合、DynamoDB は、トラフィック量の増加に応じてキャパシティの割り当て量を自動的に増やします。新しいオンデマンドテーブルでは、1 秒あたり最大 4,000 回の書き込みと 1 秒あたり最大 12,000 回の読み込みを維持できます。アクセスがパーティション間で均等に分散され、テーブルが前回のピークトラフィックの 2 倍を超えていない場合、テーブル全体はスロットリングされません。ただし、同じ 30 分以内にスループットが前回のピークの 2 倍を超えると、スロットリングが発生する可能性があります。

解決策の 1 つは、予想されるスパイクのピークキャパシティまでテーブルを事前ウォーミングすることです。アカウントの制限をチェックし、プロビジョニングモードで必要なキャパシティに到達できることを確認します。アカウントレベルとテーブルレベルの両方の制限の詳細については、「スループットのデフォルトクォータ」を参照してください。

> **Note**
>
> 既存のテーブルやオンデマンドモードの新しいテーブルを事前ウォーミングする場合は、予想されるピークの少なくとも 24 時間前に事前ウォーミングを開始します。24 時間内に実行できるスイッチの数には特定の条件があります。これらの条件の詳細については、キャパシティモードを切り替える際の考慮事項 を参照してください。

テーブルを事前ウォーミングするには、次の手順を実行します。

1. テーブルのキャパシティモードに基づいて、次のいずれかのステップを実行します。

   a. 現在オンデマンドモードになっているテーブルを事前ウォーミングするには、プロビジョンドモードに切り替えて 24 時間待ちます。

   b. プロビジョンドモードの新しいテーブル、またはプロビジョンドモードになって 24 時間経過した新しいテーブルを事前ウォーミングする場合は、待たずに次のステップに進みます。

2. テーブルの書き込みスループットを目的のピーク値に設定し、その値を数分間維持します。オンデマンドに切り替えるまでは、この大量のスループットによるコストが発生します。
3. オンデマンドキャパシティモードに切り替えます。これにより、テーブルはプロビジョニングされたスループットキャパシティ値と同様の数のリクエストを処理できるようになります。

プロビジョンドキャパシティモード

Amazon DynamoDB で新しいプロビジョンドテーブルを作成する場合は、テーブルのプロビジョンドスループットキャパシティを指定する必要があります。これは、テーブルがサポートできる読み込みおよび書き込みスループットの量です。DynamoDB は、この情報を使用して、スループット要件を満たすだけの十分なシステムリソースがあることを確認します。

オプションで、DynamoDB の自動スケーリングでテーブルのスループットキャパシティを管理できます。自動スケーリングを使用するには、テーブルの作成時に読み込みおよび書き込みキャパシティの初期設定を指定する必要があります。DynamoDB の自動スケーリングでは、これらの初期設定を開始点として使用し、アプリケーションの要件に応じて設定を動的に調整します。詳細については、「DynamoDB Auto Scaling によるスループットキャパシティの自動管理」を参照してください。

アプリケーションのデータとアクセス要件が変わると、テーブルのスループット設定の調整が必要になる場合があります。DynamoDB Auto Scaling を使用している場合、スループット設定は実際のワークロードに応じて自動的に調整されます。UpdateTable オペレーションを使用して、テーブルのスループットキャパシティを手動で調整することもできます。既存のデータストアから新しい DynamoDB テーブルにデータをバルクロードする必要がある場合などに便利です。大容量の書き込みスループットを設定したテーブルを作成し、データのバルクロードが完了してからこの設定を削減ですることができます。

テーブルは、オンデマンドモードからプロビジョンドキャパシティモードにいつでも切り替えることができます。キャパシティモード間で複数の切り替えを行う場合は、次の条件が適用されます。

- オンデマンドモードで新しく作成したテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、テーブルの作成タイムスタンプから 24 時間後のみです。
- オンデマンドモードの既存のテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、オンデマンドへの切り替えを示す最後のタイムスタンプから 24 時間後のみです。

読み込みおよび書き込みキャパシティモードの詳細については、「キャパシティモードを切り替える際の考慮事項」を参照してください。

トピック

- 読み取りキャパシティユニットと書き込みキャパシティユニット
- 初期スループット設定の選択
- DynamoDB 自動スケーリング
- DynamoDB Auto Scaling によるスループットキャパシティの自動管理
- リザーブドキャパシティ

読み取りキャパシティユニットと書き込みキャパシティユニット

プロビジョンドモードテーブルでは、スループット要件をキャパシティユニットの数で指定します。キャパシティユニットは、アプリケーションが 1 秒あたりに読み込みまたは書き込みを行う必要があるデータの量を表します。これらの設定は必要に応じて後から変更できます。また、DynamoDB Auto Scaling を有効化して自動的に変更することもできます。

最大サイズ 4 KB の項目の場合、1 つの読み込みキャパシティユニットは、1 秒あたり 1 回の強力な整合性のある読み込み、または 1 秒あたり 2 回の結果整合性のある読み込みを表します。DynamoDB の読み込み整合性モデルの詳細については「読み込み整合性」を参照してください。

1 つの書き込みキャパシティユニットは、最大サイズが 1 KB の項目の場合、1 秒あたり 1 回の書き込みを表します。さまざまな読み込みおよび書き込みオペレーションの詳細については、「読み込みと書き込みのオペレーション」を参照してください。

初期スループット設定の選択

アプリケーションごとにデータベースの読み込みおよび書き込みの要件は異なります。DynamoDB テーブルの初期スループット設定を決定する際は、以下の点を考慮してください。

- 予想される読み込みおよび書き込みリクエストレート — 1 秒あたりに実行する必要がある読み込みおよび書き込みの数を推定する必要があります。
- 項目のサイズ — 一部の項目は小さいため、読み込みや書き込みには 1 つのキャパシティユニットを使用できます。項目のサイズが大きくなると、複数のキャパシティーユニットが必要になります。テーブルに追加する項目の平均サイズを見積もることで、テーブルのプロビジョニングされたスループットに対する正確な設定を指定できます。

- 読み込みの整合性要件 — 読み込みキャパシティユニットは、強力な整合性のある読み込みオペレーションに基づいており、結果整合性のある読み込みの 2 倍のデータベースリソースを消費します。アプリケーションが強力な整合性のある読み込みを要求するか、またはこの要件を緩和して結果的に整合性のある読み込み行うかどうかを決定する必要があります DynamoDB での読み込みオペレーションは、デフォルトでは結果整合性のある読み込みです。必要に応じて、これらのオペレーションに強力な整合性のある読み込みをリクエストできます。

例えば、テーブルから 1 秒あたり 80 項目を読み込むとします。項目のサイズは 3 KB で、強力な整合性のある読み込みが必要です。このシナリオでは、読み込みごとに 1 つのプロビジョニングされた読み込みキャパシティユニットが必要です。この数を決定するには、オペレーションの項目サイズを 4 KB で割ります。次に、以下の例に示すように、最も近い整数に切り上げます。

- 3 KB / 4 KB = 0.75、または 1 つの読み込みキャパシティユニット

したがって、テーブルから 1 秒あたり 80 項目を読み込むには、次の例に示すように、テーブルのプロビジョニングされた読み込みスループットを 80 個の読み込みキャパシティユニットに設定します。

- 1 読み込み容量単位/項目 x 80 読み込み/秒 = 80 読み込み容量単位

次に、テーブルに 1 秒あたり 100 項目を書き込み、各項目のサイズが 512 バイトであるとします。この場合、書き込みごとに 1 つのプロビジョニングされた書き込みキャパシティユニットが必要です。この数を決定するには、オペレーションの項目サイズを 1 KB で割ります。次の例に示すように、結果を最も近い整数に切り上げます。

- 512 バイト/1 KB = 0.5 または 1 つの書き込みキャパシティユニット

テーブルに 1 秒あたり 100 項目を書き込むには、テーブルのプロビジョニングされた書き込みスループットを 100 個の書き込みキャパシティユニットに設定します。

- 1 書き込み容量単位/項目 x 100 書き込み/秒 = 100 書き込み容量単位

DynamoDB 自動スケーリング

DynamoDB 自動スケーリングでは、テーブルとグローバルセカンダリインデックスのプロビジョニングされたスループットキャパシティを自動的に管理します。読み込みおよび書き込みのキャパシ

ティーユニットの範囲 (上限と下限) と、また、その範囲内で目標使用率を定義します。DynamoDB Auto Scaling では、アプリケーションのワークロードが増減しても、ターゲットの使用率が維持されます。

DynamoDB Auto Scaling では、急激なトラフィック増加をリクエストのスロットリングなしに処理するために、テーブルまたはグローバルセカンダリインデックスのプロビジョンされた読み込み容量と書き込み容量を増やすことができます。ワークロードが減ると、DynamoDB Auto Scaling はスループットを低下させ、未使用のプロビジョンされた容量に料金が発生しないようにします。

> Note
>
> AWS Management Console を使用してテーブルまたはグローバルセカンダリインデックスを作成すると、デフォルトで DynamoDB オートスケーリングが有効になります。Auto Scaling の設定は、コンソール、AWS CLI、またはいずれかの AWS SDK を使用していつでも管理できます。詳細については、「DynamoDB Auto Scaling によるスループットキャパシティの自動管理」を参照してください。

使用率

使用率は、プロビジョニングキャパシティを超過しているかどうかを判断するのに役立ちます。超過している場合は、テーブルのキャパシティを減らしてコストを削減する必要があります。逆に、プロビジョニングキャパシティを下回っているかどうかを判断するのにも役立ちます。この場合、予期しないトラフィック増によりリクエストがスロットリングされる可能性を防ぐため、テーブルのキャパシティを増やす必要があります。詳細については、「Amazon DynamoDB auto scaling: Performance and cost optimization at any scale」を参照してください。

DynamoDB 自動スケーリングを使用している場合は、ターゲット使用率も設定する必要があります。自動スケーリングでは、この使用率をターゲットとして使用し、キャパシティを上下に調整します。ターゲット使用率は 70% に設定することをお勧めします。詳細については、「DynamoDB Auto Scaling によるスループットキャパシティの自動管理」を参照してください。

DynamoDB Auto Scaling によるスループットキャパシティの自動管理

多くのデータベースワークロードは本質的に循環的なものであり、前もって予測することが困難なものもあります。例えば、日中の時間帯に大部分のユーザーがアクティブなソーシャルネットワーキングアプリがあるとします。データベースは日中のアクティビティを処理できる必要がありますが、夜間のスループットに同じレベルは必要ありません。別の例としては、予想以上に急速に普及した新しいモバイルゲームアプリが挙げられます。ゲームがあまりに人気になると、利用可能なデータベース

リソースを超過し、パフォーマンスが低下して顧客が不満を感じるようになります。この種のワークロードでは多くの場合、手動介入によってデータベースリソースを使用レベルに応じて上下させる必要があります。

Amazon DynamoDB Auto Scaling は AWS Application Auto Scaling サービスを使用し、実際のトラフィックパターンに応じてプロビジョンドスループット性能をユーザーに代わって動的に調節します。これにより、テーブルまたはグローバルセカンダリインデックスで、プロビジョンされた読み込みおよび書き込み容量が拡張され、トラフィックの急激な増加をスロットリングなしに処理できるようになります。ワークロードが減ると、Application Auto Scaling はスループットを低下させ、未使用のプロビジョンされた容量に料金が発生しないようにします。

> Note
>
> AWS Management Console を使用してテーブルまたはグローバルセカンダリインデックスを作成すると、デフォルトで DynamoDB Auto Scaling が有効になります。Auto Scaling の設定はいつでも変更できます。詳細については、「AWS Management Console と DynamoDB Auto Scaling の使用」を参照してください。
> テーブルまたはグローバルテーブルレプリカを削除しても、関連するスケーラブルターゲット、スケーリングポリシー、または CloudWatch アラームが共に自動的に削除されることはありません。

Application Auto Scaling を使用して、テーブルまたはグローバルセカンダリインデックスのスケーリングポリシーを作成します。スケーリングポリシーは、テーブルまたはインデックスの読み込みキャパシティまたは書き込みキャパシティ (またはその両方) およびプロビジョニングされたキャパシティユニット設定をスケーリングするかどうかを指定します。

スケーリングポリシーには、ターゲット使用率 (ある時点で消費したプロビジョン済みスループットの割合) も含まれます。Application Auto Scaling はのターゲット追跡アルゴリズムを使用して、実際のワークロードに応じてテーブル (インデックス) のプロビジョンスループットを上下に調整することで、実際の容量使用率がターゲット使用率に、またはその近くに留まるようにします。

自動スケーリングは、2 つのデータポイントが、1 分以内に設定された目標使用率値を超えた場合にトリガーされます。したがって、消費された容量が目標使用率よりも高い状態が 2 分間継続すると、自動スケーリングが実行されます。ただし、スパイクの間隔が 1 分を超えると、自動スケーリングがトリガーされない場合があります。同様に、15 個の連続するデータポイントが目標使用率を下回ると、スケールダウンイベントがトリガーされます。いずれの場合も、自動スケーリングがトリガーされた後に UpdateTable コールが呼び出されます。この呼び出しは、テーブルまたはインデッ

クスのプロビジョンドキャパシティを更新するのに数分かかる場合があります。この期間中、テーブルの前のプロビジョンドキャパシティを超えるリクエストはスロットルされます。

⚠ Important

侵害するデータポイントの数を調整して、基礎となるアラームをトリガーすることはできません (ただし、現在の数は将来変更される可能性があります)。

読み取りおよび書き込み容量に対して、Auto Scaling ターゲット使用率の値を 20% から 90% の間で設定できます。

ⓘ Note

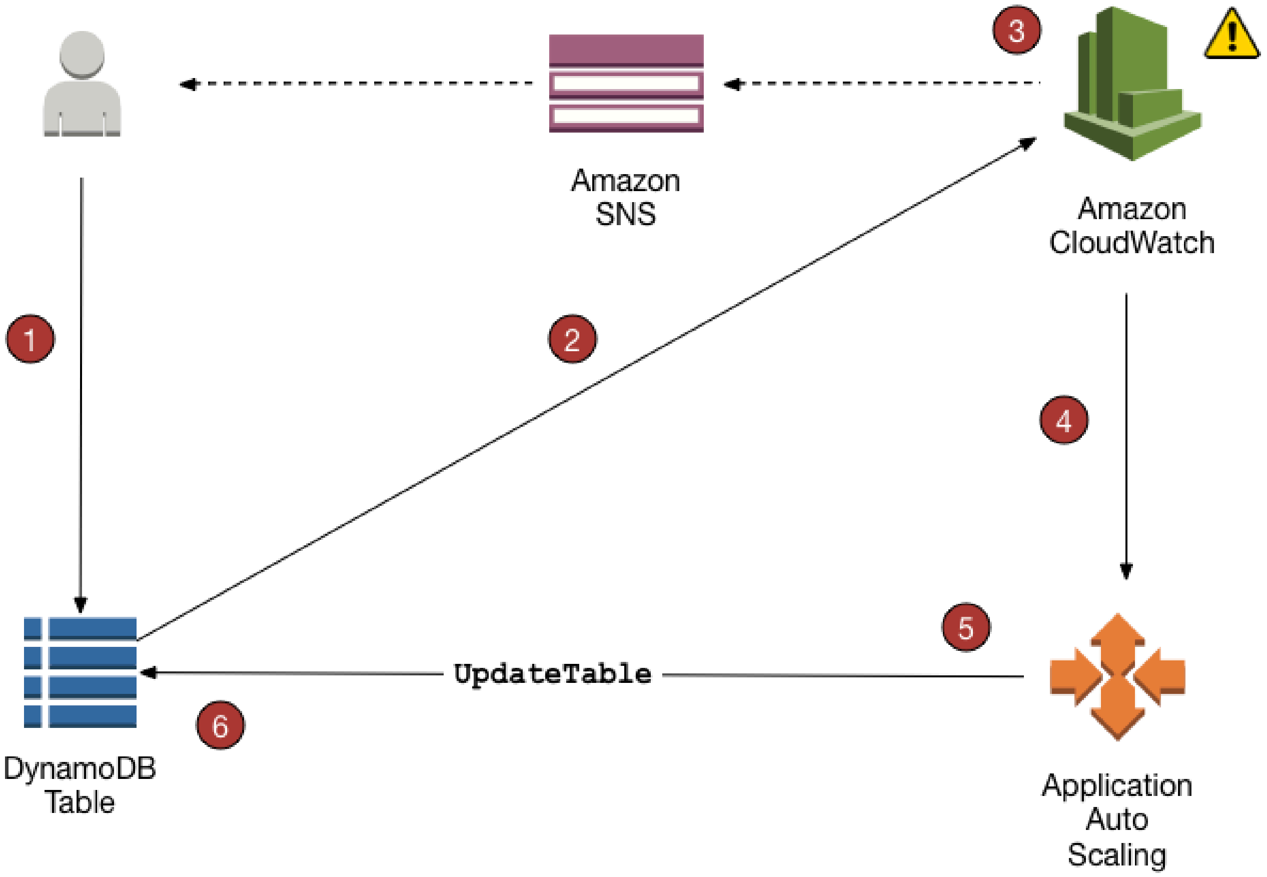
テーブルに加え、DynamoDB Auto Scaling はグローバルセカンダリインデックスもサポートします。すべてのグローバルセカンダリインデックスは、基本テーブルとは別に、固有のプロビジョンドスループット性能を持ちます。グローバルセカンダリインデックスのスケーリングポリシーを作成すると、Application Auto Scaling がインデックスのプロビジョンされたスループット設定を調整し、実際の使用率がターゲット使用率と同じか近い値で維持されるようになります。

DynamoDB Auto Scaling の仕組み

ⓘ Note

DynamoDB Auto Scaling の簡単な使用方法については、AWS Management Console と DynamoDB Auto Scaling の使用 を参照してください。

次の図表は、DynamoDB Auto Scaling によるテーブルのスループットキャパシティの管理方法について、高レベルの概要を示しています。



次のステップは、前の図に示された Auto Scaling のプロセスをまとめたものです。

1. DynamoDB テーブルの Application Auto Scaling ポリシーを作成します。
2. DynamoDB は、消費された容量メトリクスを Amazon CloudWatch に公開します。
3. テーブルの消費された容量が一定期間中にターゲット使用率を超える (または下回る) と、Amazon CloudWatch はアラームをトリガーします。コンソールでアラームを表示し、Amazon Simple Notification Service (Amazon SNS) を使用して通知を受け取ることができます。
4. CloudWatch アラームは、Application Auto Scaling をコールしてスケーリングポリシーを評価します。
5. Application Auto Scaling は `UpdateTable` リクエストを発行し、テーブルのプロビジョンされたスループットを調整します。
6. DynamoDB は `UpdateTable` リクエストを処理してテーブルのプロビジョンドスループット性能を動的に増減し、ターゲット使用率に近づけます。

DynamoDB Auto Scaling の仕組みを理解するため、ProductCatalog という名前のテーブルがあると仮定します。テーブルにはまれにデータがバルクロードされます。そのため、書き込みアクティビティが頻繁に生じることはありません。ただし、時間で変化する高度な読み込みアクティビティが生じています。ProductCatalog の Amazon CloudWatch メトリクスをモニタリングすることにより、テーブルに 1,200 ユニットの読み込み容量が必要である (アクティビティのピーク時に DynamoDB が読み込みリクエストをスロットリングしないため) と判断します。さらに、読み込みトラフィックが最も低い時点で、ProductCatalog には最少で 150 ユニットの読み込みキャパシティが必要です。スロットリングの防止の詳細については、「プロビジョンドキャパシティモードを使用した DynamoDB テーブルのスロットリング問題」を参照してください。

読み込みキャパシティ 150～1,200 ユニットの範囲内で、ProductCatalog に適したターゲット使用率 70% を決定します。ターゲット使用率は、プロビジョンされた容量単位に対して消費された容量単位の割合がパーセンテージで示されます。Application Auto Scaling は、ターゲット追跡アルゴリズムを使用して、ProductCatalog のプロビジョンされた読み込み容量が必要に応じて調整され、使用率が 70% またはその近くに留まるようにします。

> **Note**
>
> DynamoDB Auto Scaling は、実際のワークロードの増減が数分間維持された場合にのみ、プロビジョンされたスループット設定を変更します。アプリケーションオートスケーリングターゲット追跡アルゴリズムはターゲット使用率を選択した値の付近に長期に渡って維持しようとします。
> アクティビティの急激かつ短時間の上昇は、テーブルに組み込まれたバーストキャパシティで対応されます。詳細については、「バーストキャパシティ」を参照してください。

ProductCatalog テーブルの DynamoDB Auto Scaling を有効にするには、スケーリングポリシーを作成します。このポリシーでは、以下を指定します。

- 管理するテーブルまたはグローバルセカンダリインデックス
- 管理するキャパシティタイプ (読み込みキャパシティまたは書き込みキャパシティ)
- プロビジョニングされたスループット設定の上下の境界
- ターゲット使用率

スケーリングポリシーを作成すると、Application Auto Scaling はユーザーに代わって Amazon CloudWatch アラームのペアを作成します。各ペアはプロビジョニングされたスループット設定の上

下の境界を示します。CloudWatch アラームは、テーブルの実際の使用率が一定期間ターゲット使用率を逸脱したときにトリガーされます。

いずれかの CloudWatch アラームがトリガーされると、Amazon SNS は通知を送信します (有効にしている場合)。その後 CloudWatch アラームは Application Auto Scaling をコールし、`ProductCatalog` テーブルのプロビジョン容量を必要に応じて調整するように DynamoDB に通知します。

スケーリングイベント中、AWS Config は記録された設定項目ごとに課金されます。スケーリングイベントが発生すると、読み取り/書き込み自動スケーリングイベントごとに 4 つの CloudWatch アラームが作成されます。ProvisionedCapacity アラーム: ProvisionedCapacityLow、ProvisionedCapacityHigh、および ConsumedCapacity アラーム: AlarmHigh、AlarmLow です。その結果、合計 8 つのアラームが生成されます。そのため、AWS Config は、スケーリングイベントごとに 8 つの設定項目を記録します。

> Note
>
> また、DynamoDB のスケーリングを特定の時間に実行するようにスケジュールすることもできます。基本的な手順については、こちらを参照してください。

使用に関する注意事項

DynamoDB Auto Scaling の使用を開始する前に、以下を確認する必要があります。

- DynamoDB Auto Scaling は、自動スケーリングポリシーに従って、読み込み容量や書き込み容量を必要に応じて増加させます。Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ で説明されているように、すべての DynamoDB クォータは有効です。
- DynamoDB Auto Scaling により、プロビジョンされたスループット設定を手動で変更できなくなることはありません。これらの手動調整が、DynamoDB Auto Scaling に関連する既存の CloudWatch アラームに影響することはありません。
- 1 つ以上のグローバルセカンダリインデックスを持つテーブルの DynamoDB Auto Scaling を有効にする場合、これらのインデックスに Auto Scaling を均等に適用することをお勧めします。これにより、テーブルの書き込みと読み取りのパフォーマンスが向上し、スロットリングを回避できます。 auto scaling を有効にするには、AWS Management Console で [Apply same settings to global secondary indexes] (グローバルセカンダリインデックスに同じ設定を適用する) を選択します。詳細については、「既存のテーブルでの DynamoDB Auto Scaling の有効化」を参照してください。

- テーブルまたはグローバルテーブルレプリカを削除しても、関連するスケーラブルターゲット、スケーリングポリシー、または CloudWatch アラームが共に自動的に削除されることはありません。
- 既存のテーブルの GSI を作成する場合、その GSI の Auto Scaling は有効になりません。GSI を構築している間は、容量を手動で管理する必要があります。GSI のバックフィルが完了し、アクティブステータスに達すると、Auto Scaling は通常どおり動作します。

AWS Management Console と DynamoDB Auto Scaling の使用

AWS Management Console を使用して新しいテーブルを作成すると、Amazon DynamoDB Auto Scaling はデフォルトでそのテーブルに対して有効になります。コンソールを使用して、既存テーブルの Auto Scaling の有効化、Auto Scaling 設定の変更、Auto Scaling の無効化を行うこともできます。

> **Note**
>
> スケールインおよびスケールアウトのクールダウン時間の設定など、より高度な特徴については、AWS Command Line Interface (AWS CLI) を使用して DynamoDB Auto Scaling を管理します。詳細については、「AWS CLI を使用した DynamoDB Auto Scaling の管理」を参照してください。

トピック

開始する前に: DynamoDB Auto Scaling のアクセス許可をユーザーに付与する

AWS Identity and Access Management (IAM) の場合、AWS マネージドポリシー `DynamoDBFullAccess` は、DynamoDB コンソールを使用するために必要なアクセス許可を提供します。ただし、DynamoDB Auto Scaling の場合、ユーザーには追加アクセス許可が必要です。

⚠ Important

自動スケーリング対応のテーブルを削除するには、`application-autoscaling:*` アクセス許可が必要です。AWS マネージドポリシー `DynamoDBFullAccess` には、必要なアクセス許可が含まれています。

DynamoDB コンソールアクセスと DynamoDB Auto Scaling 用にユーザーを設定するには、ロールを作成し、そのロールに AmazonDynamoDBFullAccess ポリシーを追加します。次に、ロールをユーザーに割り当てます。

Auto Scaling を有効にした新しいテーブルの作成

ⓘ Note

DynamoDB Auto Scaling では、ユーザーに代わって Auto Scaling アクションを実行する、サービスリンクロール (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) の存在を必要とします。このロールは自動的に作成されます。詳細については、「Application Auto Scaling ユーザーガイド」の「Application Auto Scaling 用のサービスリンクロール」を参照してください。

Auto Scaling を有効にして新しいテーブルを作成するには

1. https://console.aws.amazon.com/dynamodb/ で DynamoDB コンソールを開きます。
2. [Create table] を選択します。
3. [テーブルを作成] ページで、[テーブル名] とプライマリキーを入力します。
4. [デフォルト設定] を選択すると、Auto Scaling を有効にしてテーブルが作成されます。

   それ以外のカスタム設定の場合:

   a. [設定をカスタマイズ] を選択します。
   b. [Read/write capacity settings] (読み込み/書き込みキャパシティ設定) セクションで、[Provisioned] (プロビジョンド) キャパシティモードを選択し、[Read capacity] (読み込みキャパシティ)、[Write capacity] (書き込みキャパシティ)、または両方に対して、Auto Scaling を On に設定します。これらのそれぞれについて、テーブルに必要なスケーリング

ポリシーを設定し、オプションでテーブルのすべてのグローバルセカンダリインデックスを設定します。

- 最小キャパシティーユニット - Auto Scaling 範囲の下限を入力します。
- 最大キャパシティーユニット - Auto Scaling 範囲の上限を入力します。
- ターゲット使用率 — テーブルの目標使用率を入力します。

> Note
>
> 新しいテーブルのグローバルセカンダリインデックスを作成する場合、作成時のインデックスの容量は、ベーステーブルの容量と同じになります。インデックスの容量は、テーブルの作成後にテーブルの設定で変更できます。

5. すべての設定が正しいことを確認したら、[テーブルを作成] を選択します。Auto Scaling パラメータを使用してテーブルが作成されます。

既存のテーブルでの DynamoDB Auto Scaling の有効化

> Note
>
> DynamoDB Auto Scaling では、ユーザーに代わって Auto Scaling アクションを実行する、サービスリンクロール (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) の存在を必要とします。このロールは自動的に作成されます。詳細については、「Application Auto Scaling 用のサービスリンクロール」を参照してください。

既存のテーブルに対して DynamoDB Auto Scaling を有効にするには

1. https://console.aws.amazon.com/dynamodb/ で DynamoDB コンソールを開きます。
2. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
3. 使用するテーブルを選択し、[追加設定] タブを選択します。
4. [読み取りキャパシティー] セクションで、[編集] を選択します。
5. [キャパシティーモード] セクションで、[プロビジョンド] を選択します。
6. [Table capacity] (テーブルキャパシティ) セクションで、[Read capacity] (読み込みキャパシティ)、[Write capacity] (書き込みキャパシティ)、または両方に対して、Auto scaling を On に設

定します。これらのそれぞれについて、テーブルに必要なスケーリングポリシーを設定し、オプションでテーブルのすべてのグローバルセカンダリインデックスを設定します。

- 最小キャパシティーユニット - Auto Scaling 範囲の下限を入力します。
- 最大キャパシティーユニット - Auto Scaling 範囲の上限を入力します。
- ターゲット使用率 — テーブルの目標使用率を入力します。
- [すべてのグローバルセカンダリインデックスに同じキャパシティー読み取り/書き込みキャパシティー設定を使用する] - グローバルセカンダリインデックスがベーステーブルと同じ Auto Scaling ポリシーを使用するかどうかを選択します。

> Note
>
> 最高のパフォーマンスを実現するには、[すべてのグローバルセカンダリインデックスに同じ読み取り/書き込み容量設定を適用する] を有効にすることをお勧めします。このオプションを使用すれば、DynamoDB Auto Scaling は、ベーステーブル上のすべてのグローバルセカンダリインデックスを均一にスケーリングできます。これには、既存のグローバルセカンダリインデックスと、将来このテーブル用に作成するその他のインデックスが含まれます。
> このオプションを有効にすると、個々のグローバルセカンダリインデックスにスケーリングポリシーを設定できなくなります。

7. すべての設定が正しいことを確認したら、[保存] を選択します。

コンソールでの Auto Scaling アクティビティの表示

アプリケーションがテーブルへの読み込みおよび書き込みトラフィックを送ると、DynamoDB Auto Scaling はテーブルのスループット設定を動的に変更します。Amazon CloudWatch は、すべての DynamoDB テーブルとセカンダリインデックスについて、プロビジョニングされた容量、消費された容量、スロットルイベント、レイテンシー、およびその他のメトリクスを追跡します。

DynamoDB コンソールでこれらのメトリクスを表示するには、操作するテーブルを選択し、[モニタリング] タブを選択します。テーブルメトリクスのカスタマイズ可能なビューを作成するには、[CloudWatch ですべてを表示] を選択します。

DynamoDB Auto Scaling 設定の変更または無効化

AWS Management Console を使用して、DynamoDB Auto Scaling 設定を変更できます。これを行うには、テーブルの [追加設定] タブに移動し、[読み取り/書き込みキャパシティー] セクションで [編

集] を選択します。これらの設定の詳細については、「既存のテーブルでの DynamoDB Auto Scaling の有効化」をご参照ください。

AWS CLI を使用した DynamoDB Auto Scaling の管理

AWS Management Console を使用する代わりに、AWS Command Line Interface (AWS CLI) を使用して、Amazon DynamoDB Auto Scaling を管理できます。このセクションのチュートリアルでは、DynamoDB Auto Scaling を管理するために AWS CLI をインストールして設定する方法を示します。このチュートリアルでは、以下の作業を行います。

- `TestTable` という DynamoDB テーブルの作成 初期スループット設定では、読み込み容量単位数が 5、書き込み容量単位が 5 です。
- `TestTable` 用アプリケーションの Auto Scaling ポリシーを作成します。このポリシーでは、プロビジョニングされた書き込みキャパシティに対する消費された書き込みキャパシティの割合を 50% に維持することを目指します。このメトリクスの範囲は 5～10 書き込みキャパシティユニットです。(Application Auto Scaling では、この範囲を超えるスループットを調整することはできません)。
- Python プログラムを実行して、書き込みトラフィックを `TestTable` に送ります。目標比率が一定期間 50% を超えると、Application Auto Scaling は DynamoDB に通知して `TestTable` のスループットを上方に調整して、50% の目標使用率を維持します。
- DynamoDB が `TestTable` のプロビジョンされた書き込み容量を正常に調整したことを確認します。

> Note
>
> また、DynamoDB のスケーリングを特定の時間に実行するようにスケジュールすることもできます。基本的な手順については、こちらを参照してください。

トピック

- ステップ 5: Application Auto Scaling アクションを表示する
- (オプション) ステップ 6: クリーンアップする

開始する前に

チュートリアルを開始する前に、以下のタスクを完了します。

AWS CLI をインストールする

まだ AWS CLI をインストールして設定していない場合は、インストールして設定する必要があります。これを行うには、AWS Command Line Interface のユーザーガイドの手順に従います。

- AWS CLI のインストール
- AWS CLI の設定

Python のインストール

このチュートリアルの一部では、Python プログラムを実行する必要があります (ステップ 4: 書き込みトラフィックを TestTable に送る を参照)。まだインスタンスしていない場合は、Python をダウンロードできます。

ステップ 1: DynamoDB テーブルを作成する

このステップでは、AWS CLI を使用して `TestTable` を作成します。プライマリキーは、pk (パーティションキー) と sk (ソートキー) で構成されます。これらの属性はいずれも、`Number` 型になります。初期スループット設定では、読み込み容量単位数が 5、書き込み容量単位が 5 です。

1. 以下の AWS CLI コマンドを使用してターゲットを作成します。

```
aws dynamodb create-table \
    --table-name TestTable \
    --attribute-definitions \
        AttributeName=pk,AttributeType=N \
        AttributeName=sk,AttributeType=N \
    --key-schema \
        AttributeName=pk,KeyType=HASH \
        AttributeName=sk,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

2. テーブルのステータスを確認するには、次のコマンドを使用します。

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

ステータスが ACTIVE になったら、テーブルは使用できる状態になります。

ステップ 2: スケーラブルなターゲットを登録する

次に、Application Auto Scaling を使用して、テーブルの書き込み容量をスケーラブルなターゲットとして登録します。これにより、Application Auto Scaling は、TestTable にプロビジョンされた書き込み容量を調整できますが、容量単位は 5～10 の範囲内に限られます。

> Note
>
> DynamoDB Auto Scaling では、ユーザーに代わって Auto Scaling アクションを実行する、サービスリンクロール (AWSServiceRoleForApplicationAutoScaling_DynamoDBTable) の存在を必要とします。このロールは自動的に作成されます。詳細については、「Application Auto Scaling ユーザーガイド」の「Application Auto Scaling 用のサービスリンクロール」を参照してください。

1. 次のコマンドを入力して、スケーラブルなターゲットを登録します。

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

2. 次のコマンドを使用して登録を確認します。

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

Note

グローバルセカンダリインデックスに対してスケーラブルターゲットを登録することもできます。たとえば、グローバルセカンダリインデックス (「test-index」) の場合、リソース ID とスケーラブルディメンションの引数は適切に更新されます。

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable/index/test-index" \
    --scalable-dimension "dynamodb:index:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

ステップ 3: スケーリングポリシーを作成する

このステップでは、TestTable のスケーリングポリシーを作成します。ポリシーでは、Application Auto Scaling がテーブルのプロビジョンされたスループットを調整できる詳細と、その場合に実行するアクションを定義します。このポリシーは、前のステップで定義したスケーラブルターゲットに関連付けます (TestTable テーブルの容量単位を書き込みます)。

ポリシーには、次の要素が含まれます。

- PredefinedMetricSpecification — Application Auto Scaling が調整できるメトリクス。DynamoDB の場合、次の値が PredefinedMetricType の有効な値です。
    - DynamoDBReadCapacityUtilization
    - DynamoDBWriteCapacityUtilization
- ScaleOutCooldown — プロビジョンされたスループットを増加させる各 Application Auto Scaling イベント間の最小時間 (秒単位)。このパラメータを使用すると、Application Auto Scaling は、実際のワークロードに応じてスループットを継続的に増加させますが、積極的に増加させることはできません。ScaleOutCooldown のデフォルトの設定は 0 です。
- ScaleInCooldown — プロビジョンされたスループットを低下させる各 Application Auto Scaling イベント間の最小時間 (秒単位)。このパラメータを使用すると、Application Auto Scaling はスループットを徐々に、予測どおりに低下させることができます。ScaleInCooldown のデフォルトの設定は 0 です。

- TargetValue — Application Auto Scaling は、プロビジョンされた容量に対する消費容量の比率がこの値またはその近くに留まるようにします。TargetValue をパーセンテージとして定義します。

Note

TargetValue がどのように機能するかをさらに理解するために、書き込み容量単位が 200 で、プロビジョンされたスループット設定を持つテーブルがあるとします。このテーブルのスケーリングポリシーを作成することにしました。TargetValue は 70% です。
ここで、実際の書き込みスループットが 150 容量単位になるように、テーブルへの書き込みトラフィックを駆動し始めたとします。消費とプロビジョンの比率は現在 (150/200)、つまり 75% です。この比率は目標を超えているため、Application Auto Scaling はプロビジョンされた書き込み容量を 215 に増やし、比率が (150/215)、つまり 69.77% になるようにします。これは、可能な限り TargetValue に近いですが、超えないようにしてください。

TestTable の場合、TargetValue を 50% に設定します。Application Auto Scaling は、テーブルのプロビジョンされたスループットを 5～10 容量単位の範囲内で調整し (ステップ 2: スケーラブルなターゲットを登録する を参照)、プロビジョンされた消費量とプロビジョンされた比率が 50% またはそれに近いままになるようにします。ScaleOutCooldown と ScaleInCooldown の値を 60 秒に設定します。

1. 次の内容で、scaling-policy.json というファイルを作成します。

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. 次の AWS CLI コマンドを使用してポリシーを作成します。

```
aws application-autoscaling put-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy" \
    --policy-type "TargetTrackingScaling" \
    --target-tracking-scaling-policy-configuration file://scaling-policy.json
```

3. 出力では、Application Auto Scaling が 2 つの Amazon CloudWatch アラームを作成したことに注意してください。スケーリング目標範囲の上限と下限に対応します。
4. 以下の AWS CLI コマンドを使用して、スケーリングポリシーの詳細を表示します。

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. 出力で、ポリシー設定が ステップ 2: スケーラブルなターゲットを登録する と ステップ 3: スケーリングポリシーを作成する からの仕様と一致することを確認します。

ステップ 4: 書き込みトラフィックを TestTable に送る

これで、`TestTable` にデータを書き込むことでスケーリングポリシーをテストできます。これを行うには、Python プログラムを実行します。

1. 次の内容で、`bulk-load-test-table.py` というファイルを作成します。

```
import boto3
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table("TestTable")

filler = "x" * 100000

i = 0
while (i < 10):
    j = 0
    while (j < 10):
        print (i, j)

        table.put_item(
            Item={
                'pk':i,
                'sk':j,
                'filler':{"S":filler}
            }
        )
        j += 1
    i += 1
```

2. プログラムを実行するには、次のコマンドを入力します。

   ```
   python bulk-load-test-table.py
   ```

   `TestTable` のプロビジョンされた書き込み容量は非常に低いため (5 書き込み容量単位)、書き込みスロットリングが原因でプログラムが停止することがあります。これは想定される動作です。

   次のステップに進む間、プログラムを実行し続けます。

ステップ 5: Application Auto Scaling アクションを表示する

このステップでは、ユーザーに代わって開始された Application Auto Scaling アクションを表示します。また、Application Auto Scaling が `TestTable` のプロビジョンされた書き込み容量を更新したことを確認します。

1. 次のコマンドを入力して、Application Auto Scaling アクションを表示します。

   ```
   aws application-autoscaling describe-scaling-activities \
       --service-namespace dynamodb
   ```

   Python プログラムの実行中に、このコマンドを時折再実行します。(スケーリングポリシーが呼び出されるまで数分かかります。) 結果的に、次のような出力が表示されます。

   ```
   ...
   {
       "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
       "Description": "Setting write capacity units to 10.",
       "ResourceId": "table/TestTable",
       "ActivityId": "0cc6fb03-2a7c-4b51-b67f-217224c6b656",
       "StartTime": 1489088210.175,
       "ServiceNamespace": "dynamodb",
       "EndTime": 1489088246.85,
       "Cause": "monitor alarm AutoScaling-table/TestTable-
   AlarmHigh-1bb3c8db-1b97-4353-baf1-4def76f4e1b9 in state ALARM triggered policy
    MyScalingPolicy",
   ```

```
    "StatusMessage": "Successfully set write capacity units to 10. Change
 successfully fulfilled by dynamodb.",
    "StatusCode": "Successful"
},
...
```

これは、Application Auto Scaling が DynamoDB に UpdateTable リクエストを発行したことを示しています。

2. DynamoDB がテーブルの書き込み容量を増やしたことを確認するには、次のコマンドを入力します。

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

WriteCapacityUnits は 5 から 10 にスケーリングされている必要があります。

(オプション) ステップ 6: クリーンアップする

このチュートリアルでは、いくつかのリソースを作成しました。これらのリソースが不要になった場合は、削除できます。

1. TestTable のスケーリングポリシーを削除します。

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. スケーラブルなターゲットを登録解除します。

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. TestTable テーブルを削除します。

```
aws dynamodb delete-table --table-name TestTable
```

AWS SDK を使用した Amazon DynamoDB テーブルでの Auto Scaling の設定

AWS Management Console と AWS Command Line Interface (AWS CLI) を使用するほかに、Amazon DynamoDB Auto Scaling とやり取りするアプリケーションを作成することもできます。このセクションには、この機能をテストするために使用できる 2 つの Java プログラムが含まれています。

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

テーブルの Application Auto Scaling の有効化

次のプログラムは、DynamoDB テーブル (`TestTable`) の Auto Scaling ポリシーをセットアップする例を示しています。プログラムの流れは次のようになります。

- `TestTable` のスケーラブルなターゲットとして書き込みキャパシティユニットを登録します。このメトリクスの範囲は 5 ~ 10 書き込みキャパシティーユニットです。
- スケーラブルなターゲットを作成したら、次にターゲットの追跡設定を作成します。このポリシーでは、プロビジョニングされた書き込みキャパシティに対する消費された書き込みキャパシティの割合を 50% に維持することを目指します。
- 次に、ターゲットの追跡設定に基づいてスケーリングポリシーを作成します。

Note

テーブルまたはグローバルテーブルレプリカを手動で削除しても、関連するスケーラブルターゲット、スケーリングポリシー、または CloudWatch アラームは自動的には削除されません。

プログラムでは、有効な Application Auto Scaling サービスリンクロールの Amazon リソースネーム (ARN) を指定する必要があります。(例: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`。) 次のプログラムの `SERVICE_ROLE_ARN_GOES_HERE` を実際の ARN に置き換えます。

```
package com.amazonaws.codesamples.autoscaling;

import com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
```

```
import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClientBuilder;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.MetricType;
import com.amazonaws.services.applicationautoscaling.model.PolicyType;
import
 com.amazonaws.services.applicationautoscaling.model.PredefinedMetricSpecification;
import com.amazonaws.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
import
 com.amazonaws.services.applicationautoscaling.model.TargetTrackingScalingPolicyConfiguration;

public class EnableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = (AWSApplicationAutoScalingClient)
 AWSApplicationAutoScalingClientBuilder
   .standard().build();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Define the scalable target
  RegisterScalableTargetRequest rstRequest = new RegisterScalableTargetRequest()
    .withServiceNamespace(ns)
    .withResourceId(resourceID)
    .withScalableDimension(tableWCUs)
    .withMinCapacity(5)
    .withMaxCapacity(10)
    .withRoleARN("SERVICE_ROLE_ARN_GOES_HERE");

  try {
```

```
  aaClient.registerScalableTarget(rstRequest);
 } catch (Exception e) {
  System.err.println("Unable to register scalable target: ");
  System.err.println(e.getMessage());
 }

 // Verify that the target was created
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);
 try {
  DescribeScalableTargetsResult dsaResult =
aaClient.describeScalableTargets(dscRequest);
  System.out.println("DescribeScalableTargets result: ");
  System.out.println(dsaResult);
  System.out.println();
 } catch (Exception e) {
  System.err.println("Unable to describe scalable target: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Configure a scaling policy
 TargetTrackingScalingPolicyConfiguration targetTrackingScalingPolicyConfiguration =
new TargetTrackingScalingPolicyConfiguration()
   .withPredefinedMetricSpecification(
     new PredefinedMetricSpecification()
       .withPredefinedMetricType(MetricType.DynamoDBWriteCapacityUtilization))
   .withTargetValue(50.0)
   .withScaleInCooldown(60)
   .withScaleOutCooldown(60);

 // Create the scaling policy, based on your configuration
 PutScalingPolicyRequest pspRequest = new PutScalingPolicyRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID)
   .withPolicyName("MyScalingPolicy")
   .withPolicyType(PolicyType.TargetTrackingScaling)

.withTargetTrackingScalingPolicyConfiguration(targetTrackingScalingPolicyConfiguration);
```

```
  try {
   aaClient.putScalingPolicy(pspRequest);
  } catch (Exception e) {
   System.err.println("Unable to put scaling policy: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scaling policy was created
  DescribeScalingPoliciesRequest dspRequest = new DescribeScalingPoliciesRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   DescribeScalingPoliciesResult dspResult =
 aaClient.describeScalingPolicies(dspRequest);
   System.out.println("DescribeScalingPolicies result: ");
   System.out.println(dspResult);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println("Unable to describe scaling policy: ");
   System.err.println(e.getMessage());
  }

 }

}
```

テーブルの Application Auto Scaling の無効化

次のプログラムは、前述のプロセスを元に戻します。Auto Scaling ポリシーを削除し、スケーラブルなターゲットを登録解除します。

```
package com.amazonaws.codesamples.autoscaling;

import com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import com.amazonaws.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DeregisterScalableTargetRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
```

```
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;

public class DisableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = new
 AWSApplicationAutoScalingClient();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Delete the scaling policy
  DeleteScalingPolicyRequest delSPRequest = new DeleteScalingPolicyRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID)
    .withPolicyName("MyScalingPolicy");

  try {
   aaClient.deleteScalingPolicy(delSPRequest);
  } catch (Exception e) {
   System.err.println("Unable to delete scaling policy: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scaling policy was deleted
  DescribeScalingPoliciesRequest descSPRequest = new DescribeScalingPoliciesRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   DescribeScalingPoliciesResult dspResult =
 aaClient.describeScalingPolicies(descSPRequest);
   System.out.println("DescribeScalingPolicies result: ");
   System.out.println(dspResult);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println("Unable to describe scaling policy: ");
   System.err.println(e.getMessage());
  }

  System.out.println();

  // Remove the scalable target
  DeregisterScalableTargetRequest delSTRequest = new DeregisterScalableTargetRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   aaClient.deregisterScalableTarget(delSTRequest);
  } catch (Exception e) {
   System.err.println("Unable to deregister scalable target: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scalable target was removed
  DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceIds(resourceID);

  try {
   DescribeScalableTargetsResult dsaResult =
 aaClient.describeScalableTargets(dscRequest);
   System.out.println("DescribeScalableTargets result: ");
   System.out.println(dsaResult);
   System.out.println();
  } catch (Exception e) {
   System.err.println("Unable to describe scalable target: ");
   System.err.println(e.getMessage());
  }

 }

}
```

リザーブドキャパシティ

標準のテーブルクラスを使用するプロビジョンドキャパシティテーブルの場合、DynamoDB は読み込みおよび書き込みキャパシティ用にリザーブドキャパシティを購入する機能を提供しています。リザーブドキャパシティの購入は、割引料金と引き換えに、契約期間中のプロビジョニングされたスループットキャパシティの最小量に対して支払う契約です。

> Note
>
> レプリケートされた書き込みキャパシティユニット (rWCU) に対しては、リザーブドキャパシティを購入できません。リザーブドキャパシティは、キャパシティを購入したリージョンにのみ適用されます。リザーブドキャパシティは、DynamoDB 標準-IA テーブルクラスまたはオンデマンドキャパシティモードを使用するテーブルでも使用できません。

リザーブドキャパシティは、100 個の WCU または 100 個の RCU の割り当て単位で購入します。最小のリザーブドキャパシティは、100 個のキャパシティユニット (読み込みまたは書き込み) です。DynamoDB リザーブドキャパシティは、1 年間のコミットメント、または限定リージョンでの 3 年間のコミットメントとして提供されます。1 年契約では標準料金から最大 54%、3 年契約では標準料金から最大 77% の割引を受けることができます。購入方法と購入時期の詳細については、「Amazon DynamoDB リザーブドキャパシティ」を参照してください。

DynamoDB リザーブドキャパシティを購入すると、1 回限りの一部前払い料金を支払うことで、コミットしたプロビジョンド使用量を割引時間料金で利用できます。実際の使用量に関係なく、コミットしたプロビジョンド使用量全体に料金を支払うため、コスト削減は使用量と密接に関連します。購入したリザーブドキャパシティを超えてプロビジョニングしたキャパシティには、標準のプロビジョンドキャパシティの料金が請求されます。読み込みキャパシティーユニットおよび書き込みキャパシティーユニットを事前に予約することで、プロビジョニングされたキャパシティーコストの大幅なコスト削減を実現できます。

リザーブドキャパシティを販売またはキャンセルしたり、別のリージョンやアカウントに移管したりすることはできません。

> Note
>
> リザーブドキャパシティは、組織専用のキャパシティではありません。これは、アカウントでの読み込みおよび/または書き込みにに対するプロビジョンドキャパシティの使用に適用される請求割引です。

バーストキャパシティとアダプティブキャパシティ

スループット例外に伴うスロットリングを最小限に抑えるために、DynamoDB はバーストキャパシティを使用して使用量のスパイクを処理します。DynamoDB は、アダプティブキャパシティを使用して、アクセスパターンの不均衡に対応できるようにします。

バーストキャパシティ

DynamoDB は、バーストキャパシティにより、スループットプロビジョニングに柔軟性をもたらします。利用可能なスループットを使い切っていない場合、DynamoDB は、キャパシティの未使用分を、後のスループットのバーストに備えて留保しておき、使用量のスパイクに対応します。バーストキャパシティーにより、スロットリングされていた可能性のある読み込みまたは書き込みリクエストが成功します。

現在、DynamoDB は、未使用の読み込みおよび書き込みキャパシティを最大 5 分 (300 秒) 保持します。読み込みまたは書き込みアクティビティのバーストが発生した場合、これらの余分なキャパシティユニットをすばやく消費できます。これは、テーブルに定義した 1 秒あたりのプロビジョンドスループットキャパシティよりも高速です。

また、DynamoDB はバックグラウンドメンテナンスやその他のタスクのために予告なしにバーストキャパシティを消費する場合があります。

これらのバーストキャパシティの詳細は将来変更される可能性があります。

アダプティブキャパシティ

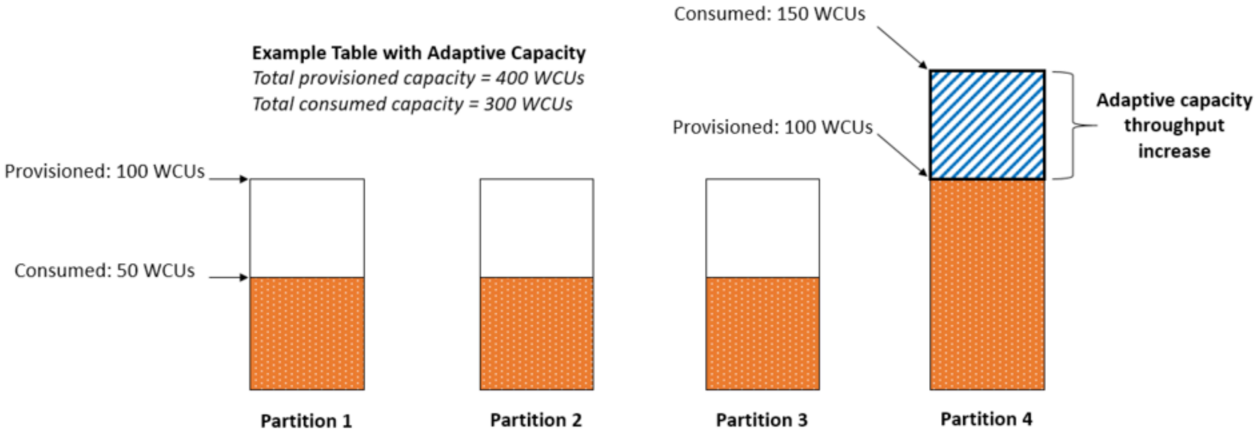
DynamoDB では、パーティション間にデータを自動的に分散します。これらは、AWS クラウドの複数のサーバーに保存されます。読み込みおよび書き込みアクティビティは常に均等に分散できるとは限りません。データアクセスが不均等の場合は、他のパーティションと比較して、「ホット」パーティションに大容量の読み書きトラフィックが送信されることがあります。パーティションの読み込みおよび書き込みオペレーションは個別に管理されるため、単一のパーティションが 3,000 を超える読み込みオペレーションまたは 1,000 を超える書き込みオペレーションを受け取ると、スロットリングが発生します。アダプティブキャパシティーでは、さらに多くのトラフィックを受け取るパーティションのスループットキャパシティーを自動的に増加させます。

不均等なアクセスパターンに対応できるように、DynamoDB アダプティブキャパシティを使用することで、スロットリングすることなく、アプリケーションでホットパーティションに対して読み書きを継続できます。ただし、トラフィックは、プロビジョニングされた合計容量またはパーティション

の最大容量を超えないものとします。アダプティブキャパシティでは、さらに多くのトラフィックを受け取るパーティションのスループットキャパシティを自動的かつ瞬時に増加させます。

次の図は、アダプティブキャパシティの機能を示しています。このテーブルの例のプロビジョニングでは、4 つのパーティション間で 400 WCU が均等に共有されることで、各パーティションで 1 秒間に最大 100 WCU を維持できます。パーティション 1、2、3 はそれぞれ、50 WCU/秒の書き込みトラフィックを受け取ります。パーティション 4 は 150 WCU/秒の書き込みトラフィックを受け取ります。このホットパーティションは、未使用のバーストキャパシティがある場合でも書き込みトラフィックを受け入ることができます。しかし、最終的には 1 秒間に 100 WCU を超えるトラフィックのスロットリングとなります。

DynamoDB アダプティブキャパシティは、パーティション 4 のキャパシティを増大して応答するため、このパーティションはスロットリングされることなく、150 WCU/秒のより高いワークロードを維持できます。


Example Table with Adaptive Capacity
Total provisioned capacity = 400 WCUs
Total consumed capacity = 300 WCUs
Provisioned: 100 WCUs
Consumed: 50 WCUs
Consumed: 150 WCUs
Provisioned: 100 WCUs
Adaptive capacity throughput increase
Partition 1
Partition 2
Partition 3
Partition 4

アダプティブキャパシティはすべての DynamoDB テーブルで自動的に有効化されます。追加料金はありません。アダプティブキャパシティを明示的に有効化または無効化する必要はありません。

アクセス頻度の高い項目を分離する

お使いのアプリケーションが 1 つ以上の項目に偏った高いトラフィックを送る場合、アクセス頻度の高い項目が同一パーティション内に存在しないように、アダプティブキャパシティによってパーティションのバランスが再調整されます。アクセス頻度の高い項目を分離することで、ワークロードが単一のパーティションのスループットクォータを超えることから生じるリクエストのスロットリングの可能性が低くなります。ソートキーの単調な増減によって追跡されるトラフィックが項目のコレクションでない限り、項目のコレクションをソートキー別にセグメントに分割することもできます。

アプリケーションから単一の項目に大量のトラフィックが一貫して送信される場合、アダプティブキャパシティはデータを再調整して、そのアクセス頻度の高い単一の項目のみをパーティションに

含めることがあります。この場合、DynamoDB では、単一項目のプライマリキーに対して、最大 3,000 RCU および 1,000 WCU のパーティションのスループットを提供できます。アダプティブキャパシティでは、テーブルにローカルセカンダリインデックスがある場合、テーブルの複数のパーティション間で項目コレクションを分割しません。

DynamoDB のセットアップ

Amazon DynamoDB ウェブサービスに加えて、AWS はコンピュータで実行できる DynamoDB のダウンロード可能バージョンを提供します。ダウンロード可能なバージョンは、コードの開発とテストに役立ちます。これを使用することで、DynamoDB ウェブサービスにアクセスせずに、ローカルでアプリケーションを書き込んでテストすることができます。

このセクションのトピックでは、DynamoDB (ダウンロード可能バージョン) と DynamoDB ウェブサービスをセットアップする方法について説明します。

トピック

- DynamoDB local (ダウンロード可能バージョン) のセットアップ
- DynamoDB (ウェブサービス) の設定

DynamoDB local (ダウンロード可能バージョン) のセットアップ

ダウンロード可能なバージョンの Amazon DynamoDB では、DynamoDB ウェブサービスにアクセスせずに、アプリケーションを開発してテストすることができます。代わりに、データベースはコンピュータ上で自己完結型となります。アプリケーションを本番稼働環境にデプロイする準備ができたら、コード内のローカルエンドポイントを削除します。その後、これは DynamoDB ウェブサービスを指します。

このローカルバージョンを使用することで、スループットやデータストレージ、データ転送料金を節約しやすくなります。また、アプリケーションを開発している間インターネットに接続しておく必要はありません。

DynamoDB local は、ダウンロード (JRE が必要)、Apache Maven 依存関係、または Docker イメージとして使用できます。

Amazon DynamoDB ウェブサービスを代わりに使用する場合は、「 DynamoDB (ウェブサービス) の設定 」を参照してください。

トピック

- コンピュータ上で DynamoDB をローカルでデプロイする
- DynamoDB local の使用に関する注意事項
- DynamoDB local のリリース履歴

- DynamoDB local のテレメトリ

コンピュータ上で DynamoDB をローカルでデプロイする

⚠ Important

DynamoDB Local jar は、ここに記載されている AWS CloudFront ディストリビューションリンクからダウンロードできます。2025 年 1 月 1 日以降、古い S3 ディストリビューションバケットはアクティブではなくなり、DynamoDB Local は CloudFront ディストリビューションリンクを通じてのみ配信されます。

DynamoDB Local には、DynamoDB Local v2.x (現行) と DynamoDB Local v1.x (レガシー) の 2 つのメジャーバージョンがあります。バージョン 2.x (現行) は Java ランタイム環境の最新バージョンをサポートし、Maven プロジェクトの jakarta.* 名前空間と互換性があるため、できる限りバージョン 2.x を使用してください。DynamoDB Local v1.x は、2025 年 1 月 1 日に標準サポートが終了します。この日を過ぎると、v1.x はアップデートやバグの修正が行われなくなります。

ⓘ Note

DynamoDB local の `AWS_ACCESS_KEY_ID` には、文字 (A～Z、a～z) と数字 (0～9) のみを使用できます。

DynamoDB Local のダウンロード

これらのステップに従って DynamoDB をコンピュータにセットアップして実行します。

コンピュータに DynamoDB をセットアップするには

1. DynamoDB Local は、次のいずれかの場所から無料でダウンロードできます。

ダウンロードリンク	チェックサム
.tar.gz | .zip	.tar.gz.sha256 | .zip.sha256

> ⚠ Important
>
> コンピュータで DynamoDB v2.4.0 以降を実行するには、Java ランタイム環境 (JRE) バージョン 17.x 以降が必要です。アプリケーションは、以前のバージョンの JRE では動作しません。

2. アーカイブをダウンロードしたら、内容を抽出し、抽出されたディレクトリを任意の場所にコピーします。

3. コンピュータで DynamoDB を開始するには、コマンドプロンプトウィンドウを開き、`DynamoDBLocal.jar` を抽出したディレクトリに移動し、次のコマンドを入力します。

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

> ⓘ Note
>
> Windows PowerShell を使用している場合は、パラメータ名または名前全体と値を次のように囲んでください。
>
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
>
> DynamoDB は、停止するまで受信リクエストを処理します。DynamoDB を停止するには、コマンドプロンプトで Ctrl+C を押します。
>
> DynamoDB は、デフォルトではポート 8000 を使用します。ポート 8000 を使用できない場合、このコマンドにより例外がスローされます。DynamoDB ランタイムオプション (`-port` を含む) の詳細なリストを表示するには、次のコマンドを入力します。
>
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

4. プログラムまたは AWS Command Line Interface (AWS CLI) を介して DynamoDB にアクセスする前に、アプリケーションで認証が有効になるように認証情報を設定する必要があります。以下の例に示されているように、ダウンロード可能な DynamoDB には、認証情報が必要です。

```
AWS Access Key ID: "fakeMyKeyId"
AWS Secret Access Key: "fakeSecretAccessKey"
Default Region Name: "fakeRegion"
```

aws configure の AWS CLI コマンドを使用して、認証情報を設定できます。詳細については、「」を参照してくださいAWS CLI の使用

5. アプリケーションの書き込みを開始します。AWS CLI を使用してローカルで実行中の DynamoDB にアクセスするには、--endpoint-url パラメータを使用します。たとえば、次のコマンドを使用して、DynamoDB テーブルを一覧表示します。

   ```
   aws dynamodb list-tables --endpoint-url http://localhost:8000
   ```

DynamoDB Local を Docker イメージとして実行する

Amazon DynamoDB のダウンロード可能バージョンは、Docker イメージとして入手できます。詳細については、「dynamodb-local」を参照してください。DynamoDB Local の現在のバージョンを確認するには、次のコマンドを入力します。

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -version
```

AWS Serverless Application Model (AWS SAM) で構築した REST アプリケーションの一部として DynamoDB local を使用する例については、「注文を管理するための SAM DynamoDB アプリケーション」を参照してください。このサンプルアプリケーションは、DynamoDB local をテストで使用する方法を示しています。

DynamoDB local のコンテナも使用するマルチコンテナアプリケーションを実行する場合は、Docker Compose を使用して、アプリケーション内のすべてのサービス (DynamoDB local を含む) を定義して実行します。

Docker Compose を使用して DynamoDB local をインストールして実行するには

1. Docker Desktop をダウンロードしてインストールします。
2. 以下のコードをファイルにコピーし、docker-compose.yml として保存します。

   ```
   version: '3.8'
   services:
    dynamodb-local:
      command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
      image: "amazon/dynamodb-local:latest"
      container_name: dynamodb-local
      ports:
        - "8000:8000"
   ```

```
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
```

アプリケーションと DynamoDB local を別々のコンテナにしたい場合は、次の yaml ファイルを使用します。

```
version: '3.8'
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
 app-node:
   depends_on:
     - dynamodb-local
   image: amazon/aws-cli
   container_name: app-node
   ports:
    - "8080:8080"
   environment:
     AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
     AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
   command:
     dynamodb describe-limits --endpoint-url http://dynamodb-local:8000 --region
 us-west-2
```

この docker-compose.yml スクリプトは、`app-node` コンテナと `dynamodb-local` コンテナを作成します。スクリプトは、AWS CLI を使用して `app-node` コンテナに接続し、アカウントとテーブルの制限を記述するコマンドを `dynamodb-local` コンテナで実行します。

独自のアプリケーションイメージで使用するには、以下の例の `image` 値をアプリケーションの値に置き換えます。

```
version: '3.8'
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
 app-node:
   image: location-of-your-dynamodb-demo-app:latest
   container_name: app-node
   ports:
     - "8080:8080"
   depends_on:
     - "dynamodb-local"
   links:
     - "dynamodb-local"
   environment:
     AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
     AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
     REGION: 'eu-west-1'
```

> ⓘ Note
>
> YAML スクリプトでは、AWS アクセスキーと AWS シークレットキーを指定する必要がありますが、DynamoDB local にアクセスするための有効な AWS キーである必要はありません。

3. 次のコマンドラインのコマンドを実行します。

```
docker-compose up
```

DynamoDB Local を Apache Maven 依存関係として実行する

Amazon DynamoDB をアプリケーション内で依存関係として使用するには、これらのステップに従います。

DynamoDB を Apache Maven リポジトリとしてデプロイするには

1. Apache Maven をダウンロードし、インストールします。詳細については、「Apache Maven のダウンロード」および「Apache Maven のインストール」を参照してください。
2. DynamoDB の Maven リポジトリをアプリケーションのプロジェクトオブジェクトモデル (POM) ファイルに追加します。

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>2.4.0</version>
   </dependency>
</dependencies>
```

Spring Boot 3 や Spring Framework 6 で使用するテンプレートの例:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>SpringMavenDynamoDB</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
   <spring-boot.version>3.0.1</spring-boot.version>
   <maven.compiler.source>17</maven.compiler.source>
   <maven.compiler.target>17</maven.compiler.target>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

   <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>3.0.1</version>
   </parent>
```

```
<dependencies>
   <dependency>
       <groupId>com.amazonaws</groupId>
       <artifactId>DynamoDBLocal</artifactId>
       <version>2.4.0</version>
   </dependency>
   <!-- Spring Boot -->
   <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter</artifactId>
       <version>${spring-boot.version}</version>
   </dependency>
   <!-- Spring Web -->
   <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
       <version>${spring-boot.version}</version>
   </dependency>
   <!-- Spring Data JPA -->
   <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-data-jpa</artifactId>
       <version>${spring-boot.version}</version>
   </dependency>
   <!-- Other Spring dependencies -->
   <!-- Replace the version numbers with the desired version -->
   <dependency>
       <groupId>org.springframework</groupId>
       <artifactId>spring-context</artifactId>
       <version>6.0.0</version>
   </dependency>
   <dependency>
       <groupId>org.springframework</groupId>
       <artifactId>spring-core</artifactId>
       <version>6.0.0</version>
   </dependency>
   <!-- Add other Spring dependencies as needed -->
   <!-- Add any other dependencies your project requires -->
</dependencies>
</project>
```

> ⓘ Note
>
> Maven 中央リポジトリの URL を使用することもできます。

JAR ファイルのダウンロード、Docker イメージとしての実行、Maven 依存関係としての使用など、DynamoDB をローカルに設定して使用する複数の方法を紹介するサンプルプロジェクトの例については、「DynamoDB Local Sample Java Project」を参照してください。

DynamoDB local の使用に関する注意事項

エンドポイントを除き、ダウンロード可能なバージョンの Amazon DynamoDB で実行するアプリケーションは DynamoDB ウェブサービスでも動作します。ただし、DynamoDB をローカルで使用する場合は、以下に注意する必要があります。

- `-sharedDb` オプションを使用すると、DynamoDB は shared-local-instance.db という名前の単一のデータベースファイルを作成します。DynamoDB に接続するプログラムはいずれも、このファイルにアクセスします。このファイルを削除すると、保存されたすべてのデータを失うことになります。
- `-sharedDb` を省略する場合のデータベースファイルの名前は、myaccesskeyid_region.db で、アプリケーション設定に表示されるとおりの AWS アクセスキー ID と AWS リージョンが使用されます。このファイルを削除すると、保存されたすべてのデータを失うことになります。
- `-inMemory` オプションを使用した場合、DynamoDB はデータベースファイルの書き込みをまったく行いません。代わりに、すべてのデータがメモリに書き込まれ、DynamoDB を終了するときにデータは保存されません。
- `-inMemory` オプション使用する場合は、`-sharedDb` オプションも必要です。
- `-optimizeDbBeforeStartup` オプションを使用した場合は、`-dbPath` パラメータも指定し、DynamoDB がそのデータベースファイルを見つけられるようにする必要があります。
- DynamoDB 用 AWS SDK では、アプリケーション設定でアクセスキーバリューと AWS リージョンの値を指定する必要があります。`-sharedDb` または `-inMemory` オプションを使用している場合を除き、DynamoDB はこれらの値を使用してローカルデータベースファイルに名前を付けます。これらの値は、ローカルで動作する有効な AWS 値である必要はありません。ただし、使用しているエンドポイントを変更して、クラウドでもコードを実行できるように、有効な値を使用する方が便利な場合があります。
- DynamoDB local は `billingModeSummary.` に対して常に null を返します。

- DynamoDB local の AWS_ACCESS_KEY_ID には、文字 (A～Z、a～z) と数字 (0～9) のみを使用できます。
- DynamoDB local はポイントインタイムリカバリ (PITR) をサポートしていません。

トピック

コマンドラインオプション

次のコマンドラインオプションは、ダウンロード可能なバージョンの DynamoDB で使用できます。

- `-cors value` — JavaScript でクロスオリジンリソース共有 (CORS) のサポートを有効にします。特定のドメインのカンマ区切りの "許可" リストを指定する必要があります。`-cors` のデフォルト設定は、パブリックアクセスを許可するアスタリスク (*) です。
- `-dbPath value` — DynamoDB がそのデータベースファイルを書き込むディレクトリ。このオプションを指定しない場合、ファイルは現在のディレクトリに書き込まれます。`-dbPath` と `-inMemory` の両方を同時に指定することはできません。
- `-delayTransientStatuses` — DynamoDB によって特定のオペレーションに遅延が生じる原因になります。DynamoDB (ダウンロード可能バージョン) では、テーブルやインデックスの作成/更新/削除オペレーションなどの一部のタスクを瞬時に行うことができます。ただし、DynamoDB サービスでこれらのタスクを行うには、より時間がかかります。このパラメータを設定すると、コンピュータ上で実行されている DynamoDB で DynamoDB ウェブサービスの動作をより正確にシミュレートしやすくなります。(現在、このパラメータではステータスが CREATING または DELETING のグローバルセカンダリインデックスに対してのみ遅延が発生します。)
- `-help` — 使用方法の概要とオプションを出力します。
- `-inMemory` — DynamoDB は、データベースファイルを使用する代わりにメモリで実行されます。DynamoDB を停止すると、データは一切保存されません。`-dbPath` と `-inMemory` の両方を同時に指定することはできません。
- `-optimizeDbBeforeStartup` — コンピュータで DynamoDB を開始する前に、基になるデータベーステーブルを最適化します。このパラメータを使用するときは、`-dbPath` も指定する必要があります。

- `-port value` — DynamoDB がアプリケーションと通信するために使用するポート番号。このオプションを指定しない場合、デフォルトポートは 8000 になります。

> **Note**
>
> DynamoDB は、デフォルトではポート 8000 を使用します。ポート 8000 を使用できない場合、このコマンドにより例外がスローされます。`-port` オプションを使用すると、異なるポート番号を指定できます。DynamoDB ランタイムオプション (`-port` を含む) の詳細なリストを表示するには、次のコマンドを入力します。
>
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

- `-sharedDb` — `-sharedDb` を指定した場合、DynamoDB では、認証情報やリージョンごとに別のファイルを使用せずに、単一のデータベースファイルを使用します。
- `-disableTelemetry` — 指定した場合、DynamoDB local はテレメトリを送信しません。
- `-version` — DynamoDB Local のバージョンを出力します。

ローカルエンドポイントの設定

AWS SDK およびツールは、デフォルトで Amazon DynamoDB ウェブサービスのエンドポイントを使用します。ダウンロード可能なバージョンの DynamoDB を使用して SDK およびツールを使用するには、ローカルエンドポイントを指定する必要があります。

`http://localhost:8000`

AWS Command Line Interface

AWS Command Line Interface (AWS CLI) を使用して、ダウンロード可能な DynamoDB を操作できます。たとえば、「DynamoDB でのコード例用のテーブルの作成とデータのロード」のすべての手順を実行する場合に使用できます。

ローカルで実行中の DynamoDB にアクセスするには、`--endpoint-url` パラメータを使用します。AWS CLI を使用して、コンピュータにある DynamoDB のテーブルを一覧表示する例を次に示します。

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

> Note
>
> AWS CLI では、ダウンロード可能なバージョンの DynamoDB をデフォルトのエンドポイントとして使用することはできません。そのため、各 --endpoint-url コマンドで AWS CLI を指定する必要があります。

AWS SDK

エンドポイントを指定する方法は、使用しているプログラミング言語と AWS SDK によって異なります。以下のセクションでは、エンドポイントの指定方法について説明します。

- Java: AWS リージョンとエンドポイントの設定 (DynamoDB local は、AWS SDK for Java V1 および V2 をサポートしています)
- .NET: AWS リージョンとエンドポイントの設定

> Note
>
> 他のプログラミング言語の例については、「DynamoDB および AWS SDK の使用開始」を参照してください。

ダウンロード可能な DynamoDB と DynamoDB ウェブサービスの違い

ダウンロード可能バージョンの DynamoDB は、開発とテストのみを目的としています。これに対して、DynamoDB ウェブサービスは、スケーラビリティ、可用性、耐久性を特徴とする本稼働環境用に最適なマネージドサービスです。

ダウンロード可能なバージョンの DynamoDB とウェブサービスは、次の点で異なります。

- AWS リージョンと個別の AWS アカウントは、クライアントレベルでサポートされません。
- プロビジョンされたスループット設定は、ダウンロード可能な DynamoDB で無視されます。ただし、この設定は `CreateTable` オペレーションで必要です。`CreateTable` の場合、プロビジョニングされた読み込みおよび書き込みスループットに対して任意の数値を指定できます。ただし、この数値は使用されません。`UpdateTable` は 1 日に必要な回数呼び出すことができます。ただし、プロビジョニングされたスループット値に対する変更はいずれも無視されます。
- `Scan` 操作が連続的に行われます。並列スキャンはサポートされていません。`Segment` と `TotalSegments` パラメータ (`Scan` オペレーション) は無視されます。

- 読み込みおよび書き込みオペレーションの速度は、コンピュータの速度によってのみ制限を受けます。CreateTable、 UpdateTable、および DeleteTable オペレーションはすぐに実行されます。テーブルの状態は常に ACTIVE です。テーブルやグローバルセカンダリインデックスでプロビジョニングされたスループット設定のみを変更する UpdateTable オペレーションは、すぐに実行されます。UpdateTable オペレーションがグローバルセカンダリインデックスを作成または削除する場合、それらのインデックスは、通常の状態 (CREATING や DELETING など) に移行してから ACTIVE 状態になります。この間、テーブルは ACTIVE のままになります。
- 読み込みオペレーションには結果整合性があります。ただし、コンピュータで実行されている DynamoDB の速度が原因で、ほとんどの読み込みに強い整合性があるように見えます。
- 項目コレクションのメトリクスや項目コレクションのサイズは追跡されません。オペレーションレスポンスでは、項目コレクションのメトリクスの代わりに、null が返されます。
- DynamoDB では、結果セットごとに、返されるデータに 1 MB の制限があります。DynamoDB ウェブサービスとダウンロード可能バージョンのいずれにもこの制限が適用されます。ただし、インデックスのクエリを実行しているとき、DynamoDB サービスは、射影されたキーと属性のサイズのみを計算します。一方で、ダウンロード可能バージョンの DynamoDB は、項目全体のサイズを計算します。
- DynamoDB Streams を使用している場合、シャードが作成される速度が異なる可能性があります。DynamoDB ウェブサービスでは、シャードの作成動作は部分的にテーブルパーティションアクティビティの影響を受けます。ローカルで DynamoDB を実行している場合は、テーブルパーティションがありません。どちらの場合も、シャードはエフェメラルのため、アプリケーションがシャードの動作の影響を受けることはありません。
- ダウンロード可能な DynamoDB では、TransactionConflictExceptions はトランザクション API に対してスローされません。Java モック作成フレームワークを使用して DynamoDB ハンドラで TransactionConflictExceptions をシミュレートし、競合するトランザクションに対するアプリケーションの応答をテストすることをお勧めします。
- DynamoDB ウェブサービスでは、コンソールまたは AWS CLI のどちらからアクセスする場合でも、テーブル名の大文字と小文字は区別されます。Authors という名前のテーブルと authors という名前のテーブルが、別のテーブルとして両方存在できます。ダウンロード可能バージョンでは、テーブル名で大文字と小文字が区別されず、これら 2 つのテーブルを作成しようとすると、エラーが発生します。
- ダウンロード可能なバージョンの DynamoDB で、タグ付けはサポートされていません。
- ダウンロード可能なバージョンの DynamoDB では、ExecuteStatement の Limit パラメータは無視されます。

DynamoDB local のリリース履歴

次の表に、DynamoDB local のリリース別の重要な変更点を示します。

Version	変更	説明	日付
2.4.0	`ReturnValuesOnConditionCheckFailure` のサポート -埋め込みモード	• 複数ストリームでのオペレーションに関する `TrimmedDataAccessException` の埋め込みモードの修正 • 埋め込みモードの SDKv2 に関する例外変換の修正	2024 年 4 月 17 日
2.3.0	Jetty および JDK のアップグレード	• Jetty 12.0.2 へのアップグレード • JDK 17 へのアップグレード • ANTLR4 から 4.10.1 へのアップグレード	2024 年 3 月 14 日
2.2.0	テーブル削除保護と `ReturnValuesOnConditionCheckFailure` パラメータのサポートが追加されました。	• テーブル削除保護のサポートが追加されました。 • ReturnValuesOnConditionCheckFailure サポートが追加されました。 • -version フラグのサポートが追加されました。	2023 年 12 月 14 日

Version	変更	説明	日付
2.1.0	Maven プロジェクト用の SQLLite ネイティブライブラリのサポートとテレメトリの追加	• DynamoDB local へのテレメトリの追加 • Maven プロジェクト用の SQLLite ネイティブライブラリを動的にコピー • io.github.ganadist.sqlite4java ライブラリを Maven 依存関係から削除 • GoogleGuava を 32.1.1-jre にアップグレード	2023 年 10 月 23 日
2.0.0	javax から jakarta 名前空間への移行と JDK11 サポート	• javax から jakarta 名前空間への移行と JDK11 サポート • サーバー起動時の無効なアクセスとシークレットキーの処理を修正 • 依存関係を更新して Maven が特定した脆弱性を修正	2023 年 7 月 5 日

Version	変更	説明	日付
1.25.0	テーブル削除保護と `ReturnValuesOnConditionCheckFailure` パラメータのサポートが追加されました。	• テーブル削除保護のサポートが追加されました。 • ReturnValuesOnConditionCheckFailure サポートが追加されました。 • -version フラグのサポートが追加されました。	2023 年 12 月 18 日
1.24.0	Maven プロジェクト用の SQLLite ネイティブライブラリのサポートとテレメトリの追加	• DynamoDB local へのテレメトリの追加 • Maven プロジェクト用の SQLLite ネイティブライブラリを動的にコピー • io.github.ganadist.sqlite4java ライブラリを Maven 依存関係から削除 • GoogleGuava を 32.1.1-jre にアップグレード	2023 年 10 月 23 日

Version	変更	説明	日付
1.23.0	サーバー起動時の無効なアクセスとシークレットキーに対処	• サーバー起動時の無効なアクセスとシークレットキーの処理を修正 • 依存関係を更新して Maven が特定した脆弱性を修正	2023 年 6 月 28 日
1.22.0	PartiQL のリミット操作のサポート	• PartiQL 用の IN 句を最適化 • リミット操作のサポート • Maven プロジェクトの M1 サポート	2023 年 6 月 8 日
1.21.0	トランザクションあたり 100 アクションをサポート	• トランザクションあたりのアクション数を 25 から 100 に増加 • Docker イメージの Open JDK を 11 にアップグレード • BatchExecuteStatement の項目が重複している場合に発生する例外のパリティを修正	2023 年 1 月 26 日
1.20.0	M1 Mac のサポートを追加	• M1 Mac のサポートを追加 • Jetty の依存関係を 9.4.48.v20220622 にアップグレード	2022 年 9 月 12 日

Version	変更	説明	日付
1.19.0	PartiQL Parser をアップグレード	PartiQL Parser およびその他の関連ライブラリをアップグレード	2022 年 7 月 27 日
1.18.0	log4j-core と Jackson-core をアップグレード	log4j-core を 2.17.1 に、Jackson-core 2.10.x を 2.12.0 にアップグレード	2022 年 1 月 10 日
1.17.2	log4j-core をアップグレード	log4j-core の依存関係をバージョン 2.16 にアップグレード	2021 年 1 月 16 日
1.17.1	log4j-core をアップグレード	log4j-core の依存関係を更新し、リモートコード実行を防ぐためにゼロデイエクスプロイトにパッチを適用 - Log4Shell	2021 年 1 月 10 日
1.17.0	JavaScript Web Shell を非推奨化	• AWS SDK 依存関係を AWS SDK for Java 1.12.x に更新 • JavaScript Web Shell を非推奨化	2021 年 1 月 8 日

DynamoDB local のテレメトリ

AWS では、お客様とのやり取りから学んだことに基づいてサービスを開発して提供し、お客様からのフィードバックを使用して製品を練り直しています。テレメトリは、お客様のニーズをより深く理解して、問題を診断し、カスタマーエクスペリエンスを向上させる機能を提供するために役立つ追加情報です。

DynamoDB local は、一般的な使用状況メトリクス、システムおよび環境の情報、エラーなどのテレメトリを収集します。収集されるテレメトリのタイプの詳細については、「収集される情報のタイプ」を参照してください。

DynamoDB local は、ユーザー名や E メールアドレスなどの個人情報は収集しません。また、プロジェクトレベルの機密情報も抽出しません。

お客様は、テレメトリを有効にするかどうかを制御し、いつでも設定を変更できます。テレメトリが有効化されたままの場合、DynamoDB local はテレメトリデータをバックグラウンドで送信します。この際、お客様との追加のやり取りは不要です。

コマンドラインオプションを使用してテレメトリを無効にする

オプション -disableTelemetry によって DynamoDB local を起動するときに、コマンドラインオプションを使用してテレメトリを無効にすることができます。詳細については、「コマンドラインオプション」を参照してください。

単一セッションのテレメトリを無効にする

macOS および Linux オペレーティングシステムでは、単一セッションのテレメトリを無効にできます。現在のセッションのテレメトリを無効にするには、以下のコマンドを実行して環境変数 DDB_LOCAL_TELEMETRY を false に設定します。新しいターミナルまたはセッションに対して、このコマンドを繰り返します。

```
export DDB_LOCAL_TELEMETRY=0
```

すべてのセッションでのプロファイルのテレメトリの無効化

オペレーティングシステムで DynamoDB local を実行している場合は、以下のコマンドを実行してすべてのセッションのテレメトリを無効にします。

Linux でテレメトリを無効にするには

1. 以下を実行します:

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. 以下を実行します:

   ```
   source ~/.profile
   ```

macOS でテレメトリを無効にするには

1. 以下を実行します:

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. 以下を実行します:

   ```
   source ~/.profile
   ```

Windows でテレメトリを無効にするには

1. 以下を実行します:

   ```
   setx DDB_LOCAL_TELEMETRY 0
   ```

2. 以下を実行します:

   ```
   refreshenv
   ```

収集される情報のタイプ

- 使用状況の情報 — サーバーの起動/停止、呼び出された API やオペレーションなどの一般的なテレメトリ。
- システムおよび環境の情報 - Java のバージョン、オペレーティングシステム (Windows、Linux、または macOS)、DynamoDB local が実行する環境 (スタンドアロン JAR、Docker コンテナ、Maven 依存関係など)、および使用状況属性のハッシュ値。

詳細はこちら

DynamoDB local で収集されるテレメトリデータは、AWS データプライバシーポリシーに準拠します。詳細については、次を参照してください:

- AWS サービス利用規約
- データプライバシーに関するよくある質問

DynamoDB (ウェブサービス) の設定

Amazon DynamoDB ウェブサービスを使用するには:

1. AWS にサインアップする。
2. AWS アクセスキーの取得 (プログラムで DynamoDB にアクセスするために使用)。

> Note
>
> DynamoDB のみを使用して AWS Management Console とやり取りする場合、AWS アクセスキーは必要ないため、「コンソールを使用する場合」に進むことができます。

3. 認証情報の設定 (プログラムで DynamoDB にアクセスするために使用)。

AWS へのサインアップ

DynamoDB サービスを使用するには、AWS アカウントが必要です。アカウントをまだお持ちでない場合は、サインアップ時に画面の指示に従って作成してください。サインアップした AWS サービスの料金は、そのサービスを使用しない限り発生することはありません。

AWS にサインアップするには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   AWS アカウントにサインアップすると、AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべてのAWS のサービスとリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

プログラムによるアクセス権を付与する

プログラムまたは AWS Command Line Interface (AWS CLI) を介して DynamoDB にアクセスするには、プログラムによるアクセス権が必要です。DynamoDB コンソールのみを使用する場合、プログラムによるアクセス権は不要です。

AWS Management Console の外部で AWS を操作するには、プログラマチックアクセス権が必要です。プログラマチックアクセス権を付与する方法は、AWS にアクセスしているユーザーのタイプによって異なります。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラマチックリクエストに署名します。	使用するインターフェイス用の手引きに従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「AWS IAM Identity Center を使用するための AWS CLI の設定」を参照してください。 • AWS SDK、ツール、および AWS API については、AWS SDK とツールリファレンスガイドの「IAM Identity Center 認証」を参照してください。
IAM	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラムによるリクエストに署名します。	「IAM ユーザーガイド」の「AWS リソースでの一時的な認証情報の使用」の指示に従ってください。
IAM	(非推奨) 長期的な認証情報を使用して、AWS CLI、AWS SDK、AWS API へのプログラムによるリクエストに署名します。	使用するインターフェイス用の手順に従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「IAM

プログラマチックアクセス権を必要とするユーザー	目的	方法
		ユーザー認証情報を使用した認証」を参照してください。 - AWS SDK とツールについては、AWS SDK とツールリファレンスガイドの「長期認証情報を使用して認証する」を参照してください。 - AWS API については、IAM ユーザーガイドの「IAM ユーザーのアクセスキーの管理」を参照してください。

認証情報の設定

プログラムまたは AWS CLI を介して DynamoDB にアクセスする前に、アプリケーションで認可が有効になるように認証情報を設定する必要があります。

この方法には、いくつかあります。たとえば、認証情報ファイルを手動で作成して、お客様のアクセスキー ID とシークレットアクセスキーを格納します。また、このファイルは、AWS CLI コマンド `aws configure` を使用して自動的に作成することもできます。または、環境変数を使用できます。認証情報の設定に関する詳細については、プログラム固有の AWS SDK デベロッパーガイドを参照してください。

AWS CLI をインストールして設定する方法については、「AWS CLI の使用」を参照してください。

DynamoDB の他のサービスとの統合

DynamoDB AWS は他の多くのサービスと統合できます。詳細については、次を参照してください:

- DynamoDB を他の AWS サービスで使用する
- DynamoDB のAWS CloudFormation

- DynamoDB での AWS Backup の使用
- AWS Identity and Access Management (IAM)
- DynamoDB での AWS Identity and Access Management の使用

DynamoDB にアクセスする

Amazon DynamoDB には、AWS Management Console、AWS Command Line Interface (AWS CLI)、または DynamoDB API を使用してアクセスできます。

トピック

コンソールを使用する場合

Amazon DynamoDB の AWS Management Console、https://console.aws.amazon.com/dynamodb/home からアクセスできます。

コンソールを使用して、DynamoDB で以下のことを行うことができます。

- 最近のアラート、書き込み容量、サービス状態、DynamoDB の最新ニュースを DynamoDB ダッシュボードでモニタリングする。
- テーブルを作成、更新、削除する。容量計算ツールでは、入力した利用率に関する情報に基づいてリクエストするために容量ユニットの見積りを確認する。
- ストリームを管理する。
- テーブルに保存されている項目を表示し、項目の追加、更新、削除を行う。有効期限 (TTL) を管理し、データベースから自動的に削除できるようにテーブルの項目の有効期限を定義する。
- テーブルをクエリおよびスキャンする。
- アラームを設定および表示してテーブルの容量の利用率をモニタリングする。CloudWatch のリアルタイムグラフで、テーブルの主要なモニタリングメトリクスを表示する。
- テーブルのプロビジョニングした容量を変更する。
- テーブルのテーブルクラスを変更します。
- グローバルセカンダリインデックスを作成/削除する。
- DynamoDB Streams を AWS Lambda 関数に接続するためにトリガーを作成します。

- リソースを整理、識別しやすいように、リソースにタグを適用する。
- リザーブドキャパシティを購入する。

コンソールには、最初のテーブルの作成を求める初期画面が表示されます。テーブルを表示するには、コンソールの左側のナビゲーションペインから、[テーブル] を選択します。

各ナビゲーションタブ内のテーブルごとに使用可能なアクションの概要を以下に示します。

- 概要 – 項目数やメトリクスなど、テーブルの詳細を表示します。
- インデックス - グローバルおよびローカルのセカンダリインデックスを管理します。
- 監視 – アラーム、CloudWatch Contributor Insights、および CloudWatch メトリクスを表示します。
- グローバルテーブル – テーブルのレプリカを管理します。
- バックアップ – ポイントインタイムリカバリとオンデマンドバックアップを管理します。
- エクスポートとストリーム – テーブルを Amazon S3 にエクスポートし、DynamoDB Streams と Kinesis Data Streams を管理します。
- 追加設定 – 読み取り/書き込み容量、有効期限 (TTL) の設定、暗号化、タグを管理します。

AWS CLI の使用

AWS Command Line Interface (AWS CLI) を使用すると、複数の AWS のサービスをコマンドラインから制御したり、スクリプトで自動化したりできます。テーブルの作成など、その場限りのオペレーションに AWS CLI を使用できます。また、ユーティリティスクリプト内に Amazon DynamoDB オペレーションを埋め込むときにも使用できます。

DynamoDB で AWS CLI を使用するには、事前にアクセスキー ID とシークレットアクセスキーを取得する必要があります。詳細については、「プログラムによるアクセス権を付与する」を参照してください。

AWS CLI で DynamoDB 向けに使用できるすべてのコマンドの完全な一覧については、「AWS CLI コマンドリファレンス」を参照してください。

トピック

- AWS CLI のダウンロードと設定
- DynamoDB での AWS CLI の使用
- DynamoDB Local での AWS CLI の使用

AWS CLI のダウンロードと設定

AWS CLI は、http://aws.amazon.com/cli で入手できます。Windows、macOS、または Linux 上で実行できます。AWS CLI をダウンロードしたら、以下の手順に従って、インストールと設定を行います。

1. AWS Command Line Interface ユーザーガイドに移動します。
2. AWS CLI のインストール、AWS CLI の設定の手順に従います。

DynamoDB での AWS CLI の使用

コマンドラインの形式は、DynamoDB オペレーション名と、それに続くそのオペレーションのパラメータで構成されます。AWS CLI では、パラメータ値の短縮構文および JSON をサポートしています。

例えば、次のコマンドでは、Music という名前のテーブルを作成します。パーティションキーは Artist で、ソートキーは SongTitle です。(読みやすくするために、このセクションの長いコマンドは、複数の行に分かれています)。

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1 \
    --table-class STANDARD
```

次のコマンドでは、新しい項目をテーブルに追加します。この例では、短縮構文と JSON を組み合わせて使用しています。

```
aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}}' \
    --return-consumed-capacity TOTAL
```

```
aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Acme Band"},
        "SongTitle": {"S": "Happy Day"},
        "AlbumTitle": {"S": "Songs About Life"} }' \
    --return-consumed-capacity TOTAL
```

コマンドラインで、有効な JSON を作成するのは難しい場合があります。ただし、AWS CLI で JSON ファイルを読み込むことができます。例えば、key-conditions.json という名前のファイルに格納されている次の JSON コードスニペットがあるとします。

```
{
    "Artist": {
        "AttributeValueList": [
            {
                "S": "No One You Know"
            }
        ],
        "ComparisonOperator": "EQ"
    },
    "SongTitle": {
        "AttributeValueList": [
            {
                "S": "Call Me Today"
            }
        ],
        "ComparisonOperator": "EQ"
    }
}
```

次のように、`Query` を使用して、AWS CLI リクエストを発行できます。この例では、key-conditions.json ファイルの内容は、`--key-conditions` パラメータに使用されます。

```
aws dynamodb query --table-name Music --key-conditions file://key-conditions.json
```

DynamoDB Local での AWS CLI の使用

AWS CLI を使用して、コンピュータで実行されている DynamoDB Local (ダウンロード可能なバージョン) を操作することもできます。これを有効にするには、各コマンドに次のパラメータを追加します。

```
--endpoint-url http://localhost:8000
```

次の例では、AWS CLI を使用して、ローカルデータベースのテーブルを一覧表示します。

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

DynamoDB がデフォルト (8000) 以外のポート番号を使用している場合は、それに応じて `--endpoint-url` 値を変更する必要があります。

> **Note**
>
> AWS CLI では、DynamoDB Local (ダウンロード可能なバージョン) をデフォルトのエンドポイントとして使用することはできません。そのため、各コマンドで `--endpoint-url` を指定する必要があります。

API の使用

AWS Management Console と AWS Command Line Interface を使用して、Amazon DynamoDB とインタラクティブに作業できます。ただし、DynamoDB を最大限に活用するためには、AWS SDK を使用してアプリケーションコードを記述できます。

AWS SDK は、DynamoDB の広範なサポート (Java、ブラウザの JavaScript、.NET、Node.js、PHP、Python、Ruby、C++、Go、Android、iOS) を提供します。このような言語を使用してすばやく開始する方法については、「DynamoDB および AWS SDK の使用開始」を参照してください。

DynamoDB を使用して AWS を使用するには、AWS アクセスキー ID とシークレットアクセスキーを取得する必要があります。詳細については、「 DynamoDB (ウェブサービス) の設定 」を参照してください。

AWS SDK を使用した DynamoDB アプリケーションのプログラミングに関する概要については、「DynamoDB と AWS SDK を使用したプログラミング」を参照してください。

DynamoDB 用の NoSQL Workbench の使用

また、DynamoDB には、DynamoDB 用の NoSQL Workbench をダウンロードして使用することによってアクセスできます。

Amazon DynamoDB 用の NoSQL Workbench は、最新のデータベース開発および運用向けのクロスプラットフォームのクライアント側 GUI アプリケーションです。Windows、macOS、Linux で使用できます。NoSQL Workbench は、DynamoDB テーブルの設計、作成、クエリ、管理に役立つデータモデリング、データ可視化、クエリ開発といった特徴を提供する視覚的開発ツールです。NoSQL Workbench に、インストールプロセスのオプションとして DynamoDB local が含まれるようになったため、DynamoDB local でデータを簡単にモデル化できます。DynamoDB local とその要件の詳細については、「DynamoDB local (ダウンロード可能バージョン) のセットアップ」を参照してください。

> Note
>
> NoSQL Workbench for DynamoDB は、現在、2 要素認証 (2FA) で構成されている AWS ログインをサポートしていません。

データモデリング

DynamoDB 用の NoSQL Workbench を使用すると、アプリケーションのデータアクセスパターンを満たす既存のデータモデルから新しいデータモデルを構築したり、既存のデータモデルに基づいてモデルを設計したりできます。プロセスの最後に、設計されたデータモデルをインポートおよびエクスポートすることもできます。詳細については、「NoSQL Workbench を使用したデータモデルの構築」を参照してください。

データの可視化

データモデルビジュアライザーは、コードを記述せずにクエリをマップし、アプリケーションのアクセスパターン (ファセット) を可視化できるキャンバスを提供します。すべてのファセットは、DynamoDB の異なるアクセスパターンに対応しています。データ型で使用するサンプルデータを自動生成できます。詳細については、「データアクセスパターンの可視化」を参照してください。

オペレーション構築

NoSQL Workbench は、クエリを開発およびテストするための豊富なグラフィカルユーザーインターフェイスを提供します。オペレーションビルダーを使用して、データセットを表示、探索、およびクエリできます。構造化オペレーションビルダーを使用して、データプレーンオペレーションを構築および実行することもできます。プロジェクションと条件式をサポートし、複数の言語でサンプルコードを生成できます。詳細については、「NoSQL Workbench を使用したデータセットの探索とオペレーションの構築」を参照してください。

IP アドレスの範囲

Amazon Web Services (AWS) は、その現在の IP アドレス範囲を JSON 形式で公開します。現在の範囲を参照するには、ip-ranges.json ファイルをダウンロードします。詳細については、AWS 全般のリファレンス の AWS IP アドレスの範囲をご参照ください。

DynamoDB テーブルおよびインデックスへのアクセスに使用できる IP アドレス範囲を見つけるには、ip-ranges.json で文字列 `"service": "DYNAMODB"` を検索します。

> ⓘ Note
>
> この IP アドレス範囲は、DynamoDB Streams および DynamoDB Accelerator (DAX) には適用されません。

DynamoDB の使用開始

このセクションの実践的チュートリアルを使用することで、Amazon DynamoDB の使用を開始し、その詳細を学ぶことができます。

トピック

DynamoDB の基本概念

開始する前に、Amazon DynamoDB の基本概念を理解する必要があります。詳細については、「Amazon DynamoDB のコアコンポーネント」を参照してください。

次に、「前提条件」に進んで、DynamoDB の設定について学習してください。

前提条件 - 入門ガイドチュートリアル

Amazon DynamoDB チュートリアルを開始する前に、DynamoDB のセットアップのステップに従います。次に、「ステップ 1: テーブルを作成します」に進んでください。

> Note
>
> - AWS Management Console のみを使用して DynamoDB とやり取りする場合、AWS アクセスキーは必要ありません。「AWS にサインアップする」のステップを完了し、「ステップ 1: テーブルを作成します」に進んでください。
> - 無料利用枠アカウントにサインアップしない場合は、DynamoDB local (ダウンロード可能バージョン) をセットアップできます。次に、「ステップ 1: テーブルを作成します」に進んでください。
> - Linux と Windows のターミナルで CLI コマンドを操作する場合、違いがあります。次のガイドでは、Linux ターミナル (macOSを含む) 用にフォーマットされたコマンドと、Windows CMD 用にフォーマットされたコマンドを紹介します。使用しているターミナルアプリケーションに最適なコマンドを選択してください。

ステップ 1: テーブルを作成します

このステップでは、Amazon DynamoDB で `Music` テーブルを作成していきます。このテーブルには次の詳細があります。

- パーティションキー – `Artist`
- ソートキー – `SongTitle`

テーブルオペレーションの詳細については、「DynamoDB でのテーブルとデータの操作」を参照してください。

> Note
>
> 開始する前に、必ず「前提条件 - 入門ガイドチュートリアル」のステップに従ってください。

AWS Management Console

DynamoDB コンソールを使用して新しい `Music` テーブルを作成するには

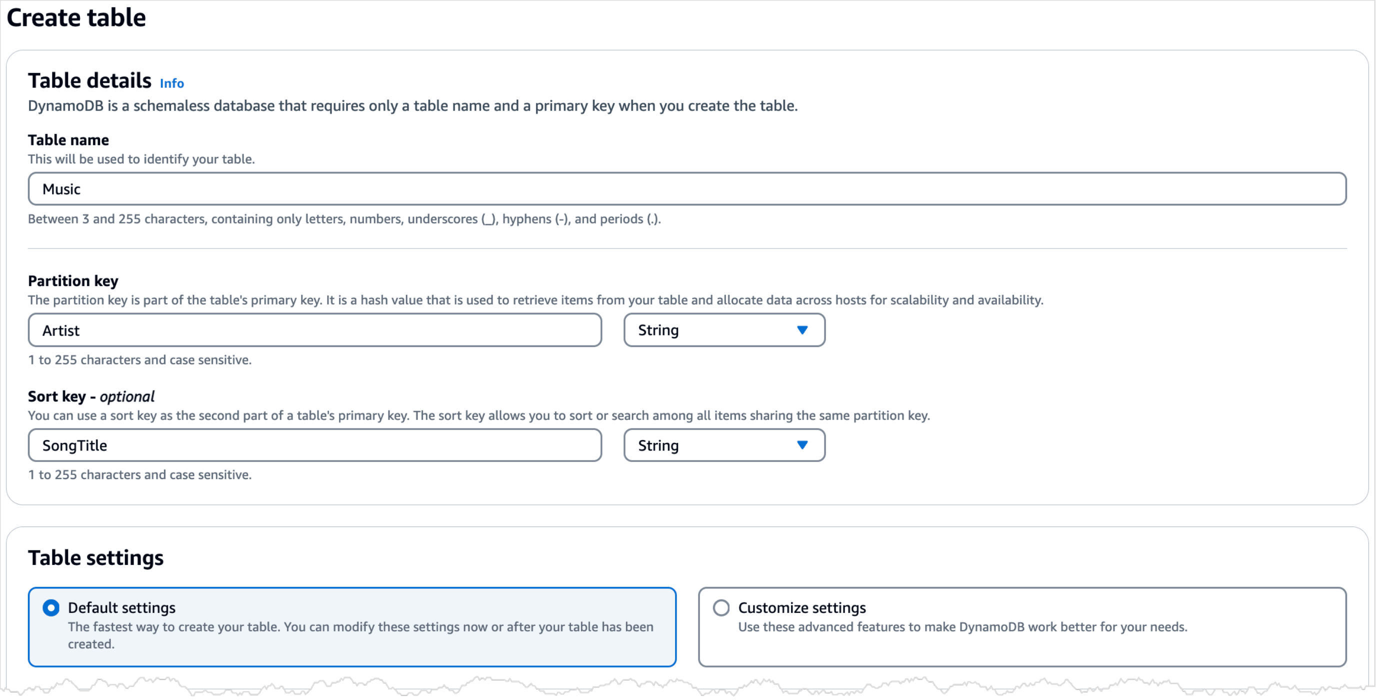
1. AWS Management Consoleにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。

2. 左のナビゲーションペインで、[テーブル] を選択します。
3. [Create table (テーブルの作成)] を選択します。
4. [テーブルの詳細] を次のように入力します。

   a. [テーブル名] に「**`Music`**」と入力します。

   b. [パーティションキー] に「**`Artist`**」と入力します。

   c. [ソートキー] に、「**`SongTitle`**」と入力します。
5. [テーブル設定] は、[デフォルト設定] を選択したままにします。
6. [テーブルを作成] を選択してテーブルを作成します。



Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

Music

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

Artist | String ▼

1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

SongTitle | String ▼

1 to 255 characters and case sensitive.

Table settings

● Default settings

The fastest way to create your table. You can modify these settings now or after your table has been created.

○ Customize settings

Use these advanced features to make DynamoDB work better for your needs.

7. テーブルのステータスが [ACTIVE] になったら、次の手順を実行して、テーブルの [DynamoDB のポイントインタイムリカバリ] を有効にすることをお勧めします。

   a. テーブル名を選択してテーブルを開きます。

   b. [バックアップ] を選択します。

   c. [ポイントインタイムリカバリ (PITR)] セクションで、[編集] を選択します。

   d. [ポイントインタイムリカバリ設定の編集] ページで、[ポイントタイムリカバリの有効化] を選択します。

   e. [Save changes] (変更の保存) をクリックします。

AWS CLI

次の AWS CLI の例では、Music を使用して新しい create-table テーブルを作成します。

Linux

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --table-class STANDARD
```

Windows CMD

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema ^
        AttributeName=Artist,KeyType=HASH ^
        AttributeName=SongTitle,KeyType=RANGE ^
    --provisioned-throughput ^
        ReadCapacityUnits=5,WriteCapacityUnits=5 ^
    --table-class STANDARD
```

create-table を使用すると、次のサンプル結果が返されます。

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-03-29T12:11:43.379000-04:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:111122223333:table/Music",
        "TableId": "60abf404-1839-4917-a89b-a8b0ab2a1b87",
        "TableClassSummary": {
            "TableClass": "STANDARD"
        }
    }
}
}
```

`TableStatus` フィールドの値は `CREATING` に設定されていることに注意してください。

DynamoDB が `Music` テーブルの作成を終了したことを確認するには、`describe-table` コマンドを使用します。

Linux

```
aws dynamodb describe-table --table-name Music | grep TableStatus
```

Windows CMD

```
aws dynamodb describe-table --table-name Music | findstr TableStatus
```

このコマンドは、次の結果を返します。DynamoDB がテーブルの作成を終了すると、TableStatus フィールドの値が ACTIVE に設定されます。

```
"TableStatus": "ACTIVE",
```

テーブルのステータスが ACTIVE になったら、以下のコマンドを実行して、テーブルで DynamoDB のポイントインタイムリカバリ を有効にするのがベストプラクティスと考えられています。

Linux

```
aws dynamodb update-continuous-backups \
    --table-name Music \
    --point-in-time-recovery-specification \
        PointInTimeRecoveryEnabled=true
```

Windows CMD

```
aws dynamodb update-continuous-backups --table-name Music --point-in-time-recovery-specification PointInTimeRecoveryEnabled=true
```

このコマンドは、次の結果を返します。

```
{
    "ContinuousBackupsDescription": {
        "ContinuousBackupsStatus": "ENABLED",
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": "2023-03-29T12:18:19-04:00",
            "LatestRestorableDateTime": "2023-03-29T12:18:19-04:00"
        }
    }
}
```

> Note
>
> ポイントインタイムリカバリによる連続バックアップを有効にすると、コストがかかります。料金の詳細については、「Amazon DynamoDB の料金表」を参照してください。

新しいテーブルの作成後、「ステップ 2: コンソールまたは AWS CLI を使用して、テーブルにデータを書き込みます」に進みます。

ステップ 2: コンソールまたは AWS CLI を使用して、テーブルにデータを書き込みます

このステップでは、ステップ 1: テーブルを作成します で作成した `Music` テーブルに複数の項目を挿入します。

書き込みオペレーションの詳細については、「項目を書き込みます」を参照してください。

AWS Management Console

以下の手順に従い、DynamoDB コンソールを使用して `Music` テーブルにデータを書き込みます。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. 左のナビゲーションペインで、[テーブル] を選択します。
3. [テーブル] ページで、[Music] テーブルを選択します。
4. [テーブルアイテムの探索] を選択します。
5. [返された項目] セクションで、[項目を作成] を選択します。
6. [項目を作成] ページで、次の操作を行ってテーブルに項目を追加します。
   a. [新規属性を追加] を選択し、[数値] を選択します。
   b. [属性名] に、「**`Awards`**」と入力します。
   c. このプロセスを繰り返して、[String (文字列型)] の **`AlbumTitle`** を作成します。
   d. 項目に以下の値を入力します。
      i. [Artist] に「**`No One You Know`**」と入力します。
      ii. [SongTitle] に「**`Call Me Today`**」と入力します。

iii. [AlbumTitle] に「**Somewhat Famous**」と入力します。

iv. [Awards] に「**1**」と入力します。

7. [項目を作成] を選択します。
8. このプロセスを繰り返して、次の値を持つ別の項目を作成します。

   a. [Artist] に「**Acme Band**」と入力します。

   b. [SongTitle] に「**Happy Day**」と入力します。

   c. [AlbumTitle] に「**Songs About Life**」と入力します。

   d. [Awards] に「**10**」と入力します。
9. これをもう一度実行して、前のステップと同じ Artist を含むが、その他の属性の値が異なる項目を作成します。

   a. [Artist] に「**Acme Band**」と入力します。

   b. [SongTitle] に「**PartiQL Rocks**」と入力します。

   c. [AlbumTitle] に「**Another Album Title**」と入力します。

   d. [Awards] に「**8**」と入力します。

AWS CLI

次の AWS CLI の例では、`Music` テーブルで複数の新しい項目を作成します。この操作は、DynamoDB API または PartiQL (DynamoDB の SQL 互換クエリ言語) を介して行うことができます。

DynamoDB API

Linux

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Howdy"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "2"}}'
```

```
aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "PartiQL Rocks"},
 "AlbumTitle": {"S": "Another Album Title"}, "Awards": {"N": "8"}}'
```

Windows CMD

```
aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Call
 Me Today\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\":
 \"1\"}}"

aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Howdy
\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\": \"2\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day\"},
 \"AlbumTitle\": {\"S\": \"Songs About Life\"}, \"Awards\": {\"N\": \"10\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"PartiQL Rocks
\"}, \"AlbumTitle\": {\"S\": \"Another Album Title\"}, \"Awards\": {\"N\": \"8\"}}"
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Call Me Today',
 'AlbumTitle':'Somewhat Famous', 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Howdy',
 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs
 About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'PartiQL Rocks',
 'AlbumTitle':'Another Album Title', 'Awards':'8'}"
```

Windows CMD

```
aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Call Me Today', 'AlbumTitle':'Somewhat Famous',
 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Howdy', 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'PartiQL Rocks', 'AlbumTitle':'Another Album Title',
 'Awards':'8'}"
```

PartiQL を使用したデータの書き込みについては、「PartiQL 挿入ステートメント」を参照してください。

DynamoDB でサポートされるデータ型の詳細については、「データ型」を参照してください。

JSON で DynamoDB のデータ型を表す方法については、「属性値」を参照してください。

テーブルにデータを書き込んだら、「ステップ 3: テーブルからデータを読み込みます」に進みます。

ステップ 3: テーブルからデータを読み込みます

このステップでは、「ステップ 2: コンソールまたは AWS CLI を使用して、テーブルにデータを書き込みます」で作成した項目の 1 つを読み込みます。DynamoDB コンソールまたは AWS CLI を使用して、`Artist` および `SongTitle` を指定しながら `Music` テーブルの項目を読み込むことができます。

DynamoDB の読み込みオペレーションの詳細については、「項目の読み込み」を参照してください。

AWS Management Console

以下の手順に従い DynamoDB コンソールを使用して、`Music` テーブルからデータを読み込みます。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. 左のナビゲーションペインで、[テーブル] を選択します。
3. [テーブル] ページで、[Music] テーブルを選択します。
4. [テーブルアイテムの探索] を選択します。
5. [返された項目] セクションで、テーブルに保存されている項目のリストを、`Artist` と `SongTitle` でソートして表示します。リストの最初の項目は、[Artist] として [Acme Band]、[SongTitle] として [PartiQL Rocks] になります。

AWS CLI

次の AWS CLI の例では、`Music` から項目を読み込みます。この操作は、DynamoDB API または PartiQL (DynamoDB の SQL 互換クエリ言語) を介して行うことができます。

DynamoDB API

> **Note**
>
> DynamoDB では、デフォルトで整合性のある読み込みを行います。以下では、`consistent-read` パラメータを使用して強力な整合性のある読み込みを示します。

Linux

```
aws dynamodb get-item --consistent-read \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'
```

Windows CMD

```
aws dynamodb get-item --consistent-read ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}"
```

`get-item` を使用すると、次のサンプル結果が返されます。

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
WHERE Artist='Acme Band' AND SongTitle='Happy Day'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band' AND SongTitle='Happy Day'"
```

PartiQL `Select` ステートメントを使用すると、次のサンプル結果が返されます。

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Songs About Life"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

PartiQL を使用したデータの読み込みについては、「PartiQL 選択ステートメント」を参照してください。

テーブルのデータを更新するには、「ステップ 4: テーブルのデータを更新します」に進みます。

ステップ 4: テーブルのデータを更新します

このステップでは、「ステップ 2: コンソールまたは AWS CLI を使用して、テーブルにデータを書き込みます」で作成した項目を更新できます。DynamoDB コンソールまたは AWS CLI を使用して、`Artist`、`SongTitle` および更新された `AlbumTitle` を指定して `Music` テーブルの項目の `AlbumTitle` を更新できます。

書き込みオペレーションの詳細については、「項目を書き込みます」を参照してください。

AWS Management Console

DynamoDB コンソールを使用して、`Music` テーブルのデータを更新できます。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. 左のナビゲーションペインで、[テーブル] を選択します。
3. テーブルのリストから [Music] テーブルを選択します。
4. [テーブルアイテムの探索] を選択します。
5. [返された項目] の項目行 ([Acme Band] という [Artist] と [Happy Day] という [SongTitle]) で、次の操作を行います。

    a. [Songs About Life] という名前の [AlbumTitle] にカーソルを置きます。

    b. [編集] アイコンを選択します。

    c. [文字列を編集] ポップアップウィンドウに、「**`Songs of Twilight`**」と入力します。

    d. [Save] を選択します。

> Tip
>
> または、項目を更新するには、[返された項目] セクションで次の操作を行います。
>
> 1. [Artist] として [Acme Band] および [SongTitle] として [Happy Day] を示す項目行を選択します。
> 2. [アクション] ドロップダウンリストで、[項目の編集] を選択します。
> 3. [AlbumTitle] に「**`Songs of Twilight`**」と入力します。
> 4. [保存して閉じる] を選択します。

AWS CLI

次の AWS CLI の例では、`Music` テーブルの項目を更新します。この操作は、DynamoDB API または [PartiQL](#) (DynamoDB の SQL 互換クエリ言語) を介して行うことができます。

DynamoDB API

Linux

```
aws dynamodb update-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' \
    --update-expression "SET AlbumTitle = :newval" \
    --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' \
    --return-values ALL_NEW
```

Windows CMD

```
aws dynamodb update-item ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}" ^
    --update-expression "SET AlbumTitle = :newval" ^
    --expression-attribute-values "{\":newval\":{\"S\":\"Updated Album Title\"}}" ^
    --return-values ALL_NEW
```

`update-item` を使用すると、`return-values ALL_NEW` が指定されたために次のサンプル結果が返されます。

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Updated Album Title"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "UPDATE Music  \
    SET AlbumTitle='Updated Album Title'  \
    WHERE Artist='Acme Band' AND SongTitle='Happy Day' \
    RETURNING ALL NEW *"
```

Windows CMD

```
aws dynamodb execute-statement --statement "UPDATE Music SET AlbumTitle='Updated
 Album Title' WHERE Artist='Acme Band' AND SongTitle='Happy Day' RETURNING ALL NEW
 *"
```

`Update` ステートメントを使用すると、`RETURNING ALL NEW *` が指定されたために次のサンプル結果が返されます。

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

PartiQL を使用したデータの更新については、「PartiQL 更新ステートメント」を参照してください。

`Music` テーブルのデータをクエリするには、「ステップ 5: テーブルのデータをクエリします」に進みます。

ステップ 5: テーブルのデータをクエリします

このステップでは、Music を指定して the section called “ステップ 2: データを書き込みます” の Artist テーブルに書き込んだデータをクエリします。これにより、パーティションキー Artist に関連付けられているすべての曲が表示されます。

クエリオペレーションの詳細については、「DynamoDB のクエリオペレーション」を参照してください。

AWS Management Console

DynamoDB コンソールを使用して Music テーブルのデータをクエリするには、次のステップに従います。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. 左のナビゲーションペインで、[テーブル] を選択します。
3. テーブルのリストから [Music] テーブルを選択します。
4. [テーブルアイテムの探索] を選択します。
5. [項目のスキャンまたはクエリ] で、[クエリ] が選択されていることを確認します。
6. [パーティションキー] に「**Acme Band**」と入力し、[実行] を選択します。

AWS CLI

次の AWS CLI の例では、Music テーブルの項目にクエリを実行します。この操作は、DynamoDB API または PartiQL (DynamoDB の SQL 互換クエリ言語) を介して行うことができます。

DynamoDB API

query を使用してパーティションキーを指定することで、DynamoDB API を介して項目にクエリを実行できます。

Linux

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

Windows CMD

```
aws dynamodb query ^
    --table-name Music ^
    --key-condition-expression "Artist = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Acme Band\"}}"
```

query を使用すると、この特定の Artist に関連付けられているすべての曲が返されます。

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "N": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "N": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

PartiQL for DynamoDB

Select ステートメントを使用してパーティションキーを指定することで、PartiQL を介して項目にクエリを実行できます。

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band'"
```

このように Select ステートメントを使用すると、この特定の Artist に関連付けられているすべての曲が返されます。

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "S": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

PartiQL を使用したデータへのクエリの実行については、「PartiQL 選択ステートメント」を参照してください。

テーブルのグローバルセカンダリインデックスを作成するには、「ステップ 6: グローバルセカンダリインデックスを作成します」に進みます。

ステップ 6: グローバルセカンダリインデックスを作成します

このステップでは、「Music」で作成した ステップ 1: テーブルを作成します テーブルのグローバルセカンダリインデックスを作成します。

グローバルセカンダリインデックスの詳細については、「DynamoDB のグローバルセカンダリインデックスの使用」を参照してください。

AWS Management Console

Amazon DynamoDB コンソールを使用して、Music テーブルのグローバルセカンダリインデックス AlbumTitle-index を作成するには:

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. 左のナビゲーションペインで、[テーブル] を選択します。
3. テーブルのリストから [Music] テーブルを選択します。
4. [Music] テーブルの [インデックス] タブを選択します。
5. [インデックスの作成] を選択します。
6. [グローバルセカンダリインデックスの作成] ページで、次の操作を行います。
   a. [パーティションキー] に「**AlbumTitle**」と入力します。
   b. [インデックス名] に「**AlbumTitle-index**」と入力します。

c. ページの他の設定はデフォルトのままにし、[インデックスの作成] を選択します。

AWS CLI

次の AWS CLI の例では、`AlbumTitle-index` を使用して `Music` テーブルのグローバルセカンダリインデックス `update-table` を作成します。

Linux

```
aws dynamodb update-table \
    --table-name Music \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates \
        "[{\"Create\":{\"IndexName\": \"AlbumTitle-index\",\"KeySchema\":
[{\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"HASH\"}], \
        \"ProvisionedThroughput\": {\"ReadCapacityUnits\": 10, \"WriteCapacityUnits\":
 5      },\"Projection\":{\"ProjectionType\":\"ALL\"}}}]"
```

Windows CMD

```
aws dynamodb update-table ^
    --table-name Music ^
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S ^
    --global-secondary-index-updates "[{\"Create\":{\"IndexName\": \"AlbumTitle-
index\",\"KeySchema\":[{\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"HASH\"}],
 \"ProvisionedThroughput\": {\"ReadCapacityUnits\": 10, \"WriteCapacityUnits\": 5},
\"Projection\":{\"ProjectionType\":\"ALL\"}}}]"
```

`update-table` を使用すると、次のサンプル結果が返されます。

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-west-2:111122223333:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexSizeBytes": 0,
                "IndexName": "AlbumTitle-index",
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "WriteCapacityUnits": 5,
                    "ReadCapacityUnits": 10
                },
                "IndexStatus": "CREATING",
                "Backfilling": false,
                "KeySchema": [
                    {
                        "KeyType": "HASH",
                        "AttributeName": "AlbumTitle"
                    }
                ],
                "IndexArn": "arn:aws:dynamodb:us-west-2:111122223333:table/Music/index/
AlbumTitle-index",
                "ItemCount": 0
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "UPDATING",
        "TableId": "a04b7240-0a46-435b-a231-b54091ab1017",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1558028402.69
    }
}
```

IndexStatus フィールドの値は CREATING に設定されていることに注意してください。

DynamoDB がグローバルセカンダリインデックス AlbumTitle-index の作成を終了したことを確認するには、describe-table コマンドを使用します。

Linux

```
aws dynamodb describe-table --table-name Music | grep IndexStatus
```

Windows CMD

```
aws dynamodb describe-table --table-name Music | findstr IndexStatus
```

このコマンドは、次の結果を返します。返された IndexStatus フィールドの値が ACTIVE に設定されている場合、インデックスは使用できる状態です。

```
"IndexStatus": "ACTIVE",
```

次に、グローバルセカンダリインデックスにクエリできます。詳細については、「ステップ 7: グローバルセカンダリインデックスをクエリします」を参照してください。

ステップ 7: グローバルセカンダリインデックスをクエリします

このステップでは、Amazon DynamoDB コンソールまたは AWS CLI を使用して、Music テーブルのグローバルセカンダリインデックスをクエリします。

グローバルセカンダリインデックスの詳細については、「DynamoDB のグローバルセカンダリインデックスの使用」を参照してください。

AWS Management Console

以下のステップに従って DynamoDB コンソールを使用し、グローバルセカンダリインデックス `AlbumTitle-index` を通じてデータをクエリします。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. 左のナビゲーションペインで、[テーブル] を選択します。
3. テーブルのリストから [Music] テーブルを選択します。
4. [テーブルアイテムの探索] を選択します。
5. [項目のスキャンまたはクエリ] で、[クエリ] はデフォルトのままにします。
6. [テーブルまたはインデックスを選択] で、[AlbumTitle-index] を選択します。
7. [AlbumTitle] に「**`Somewhat Famous`**」と入力し、[実行] を選択します。

AWS CLI

次の AWS CLI の例では、`AlbumTitle-index` テーブルのグローバルセカンダリインデックス `Music` をクエリします。この操作は、DynamoDB API または PartiQL (DynamoDB の SQL 互換クエリ言語) を介して行うことができます。

DynamoDB API

`query` を使用してインデックス名を指定することで、DynamoDB API を介してグローバルセカンダリインデックスにクエリを実行できます。

Linux

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitle-index \
    --key-condition-expression "AlbumTitle = :name" \
    --expression-attribute-values  '{":name":{"S":"Somewhat Famous"}}'
```

Windows CMD

```
aws dynamodb query ^
    --table-name Music ^
    --index-name AlbumTitle-index ^
    --key-condition-expression "AlbumTitle = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Somewhat Famous\"}}"
```

query を使用すると、次のサンプル結果が返されます。

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Awards": {
                "S": "1"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Awards": {
                "N": "2"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Howdy"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

PartiQL for DynamoDB

Select ステートメントを使用してインデックス名を指定することで、PartiQL を介してグローバルセカンダリインデックスにクエリを実行できます。

Note

CLI を介してこれを行うため、Music と AlbumTitle-index を囲む二重引用符をエスケープする必要があります。

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM \"Music\".\"AlbumTitle-
index\"  \
                                            WHERE AlbumTitle='Somewhat Famous'"
```

Windows CMD

```
aws dynamodb execute-statement --statement "SELECT * FROM \"Music\".\"AlbumTitle-
index\" WHERE AlbumTitle='Somewhat Famous'"
```

このように Select ステートメントを使用すると、次のサンプル結果が返されます。

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Awards": {
                "S": "1"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Awards": {
                "S": "2"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Howdy"
            }
        }
    ]
}
```

PartiQL を使用したデータへのクエリの実行については、「PartiQL 選択ステートメント」を参照してください。

ステップ 8: (オプション) リソースをクリーンアップする

チュートリアルで作成した Amazon DynamoDB テーブルが不要になった場合は、削除します。このステップにより、使用していないリソースに対して課金されることがなくなります。DynamoDB コンソールまたは AWS CLI を使用して、ステップ 1: テーブルを作成します で作成した `Music` テーブルを削除することができます。

DynamoDB のテーブルオペレーションの詳細については、「DynamoDB でのテーブルとデータの操作」を参照してください。

AWS Management Console

コンソールを使用して `Music` テーブルを削除するには:

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. 左のナビゲーションペインで、[テーブル] を選択します。
3. テーブルリストで、[Music] テーブルの横のチェックボックスをオンにします。
4. [削除] を選択します。

AWS CLI

以下の AWS CLI の例では、`Music` を使用して `delete-table` テーブルを削除します。

```
aws dynamodb delete-table --table-name Music
```

DynamoDB の使用開始: 次のステップ

Amazon DynamoDB の使用に関する詳細については、以下のトピックを参照してください。

- DynamoDB でのテーブルとデータの操作
- 項目と属性の操作
- DynamoDB のクエリオペレーション
- DynamoDB のグローバルセカンダリインデックスの使用
- トランザクションでの使用
- DynamoDB Accelerator (DAX) とインメモリアクセラレーション
- DynamoDB および AWS SDK の使用開始
- DynamoDB と AWS SDK を使用したプログラミング

DynamoDB および AWS SDK の使用開始

このセクションの実践的チュートリアルを使用することで、Amazon DynamoDB と AWS SDK の使用を開始します。DynamoDB のダウンロード可能バージョン、または DynamoDB ウェブサービスのいずれかでコード例を実行できます。

トピック

DynamoDB テーブルを作成する

AWS Management Console、AWS CLI、または AWS SDK を使用してテーブルを作成できます。テーブルの詳細については、「Amazon DynamoDB のコアコンポーネント」を参照してください。

AWS SDK を使用して、DynamoDB テーブルを作成する

次のコード例は、AWS SDK を使用して DynamoDB テーブルを作成する方法を示しています。

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        /// <summary>
        /// Creates a new Amazon DynamoDB table and then waits for the new
        /// table to become active.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.CreateTableAsync(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5,
                },
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await 
client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
```

- API の詳細については、『AWS SDK for .NET API リファレンス』の「CreateTable」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#       -p provisioned_throughput -- Provisioned throughput settings for the
 table.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema provisioned_throughput
 response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
```

```
   echo " -k key_schema -- JSON file path of a list of attributes and their key
types."
   echo " -p provisioned_throughput -- Provisioned throughput settings for the
table."
   echo ""
 }

 # Retrieve the calling parameters.
 while getopts "n:a:k:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     a) attribute_definitions="${OPTARG}" ;;
     k) key_schema="${OPTARG}" ;;
     p) provisioned_throughput="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_definitions" ]]; then
   errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
   usage
   return 1
 fi

 if [[ -z "$key_schema" ]]; then
   errecho "ERROR: You must provide a key schema json file path the -k
parameter."
   usage
```

```
    return 1
  fi

  if [[ -z "$provisioned_throughput" ]]; then
    errecho "ERROR: You must provide a provisioned throughput json file path the
 -p parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho "    provisioned_throughput:   $provisioned_throughput"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --key-schema file://"$key_schema" \
    --provisioned-throughput "$provisioned_throughput")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
```

```
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
```

```
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「CreateTable」を参照してください。

C++

SDK for C++

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;
    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、『AWS SDK for C++ API リファレンス』の「CreateTable」を参照してください。

CLI

AWS CLI

例 1: タグ付きのテーブルを作成するには

次の create-table の例では、指定された属性とキースキーマを使用して、MusicCollection という名前のテーブルを作成します。このテーブルはプロビジョニングされたスループットを使用し、保存時にはデフォルトの AWS 所有の CMK を使用して暗号化されます。またこのコマンドは、Owner キーと blueTeam 値を使用して、テーブルにタグを適用します。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 2: オンデマンドモードでテーブルを作成するには

次の例は、プロビジョニングされたスループットモードではなく、オンデマンドモードを使用して `MusicCollection` というテーブルを作成します。これは、ワークロードが予測できないテーブルに役立ちます。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 3: テーブルを作成してカスタマーマネージド CMK で暗号化するには

次の例では、MusicCollection という名前のテーブルを作成し、カスタマーマネージド CMK を使用して暗号化します。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 4: ローカルセカンダリインデックスを持つテーブルを作成するには

次の例では、指定された属性とキースキーマを使用して、AlbumTitleIndex という名前のローカルセカンダリインデックスを持つ MusicCollection という名前のテーブルを作成します。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S AttributeName=AlbumTitle,AttributeType=S
 \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 5: グローバルセカンダリインデックスを持つテーブルを作成するには

次の例では、GameTitleIndex という名前のグローバルセカンダリインデックスを持つ GameScores という名前のテーブルを作成します。ベーステーブルには、パーティションキー UserId とソートキー GameTitle があり、特定のゲームの個々のユーザーのベストスコアを効率的に見つけることができます。一方、GSI にはパーティションキー GameTitle とソートキー TopScore があり、特定のゲームの全体的な最高スコアをすばやく見つけることができます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S AttributeName=TopScore,AttributeType=N \
    --key-schema AttributeName=UserId,KeyType=HASH \
                AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 6: 複数のグローバルセカンダリインデックスを持つテーブルを一度に作成するには

次の例では、2 つのグローバルセカンダリインデックスを持つ `GameScores` という名前のテーブルを作成します。GSI スキーマはコマンドラインではなくファイルを介して渡されます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S AttributeName=TopScore,AttributeType=N
 AttributeName=Date,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

`gsi.json` の内容：

```
[
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 7: ストリームが有効なテーブルを作成するには

次の例では、DynamoDB ストリームを有効にした状態の GameScores という名前のテーブルを作成します。各アイテムの新しいイメージと古いイメージの両方がストリームに書き込まれます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 8: Keys-Only ストリームが有効なテーブルを作成するには

次の例では、DynamoDB ストリームを有効にした状態の GameScores という名前のテーブルを作成します。変更された項目のキー属性のみがストリームに書き込まれます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB Streams の変更データキャプチャ」を参照してください。

例 9: Standard Infrequent Access クラスでテーブルを作成するには

次の例は、GameScores という名前のテーブルを作成し、Standard-Infrequent Access (DynamoDB Standard-IA) テーブルクラスを割り当てます。このテーブルクラスは、ストレージが主なコストとなるように最適化されています。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「テーブルクラス」を参照してください。

例 10: 削除保護を有効にしたテーブルを作成するには

次の例では、GameScores というテーブルを作成し、削除保護を有効にします。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「削除保護の使用」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「CreateTable」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable() (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(context.TODO(),
 &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName: aws.String(basics.TableName),
  ProvisionedThroughput: &types.ProvisionedThroughput{
   ReadCapacityUnits:  aws.Int64(10),
   WriteCapacityUnits: aws.Int64(10),
  },
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(context.TODO(), &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
 }
 return tableDesc, err
}
```

- API の詳細については、『AWS SDK for Go API リファレンス』の「CreateTable」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key>

                Where:
                    tableName - The Amazon DynamoDB table to create (for example,
 Music3).
                    key - The key for the Amazon DynamoDB table (for example,
 Artist).
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
```

```
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
               .region(region)
               .build();

       String result = createTable(ddb, tableName, key);
       System.out.println("New table is " + result);
       ddb.close();
   }

   public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
       DynamoDbWaiter dbWaiter = ddb.waiter();
       CreateTableRequest request = CreateTableRequest.builder()
               .attributeDefinitions(AttributeDefinition.builder()
                       .attributeName(key)
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(KeySchemaElement.builder()
                       .attributeName(key)
                       .keyType(KeyType.HASH)
                       .build())
               .provisionedThroughput(ProvisionedThroughput.builder()
                       .readCapacityUnits(10L)
                       .writeCapacityUnits(10L)
                       .build())
               .tableName(tableName)
               .build();

       String newTable;
       try {
           CreateTableResponse response = ddb.createTable(request);
           DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                   .tableName(tableName)
                   .build();

           // Wait until the Amazon DynamoDB table is created.
           WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
           waiterResponse.matched().response().ifPresent(System.out::println);
           newTable = response.tableDescription().tableName();
           return newTable;

       } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の「CreateTable」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、『AWS SDK for JavaScript API リファレンス』の「CreateTable」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、『AWS SDK for JavaScript API リファレンス』の「CreateTable」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun createNewTable(tableNameVal: String, key: String): String? {
    val attDef = AttributeDefinition {
        attributeName = key
        attributeType = ScalarAttributeType.S
    }

    val keySchemaVal = KeySchemaElement {
        attributeName = key
        keyType = KeyType.Hash
    }

    val provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef)
        keySchema = listOf(keySchemaVal)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->

        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists { // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「CreateTable」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルを作成します。

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
 'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
 'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
 'WriteCapacityUnits' => 10],
        ]);
    }
```

- API の詳細については、『AWS SDK for PHP API リファレンス』の「CreateTable」を参照してください。

PowerShell

Tools for PowerShell

例 1: この例では、「ForumName」(キータイプハッシュ) と「Subject」(キータイプ範囲) で構成されるプライマリキーを持つ「Thread」という名前のテーブルを作成します。テーブルの作成に使用したスキーマは、図のように各 cmdlet にパイプ処理するか、-Schema パラメータを使用して指定できます。

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

出力:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

例 2: この例では、「ForumName」(キータイプハッシュ) と「Subject」(キータイプ範囲) で構成されるプライマリキーを持つ Thread という名前のテーブルを作成します。ローカルセカンダリインデックスも定義されます。ローカルセカンダリインデックスのキーは、テーブルのプライマリハッシュキー (ForumName) から自動的に設定されます。テーブルの作成に使用したスキーマは、図のように各 cmdlet にパイプ処理するか、-Schema パラメータを使用して指定できます。

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

出力:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

例 3: この例では、単一のパイプラインを使用して、「ForumName」(キータイプハッシュ) と「Subject」(キータイプ範囲) で構成されるプライマリキー、およびローカルセカンダリインデックスを持つ「Thread」という名前のテーブルを作成する方法を示します。TableSchema がパイプラインまたは -Schema パラメータから提供されない場合、Add-DDBKeySchema と Add-DDBIndexSchema によって新しい TableSchema オブジェクトが作成されます。

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

出力:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「CreateTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

映画データを格納するためのテーブルを作成します。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                ProvisionedThroughput={
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10,
                },
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「CreateTable」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource
  attr_reader :table_name
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        {attribute_name: "year", key_type: "HASH"},  # Partition key
        {attribute_name: "title", key_type: "RANGE"}  # Sort key
      ],
      attribute_definitions: [
        {attribute_name: "year", attribute_type: "N"},
        {attribute_name: "title", attribute_type: "S"}
      ],
    provisioned_throughput: {read_capacity_units: 10, write_capacity_units:
 10})
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、『AWS SDK for Ruby API リファレンス』の「CreateTable」を参照してください。

Rust

SDK for Rust

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .provisioned_throughput(pt)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateTable」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
      " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
```

```
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「CreateTable」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = CreateTableInput(
            attributeDefinitions: [
                DynamoDBClientTypes.AttributeDefinition(attributeName: "year",
 attributeType: .n),
                DynamoDBClientTypes.AttributeDefinition(attributeName: "title",
 attributeType: .s),
            ],
            keySchema: [
                DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
            ],
            provisionedThroughput: DynamoDBClientTypes.ProvisionedThroughput(
                readCapacityUnits: 10,
                writeCapacityUnits: 10
            ),
            tableName: self.tableName
        )
        let output = try await client.createTable(input: input)
        if output.tableDescription == nil {
            throw MoviesError.TableNotFound
        }
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「CreateTable」を参照してください。

DynamoDB のその他の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

項目を DynamoDB テーブルに書き込む

AWS Management Console、AWS CLI、または AWS SDK を使用して、DynamoDB テーブルに項目を書き込むことができます。項目の詳細については、「Amazon DynamoDB のコアコンポーネント」を参照してください。

AWS SDK を使用して DynamoDB テーブルに項目を書き込む

次のコード例は、AWS SDK を使用して DynamoDB テーブルに項目を書き込む方法を示しています。

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Adds a new item to the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing informtation for
        /// the movie to add to the table.</param>
        /// <param name="tableName">The name of the table where the item will be
 added.</param>
        /// <returns>A Boolean value that indicates the results of adding the
 item.</returns>
        public static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「PutItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

ℹ Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }
  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
```

```
#           0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「PutItem」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Put an item in an Amazon DynamoDB table.
/*!
```

```
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「PutItem」を参照してください。

CLI

AWS CLI

例 1: テーブルに項目を追加するには

次の `put-item` の例は、MusicCollection テーブルに新しい項目を追加します。

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

`item.json` の内容:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

出力:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

例 2: テーブル内の項目を条件付きで上書きするには

次の put-item の例は、MusicCollection テーブル内の既存の項目に Greatest Hits の値を持つ AlbumTitle 属性がある場合にのみ、その項目を上書きします。このコマンドは、その項目の以前の値を返します。

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

item.json の内容:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

names.json の内容:

```
{
    "#A": "AlbumTitle"
}
```

`values.json` の内容:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

出力:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

キーが存在する場合は、次のような出力が表示されます。

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「PutItem」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(context.TODO(), &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
```

```
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「PutItem」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用してテーブルに項目を配置します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is 
placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example, 
Artist).
                    keyval - The key value that represents the item to get (for 
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs 
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example, 
Happy Day).
                **Warning** This program will  place an item that you specify 
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue, 
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }
    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「PutItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「PutCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutItem」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルに項目を配置します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルに項目を配置します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「PutItem」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request = PutItemRequest {
        tableName = tableNameVal
        item = itemValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「PutItem」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「PutItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: 新しい項目を作成する、または既存の項目を新しい項目で置き換えます。

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
```

```
Set-DDBItem -TableName 'Music' -Item $item
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「PutItem」を参照してください。

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「PutItem」を参照してください。

Ruby

SDK for Ruby

> ⓘ Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        "year" => movie[:year],
        "title" => movie[:title],
        "info" => {"plot" => movie[:plot], "rating" => movie[:rating]}})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「PutItem」を参照してください。

Rust

SDK for Rust

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「PutItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「PutItem」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    /// 
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        // Get a DynamoDB item containing the movie data.
        let item = try await movie.getAsItem()

        // Send the `PutItem` request to Amazon DynamoDB.

        let input = PutItemInput(
            item: item,
            tableName: self.tableName
        )
        _ = try await client.putItem(input: input)
    }

    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
 [Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「PutItem」を参照してください。

DynamoDB のその他の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

DynamoDB テーブルから項目を読み込む

AWS Management Console、AWS CLI、または AWS SDK を使用して、DynamoDB テーブルから項目を読み込むことができます。項目の詳細については、「Amazon DynamoDB のコアコンポーネント」を参照してください。

AWS SDK を使用して DynamoDB テーブルから項目を読み込む

次のコード例は、AWS SDK を使用して DynamoDB テーブルから項目を読み込む方法を示しています。

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Gets information about an existing movie from the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing information about
        /// the movie to retrieve.</param>
        /// <param name="tableName">The name of the table containing the movie.</
param>
        /// <returns>A Dictionary object containing information about the item
        /// retrieved.</returns>
        public static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await client.GetItemAsync(request);
            return response.Item;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
#############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
  }
  query=""
  while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi
```

```
  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
```

```
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「GetItem」を参照してください。

C++

SDK for C++

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「GetItem」を参照してください。

CLI

AWS CLI

例 1: テーブル内の項目を読み込むには

次の get-item の例は、MusicCollection テーブルから項目を取得します。テーブルにはハッシュおよび範囲プライマリキー (ArtistおよびSongTitle) があるため、これらの属性の両方を指定する必要があります。このコマンドは、オペレーションによって消費される読み込み容量に関する情報も要求します。

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

key.json の内容：

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

出力:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目の読み込み」を参照してください。

例 2: 整合性のある読み込みを使用して項目を読み込むには

次の例では、強力な整合性のある読み込みを使用して MusicCollection テーブルから項目を読み込みます。

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

key.json の内容：

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

出力:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目の読み込み」を参照してください。

例 3: 項目の特定の属性を取得するには

次の例は、射影式を使用して目的のアイテムの 3 つの属性のみを取得します。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

`names.json` の内容：

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

出力:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目の読み込み」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「GetItem」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(title string, year int) (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(context.TODO(),
 &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}




// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「GetItem」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用して、テーブルから項目を取得します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();
            if (returnedItem.isEmpty())
```

```
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「GetCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetItem」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

テーブルから項目を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
```

```
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルから項目を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetItem」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
suspend fun getSpecificItem(tableNameVal: String, keyName: String, keyVal:
 String) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request = GetItemRequest {
        key = keyToGet
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetItem」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「GetItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: パーティションキー SongTitle とソートキー Artist を含む DynamoDB 項目を返します。

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「GetItem」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「GetItem」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end
  # Gets movie data from the table for a specific movie.
  #
  # @param title [String] The title of the movie.
  # @param year [Integer] The release year of the movie.
  # @return [Hash] The data about the requested movie.
  def get_item(title, year)
    @table.get_item(key: {"year" => year, "title" => title})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「GetItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        oo_item = lo_dyn->getitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key ).
        DATA(lt_attr) = oo_item->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( )
          && 'Movie year is: ' && lo_year->get_n( )
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「GetItem」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
/// Return a `Movie` record describing the specified movie from the Amazon
/// DynamoDB table.
///
/// - Parameters:
///   - title: The movie's title (`String`).
///   - year: The movie's release year (`Int`).
///
/// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
///
/// - Returns: A `Movie` record with the movie's details.
func get(title: String, year: Int) async throws -> Movie {
    guard let client = self.ddbClient else {
        throw MoviesError.UninitializedClient
    }

    let input = GetItemInput(
        key: [
            "year": .n(String(year)),
            "title": .s(title)
        ],
        tableName: self.tableName
    )
    let output = try await client.getItem(input: input)
    guard let item = output.item else {
        throw MoviesError.ItemNotFound
    }

    let movie = try Movie(withItem: item)
    return movie
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「GetItem」を参照してください。

DynamoDB のその他の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

DynamoDB テーブルで項目を更新する

AWS Management Console、AWS CLI、または AWS SDK を使用して、DynamoDB テーブルから項目を更新することができます。項目の詳細については、「Amazon DynamoDB のコアコンポーネント」を参照してください。

AWS SDK を使用して DynamoDB テーブルで項目を更新する

次のコード例は、AWS SDK を使用して DynamoDB テーブルで項目を更新する方法を示しています。

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Updates an existing item in the movies table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing information for
        /// the movie to update.</param>
        /// <param name="newInfo">A MovieInfo object that contains the
        /// information that will be changed.</param>
        /// <param name="tableName">The name of the table that contains the
 movie.</param>
        /// <returns>A Boolean value that indicates the success of the
 operation.</returns>
        public static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「UpdateItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
```

```
#############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi
  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi
  if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

この例で使用されているユーティリティ関数。

```
################################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
```

```
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「UpdateItem」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */
```

```
/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);
    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「UpdateItem」を参照してください。

CLI

AWS CLI

例 1: テーブル内の項目を更新するには

次の update-item の例では、MusicCollection テーブルの項目を更新します。新しい属性 (Year) を追加して、AlbumTitle 属性を更新します。更新後に表示される項目内の属性はすべて、レスポンスで返されます。

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

key.json の内容：

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

`expression-attribute-names.json` の内容：

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

`expression-attribute-values.json` の内容：

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

出力:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

例 2: 項目を条件付きで更新するには

次の例は、既存の項目に `Year` 属性がない場合にのみ、`MusicCollection` テーブル内の項目を更新します。

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

`key.json` の内容：

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

`expression-attribute-names.json` の内容：

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

`expression-attribute-values.json` の内容：

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

項目にすでに Year 属性がある場合、DynamoDB は次の出力を返します。

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「UpdateItem」を参照してください。

Go

SDK for Go V2

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
```

```
// expression.
func (basics TableBasics) UpdateMovie(movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(context.TODO(),
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}




// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「UpdateItem」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用してテーブルで項目を更新します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }
        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「UpdateItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「UpdateCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「UpdateItem」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] = AttributeValueUpdate {
        value = AttributeValue.S(updateVal)
        action = AttributeAction.Put
    }

    val request = UpdateItemRequest {
        tableName = tableNameVal
        key = itemKey
        attributeUpdates = updatedValues
    }
    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「UpdateItem」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「UpdateItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: パーティションキー SongTitle とソートキー Artist を含む DynamoDB 項目のジャンル属性を「Rap」に設定します。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

出力:

```
Name                           Value
----                           -----
```

```
Genre                           Rap
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「UpdateItem」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

更新式を使用して項目を更新します。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

算術演算を含む更新式を使用して、項目を更新します。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

特定の条件を満たす場合にのみ、項目を更新します。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「UpdateItem」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table
  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)

    response = @table.update_item(
      key: {"year" => movie[:year], "title" => movie[:title]},
      update_expression: "set info.rating=:r",
      expression_attribute_values: { ":r" => movie[:rating] },
      return_values: "UPDATED_NEW")
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「UpdateItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    TRY.
```

```
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「UpdateItem」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
               -> [Swift.String:DynamoDBClientTypes.AttributeValue]? {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        // Build the update expression and the list of expression attribute
        // values. Include only the information that's changed.

        var expressionParts: [String] = []
        var attrValues: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]

        if rating != nil {
            expressionParts.append("info.rating=:r")
            attrValues[":r"] = .n(String(rating!))
        }
        if plot != nil {
            expressionParts.append("info.plot=:p")
            attrValues[":p"] = .s(plot!)
        }
        let expression: String = "set \(expressionParts.joined(separator: ", "))"

        let input = UpdateItemInput(
            // Create substitution tokens for the attribute values, to ensure
            // no conflicts in expression syntax.
            expressionAttributeValues: attrValues,
            // The key identifying the movie to update consists of the release
            // year and title.
            key: [
                "year": .n(String(year)),
                "title": .s(title)
            ],
            returnValues: .updatedNew,
            tableName: self.tableName,
            updateExpression: expression
        )
        let output = try await client.updateItem(input: input)
        guard let attributes: [Swift.String:DynamoDBClientTypes.AttributeValue] =
output.attributes else {
            throw MoviesError.InvalidAttributes
        }
        return attributes
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「UpdateItem」を参照してください。

DynamoDB のその他の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

DynamoDB テーブルで項目を削除する

AWS Management Console、AWS CLI、または AWS SDK を使用して、DynamoDB テーブルから項目を削除できます。項目の詳細については、「Amazon DynamoDB のコアコンポーネント」を参照してください。

AWS SDK を使用して DynamoDB テーブルで項目を削除する

次のコード例は、AWS SDK を使用して DynamoDB テーブルで項目を削除する方法を示しています。

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Deletes a single item from a DynamoDB table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table from which the item
        /// will be deleted.</param>
        /// <param name="movieToDelete">A movie object containing the title and
        /// year of the movie to delete.</param>
        /// <returns>A Boolean value indicating the success or failure of the
        /// delete operation.</returns>
        public static async Task<bool> DeleteItemAsync(
            AmazonDynamoDBClient client,
            string tableName,
            Movie movieToDelete)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
 movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = key,
            };

            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0
```

```
}
```

この例で使用されているユーティリティ関数。

```
################################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteItem」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Delete an item from an Amazon DynamoDB table.
/*!
  \sa deleteItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::deleteItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteItemRequest request;

    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteItemOutcome &outcome =
 dynamoClient.DeleteItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item \"" << partitionValue << "\" deleted!" << std::endl;
    }
    else {
        std::cerr << "Failed to delete item: " << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteItem」を参照してください。

CLI

AWS CLI

例 1: 項目を削除するには

次の delete-item の例は、MusicCollection テーブルから項目を削除し、削除した項目とそのリクエストで使用された容量に関する詳細を取得します。

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

`key.json` の内容:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

出力:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Blue Sky Blues"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Scared of My Shadow"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 2.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

例 2: 条件付きで項目を削除するには

次の例では、ProductCategory が Sporting Goods または Gardening Supplies で、その価格が 500 および 600 の場合のみ、ProductCatalog テーブルから項目を削除します。削除された項目に関する詳細が返されます。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

names.json の内容:

```
{
    "#P": "Price"
}
```

values.json の内容:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

出力:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteItem」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(context.TODO(),
 &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}




// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「DeleteItem」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyval>

                Where:
                    tableName - The Amazon DynamoDB table to delete the item from
 (for example, Music3).
```

```
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to delete
 (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Deleting item \"%s\" from %s\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();

        try {
            ddb.deleteItem(deleteReq);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> ⓘ Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「DeleteCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。

- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteItem」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルから項目を削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルから項目を削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteItem」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteDynamoDBItem(tableNameVal: String, keyName: String, keyVal:
 String) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)
    val request = DeleteItemRequest {
        tableName = tableNameVal
        key = keyToGet
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DeleteItem」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「DeleteItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定されたキーと一致する DynamoDB 項目を削除します。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「DeleteItem」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

項目が特定の条件を満たす場合にのみ削除されるように条件を指定できます。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def delete_underrated_movie(self, title, year, rating):
        """
        Deletes a movie only if it is rated below a specified value. By using a
        condition expression in a delete operation, you can specify that an item
 is
        deleted only when it meets certain criteria.
        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        :param rating: The rating threshold to check before deleting the movie.
        """
        try:
            self.table.delete_item(
                Key={"year": year, "title": title},
                ConditionExpression="info.rating <= :val",
                ExpressionAttributeValues={":val": Decimal(str(rating))},
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"ConditionalCheckFailedException":
                logger.warning(
                    "Didn't delete %s because its rating is greater than %s.",
                    title,
                    rating,
                )
            else:
                logger.error(
                    "Couldn't delete movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DeleteItem」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Deletes a movie from the table.
  #
  # @param title [String] The title of the movie to delete.
  # @param year [Integer] The release year of the movie to delete.
  def delete_item(title, year)
    @table.delete_item(key: {"year" => year, "title" => title})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete movie #{title}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteItem」を参照してください。

Rust

SDK for Rust

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        DATA(lo_resp) = lo_dyn->deleteitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key_input ).
        MESSAGE 'Deleted one item.' TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
```

```
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「DeleteItem」を参照してください。

Swift

SDK for Swift

Note

これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = DeleteItemInput(
            key: [
                "year": .n(String(year)),
                "title": .s(title)
            ],
            tableName: self.tableName
        )
        _ = try await client.deleteItem(input: input)
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「DeleteItem」を参照してください。

DynamoDB のその他の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

DynamoDB テーブルに対してクエリを実行する

AWS Management Console、AWS CLI、または AWS SDK を使用して DynamoDB テーブルでクエリを実行できます。クエリの詳細については、「DynamoDB のクエリオペレーション」を参照してください。

AWS SDK を使用して、DynamoDB テーブルに対してクエリを実行する

次のコード例は、AWS SDK を使用して DynamoDB テーブルに対してクエリを実行する方法を示しています。

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Queries the table for movies released in a particular year and
        /// then displays the information for the movies returned.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table to query.</param>
        /// <param name="year">The release year for which we want to
        /// view movies.</param>
        /// <returns>The number of movies that match the query.</returns>
        public static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "title",
                  "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            Search search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Query」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
#############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
 parameter."
    usage
    return 1
```

```
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

この例で使用されているユーティリティ関数。

```
################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
  return 0
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「Query」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
&clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
   Aws::DynamoDB::Model::QueryRequest request;

   request.SetTableName(tableName);

   if (!projectionExpression.empty()) {
       request.SetProjectionExpression(projectionExpression);
   }

   // Set query key condition expression.
   request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

   // Set Expression AttributeValues.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
   attributeValues.emplace(":valueToMatch", partitionValue);

   request.SetExpressionAttributeValues(attributeValues);

   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
                         << std::endl;
               // Iterate each item and print.
               for (const auto &item: items) {
                   std::cout
```

```
                            <<
 "******************************************************"
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「Query」を参照してください。

CLI

AWS CLI

例 1: テーブルにクエリを実行するには

次の query の例では、MusicCollection テーブルの項目にクエリを実行します。テーブルにはハッシュおよび範囲プライマリキー (ArtistおよびSongTitle) がありますが、このクエリではハッシュキー値のみを指定します。「No One You Know」という名前のアーティストの曲タイトルが返されます。

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

`expression-attributes.json` の内容：

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 2: 強力な整合性のある読み込みを使用してテーブルにクエリを実行し、インデックスを降順で走査するには

次の例では、最初の例と同じクエリを実行しますが、結果は逆の順序で返され、強力な整合性のある読み込みが使用されます。

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

`expression-attributes.json` の内容：

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 3: 特定の結果を除外するには

次の例は、MusicCollection をクエリしますが、AlbumTitle 属性に特定の値を含む結果を除外します。このフィルターは項目が読み込まれた後に適用されるため、ScannedCount または ConsumedCapacity には影響しないことに注意してください。

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

values.json の内容：

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

names.json の内容：

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

出力:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 4: 項目数だけを取得するには

次の例は、クエリに一致する項目数を取得しますが、項目自体は取得しません。

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

`expression-attributes.json` の内容：

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 5: インデックスをクエリするには

次の例は、ローカルセカンダリインデックス`AlbumTitleIndex`をクエリします。クエリは、ローカルセカンダリインデックスに射影されたベーステーブルのすべての属性を返します。ローカルセカンダリインデックスまたはグローバルセカンダリインデックスをクエリする場合は、`table-name`パラメータを使用してベーステーブルの名前も指定する必要があることに注意してください。

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

`expression-attributes.json`の内容：

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「Query」を参照してください。

Go

SDK for Go V2

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(releaseYear int) ([]Movie, error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(context.TODO())
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}




// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「Query」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用してテーブルに対してクエリを実行します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
```

```
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

`DynamoDbClient` とセカンダリインデックスを使用してテーブルに対してクエリを実行します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「Query」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「QueryCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Query」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request = QueryRequest {
        tableName = tableNameVal
        keyConditionExpression = "$partitionAlias = :$partitionKeyName"
        expressionAttributeNames = attrNameAlias
        this.expressionAttributeValues = attrValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
```

```
        val response = ddb.query(request)
        return response.count
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「Query」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「Query」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定された SongTitle と Artist を含む DynamoDB 項目を返すクエリを呼び出します。

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
```

```
AlbumTitle                     Somewhat Famous
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「Query」を参照してください。

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

キー条件式を使用して項目に対してクエリを実行します。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

項目に対してクエリを実行し、データのサブセットを返すように射影します。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「Query」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: "#yr = :year",
      expression_attribute_names: {"#yr" => "year"},
      expression_attribute_values: {":year" => year})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「Query」を参照してください。

Rust

SDK for Rust

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

指定した年に作られた映画を検索します。

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「Query」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```

    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「Query」を参照してください。

Swift

SDK for Swift

> **Note**
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = QueryInput(
            expressionAttributeNames: [
                "#y": "year"
            ],
            expressionAttributeValues: [
                ":y": .n(String(year))
            ],
            keyConditionExpression: "#y = :y",
            tableName: self.tableName
        )
        let output = try await client.query(input: input)

        guard let items = output.items else {
            throw MoviesError.ItemNotFound
        }

        // Convert the found movies into `Movie` objects and return an array
        // of them.

        var movieList: [Movie] = []
        for item in items {
            let movie = try Movie(withItem: item)
            movieList.append(movie)
        }
        return movieList
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「Query」を参照してください。

DynamoDB のその他の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

DynamoDB テーブルをスキャンする

AWS Management Console、AWS CLI、または AWS SDK を使用して DynamoDB テーブルのスキャンを実行できます。スキャンの詳細については、「DynamoDB でのスキャンの使用」を参照してください。

AWS SDK を使用して、DynamoDB テーブルをスキャンする

次のコード例は、AWS SDK を使用して DynamoDB テーブルをスキャンする方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                  { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Scan」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
```

```
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$filter_expression" ]]; then
   errecho "ERROR: You must provide a filter expression with the -f parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_names" ]]; then
   errecho "ERROR: You must provide expression attribute names with the -a
parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_values" ]]; then
   errecho "ERROR: You must provide expression attribute values with the -v
parameter."
   usage
   return 1
 fi

 if [[ -z "$projection_expression" ]]; then
   response=$(aws dynamodb scan \
     --table-name "$table_name" \
     --filter-expression "$filter_expression" \
     --expression-attribute-names file://"$expression_attribute_names" \
     --expression-attribute-values file://"$expression_attribute_values")
 else
   response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
```

```
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「Scan」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Scan an Amazon DynamoDB table.
/*!
  \sa scanTable()
  \param tableName: Name for the DynamoDB table.
  \param projectionExpression: An optional projection expression, ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                 const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::ScanRequest request;
    request.SetTableName(tableName);

    if (!projectionExpression.empty())
        request.SetProjectionExpression(projectionExpression);

    Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
 all_items;
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 last_evaluated_key; // Used for pagination;
    do {
        if (!last_evaluated_key.empty()) {
            request.SetExclusiveStartKey(last_evaluated_key);
        }
        const Aws::DynamoDB::Model::ScanOutcome &outcome =
 dynamoClient.Scan(request);
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            all_items.insert(all_items.end(), items.begin(), items.end());

            last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Scan items: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }

    } while (!last_evaluated_key.empty());
    if (!all_items.empty()) {
        std::cout << "Number of items retrieved from scan: " << all_items.size()
                  << std::endl;
        // Iterate each item and print.
        for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 &itemMap: all_items) {
            std::cout << "******************************************************"
                      << std::endl;
            // Output each retrieved field and its value.
            for (const auto &itemEntry: itemMap)
                std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
                          << std::endl;
        }
    }

    else {
        std::cout << "No items found in table: " << tableName << std::endl;
    }

    return true;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「Scan」を参照してください。

CLI

AWS CLI

テーブルをスキャンするには

次の scan の例は、MusicCollection テーブル全体をスキャンし、その結果をアーティスト「No One You Know」の曲に絞り込みます。各項目について、アルバムタイトルと曲タイトルのみが返されます。

```
aws dynamodb scan \
    --table-name MusicCollection \
    --filter-expression "Artist = :a" \
    --projection-expression "#ST, #AT" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json
```

`expression-attribute-names.json` の内容：

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

`expression-attribute-values.json` の内容：

```
{
    ":a": {"S": "No One You Know"}
}
```

出力:

```
{
    "Count": 2,
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "AlbumTitle": {
                "S": "Somewhat Famous"
            }
        },
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            },
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            }
        }
    ],
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB でのスキャンの使用」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「Scan」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(startYear int, endYear int) ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(context.TODO())
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「Scan」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用して Amazon DynamoDB テーブルをスキャンします。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To scan items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, See the EnhancedScanRecords example.
 */

public class DynamoDBScanItems {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information from
 (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        scanItems(ddb, tableName);
        ddb.close();
    }

    public static void scanItems(DynamoDbClient ddb, String tableName) {
        try {
            ScanRequest scanRequest = ScanRequest.builder()
                    .tableName(tableName)
                    .build();

            ScanResponse response = ddb.scan(scanRequest);
            for (Map<String, AttributeValue> item : response.items()) {
                Set<String> keys = item.keySet();
                for (String key : keys) {
                    System.out.println("The key name is " + key + "\n");
                    System.out.println("The value is " + item.get(key).s());
                }
            }

        } catch (DynamoDbException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「Scan」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「ScanCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Scan」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values
 you want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「Scan」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun scanItems(tableNameVal: String) {
    val request = ScanRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「Scan」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    $yearsKey = [
        'Key' => [
            'year' => [
                'N' => [
                    'minRange' => 1990,
                    'maxRange' => 1999,
                ],
            ],
        ],
    ];
    $filter = "year between 1990 and 1999";
    echo "\nHere's a list of all the movies released in the 90s:\n";
    $result = $service->scan($tableName, $yearsKey, $filter);
    foreach ($result['Items'] as $movie) {
        $movie = $marshal->unmarshalItem($movie);
        echo $movie['title'] . "\n";
    }

public function scan(string $tableName, array $key, string $filters)
{
    $query = [
        'ExpressionAttributeNames' => ['#year' => 'year'],
        'ExpressionAttributeValues' => [
            ":min" => ['N' => '1990'],
            ":max" => ['N' => '1999'],
        ],
        'FilterExpression' => "#year between :min and :max",
        'TableName' => $tableName,
    ];
    return $this->dynamoDbClient->scan($query);
}
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「Scan」を参照してください。

PowerShell

Tools for PowerShell

例 1: Music テーブルのすべての項目を返します。

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

例 2: Music テーブル内の CriticRating が 9 以上の項目を返します。

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「Scan」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「Scan」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Scans for movies that were released in a range of years.
  # Uses a projection expression to return a subset of data for each movie.
  #
  # @param year_range [Hash] The range of years to retrieve.
  # @return [Array] The list of movies released in the specified years.
  def scan_items(year_range)
    movies = []
    scan_hash = {
      filter_expression: "#yr between :start_yr and :end_yr",
      projection_expression: "#yr, title, info.rating",
      expression_attribute_names: {"#yr" => "year"},
      expression_attribute_values: {
        ":start_yr" => year_range[:start], ":end_yr" => year_range[:end]}
    }
    done = false
    start_key = nil
    until done
      scan_hash[:exclusive_start_key] = start_key unless start_key.nil?
      response = @table.scan(scan_hash)
      movies.concat(response.items) unless response.items.empty?
      start_key = response.last_evaluated_key
      done = start_key.nil?
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't scan for movies. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    movies
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「Scan」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
 Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("   {:?}", item);
    }
    Ok(())
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「Scan」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        " Scan movies for rating greater than or equal to the rating specified
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_rating }| ) ) ).
        DATA(lt_filter_conditions) = VALUE /aws1/
cl_dyncondition=>tt_filterconditionmap(
          ( VALUE /aws1/cl_dyncondition=>ts_filterconditionmap_maprow(
          key = 'rating'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |GE|
          ) ) ) ).
        oo_scan_result = lo_dyn->scan( iv_tablename = iv_table_name
          it_scanfilter = lt_filter_conditions ).
        DATA(lt_items) = oo_scan_result->get_items( ).
        LOOP AT lt_items INTO DATA(lo_item).
          " You can loop over to get individual attributes.
          DATA(lo_title) = lo_item[ key = 'title' ]-value.
          DATA(lo_year) = lo_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_scan_result->get_count( ).
        MESSAGE 'Found ' && lv_count && ' items' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
```

```
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「Scan」を参照してください。

Swift

SDK for Swift

Note

これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String:DynamoDBClientTypes.AttributeValue]? =
nil)
```

```
                   async throws -> [Movie] {
        var movieList: [Movie] = []

        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = ScanInput(
            consistentRead: true,
            exclusiveStartKey: startKey,
            expressionAttributeNames: [
                "#y": "year"           // `year` is a reserved word, so use `#y`
 instead.
            ],
            expressionAttributeValues: [
                ":y1": .n(String(firstYear)),
                ":y2": .n(String(lastYear))
            ],
            filterExpression: "#y BETWEEN :y1 AND :y2",
            tableName: self.tableName
        )

        let output = try await client.scan(input: input)

        guard let items = output.items else {
            return movieList
        }

        // Build an array of `Movie` objects for the returned items.

        for item in items {
            let movie = try Movie(withItem: item)
            movieList.append(movie)
        }

        // Call this function recursively to continue collecting matching
        // movies, if necessary.

        if output.lastEvaluatedKey != nil {
            let movies = try await self.getMovies(firstYear: firstYear, lastYear:
 lastYear,
                         startKey: output.lastEvaluatedKey)
            movieList += movies
        }
        return movieList
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「Scan」を参照してください。

DynamoDB のその他の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

AWS SDK で DynamoDB を使用する

AWS ソフトウェア開発キット (SDK) は、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コードの例
AWS SDK for C++	AWS SDK for C++ コードの例
AWS CLI	AWS CLI コードの例
AWS SDK for Go	AWS SDK for Go コードの例
AWS SDK for Java	AWS SDK for Java コードの例
AWS SDK for JavaScript	AWS SDK for JavaScript コードの例
AWS SDK for Kotlin	AWS SDK for Kotlin コードの例
AWS SDK for .NET	AWS SDK for .NET コードの例
AWS SDK for PHP	AWS SDK for PHP コードの例
AWS Tools for PowerShell	Tools for PowerShell のコード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コードの例
AWS SDK for Ruby	AWS SDK for Ruby コードの例

SDK ドキュメント	コードの例
AWS SDK for Rust	AWS SDK for Rust コードの例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コードの例
AWS SDK for Swift	AWS SDK for Swift コードの例

DynamoDB に固有の例については、「AWS SDK を使用した DynamoDB のコード例」を参照してください。

ⓘ 可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

DynamoDB と AWS SDK を使用したプログラミング

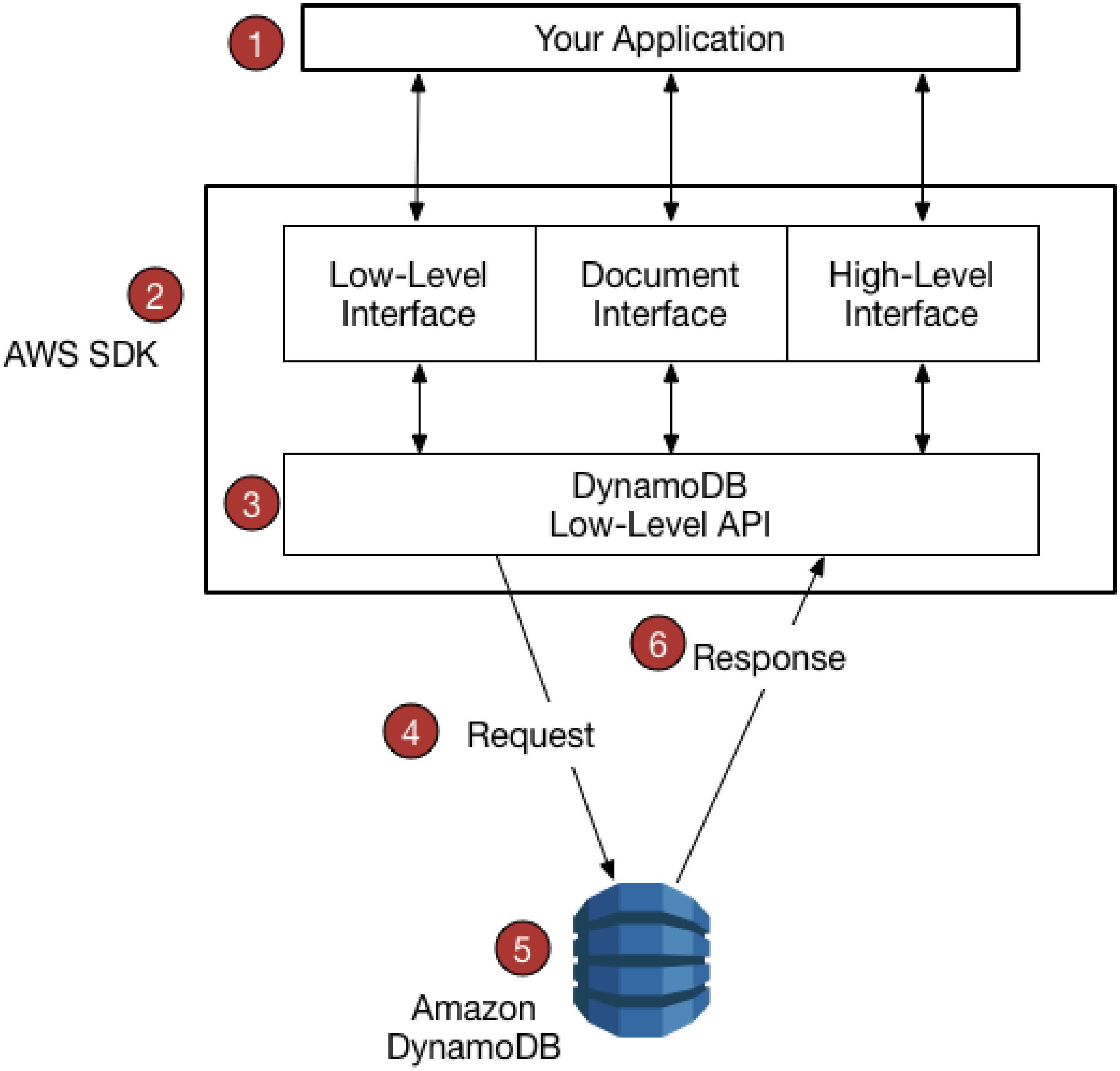
このセクションでは、デベロッパーに関連するトピックを説明します。代わりにコードのサンプルを実行する場合は、「このデベロッパーガイドのコード例の実行」を参照してください。

> Note
>
> 2017 年 12 月、AWS はすべての Amazon DynamoDB エンドポイントを移行し、Amazon Trust Services (ATS) が発行する、安全な証明書を使用するプロセスを開始しました。詳細については、「SSL/TLS 接続の確立に関する問題をトラブルシューティングする」を参照してください。

トピック

DynamoDB に対する AWS SDK サポートの概要

次の図表は、AWS SDK を使用した Amazon DynamoDB アプリケーションの、プログラミングの概要を示しています。



1. 自分が使用するプログラミング言語用のための AWS SDK を使用して、アプリケーションを作成します。

2. 各 AWS SDK には、DynamoDB を操作するプログラミングインターフェイスが 1 つ以上用意されています。使用できるインターフェイスは、使用するプログラミング言語と AWS SDK によって異なります。オプションには以下が含まれます。

   - 低レベルインターフェイス
   - ドキュメントインターフェイス

- オブジェクト永続性インターフェイス
- 高レベルのインターフェイス

3. AWS SDK は、低レベル DynamoDB API で使用する HTTP (S) リクエストを作成します。
4. AWS SDK は DynamoDB エンドポイントにリクエストを送信します。
5. DynamoDB がリクエストを実行します。リクエストが成功すると、DynamoDB は HTTP 200 レスポンスコード (OK) を返します。リクエストが失敗した場合は、DynamoDB は HTTP エラーコードとエラーメッセージを返します。
6. AWS SDK はレスポンスを処理し、アプリケーションに伝達します。

各 AWS SDK は、次のような重要なサービスをアプリケーションに提供します。

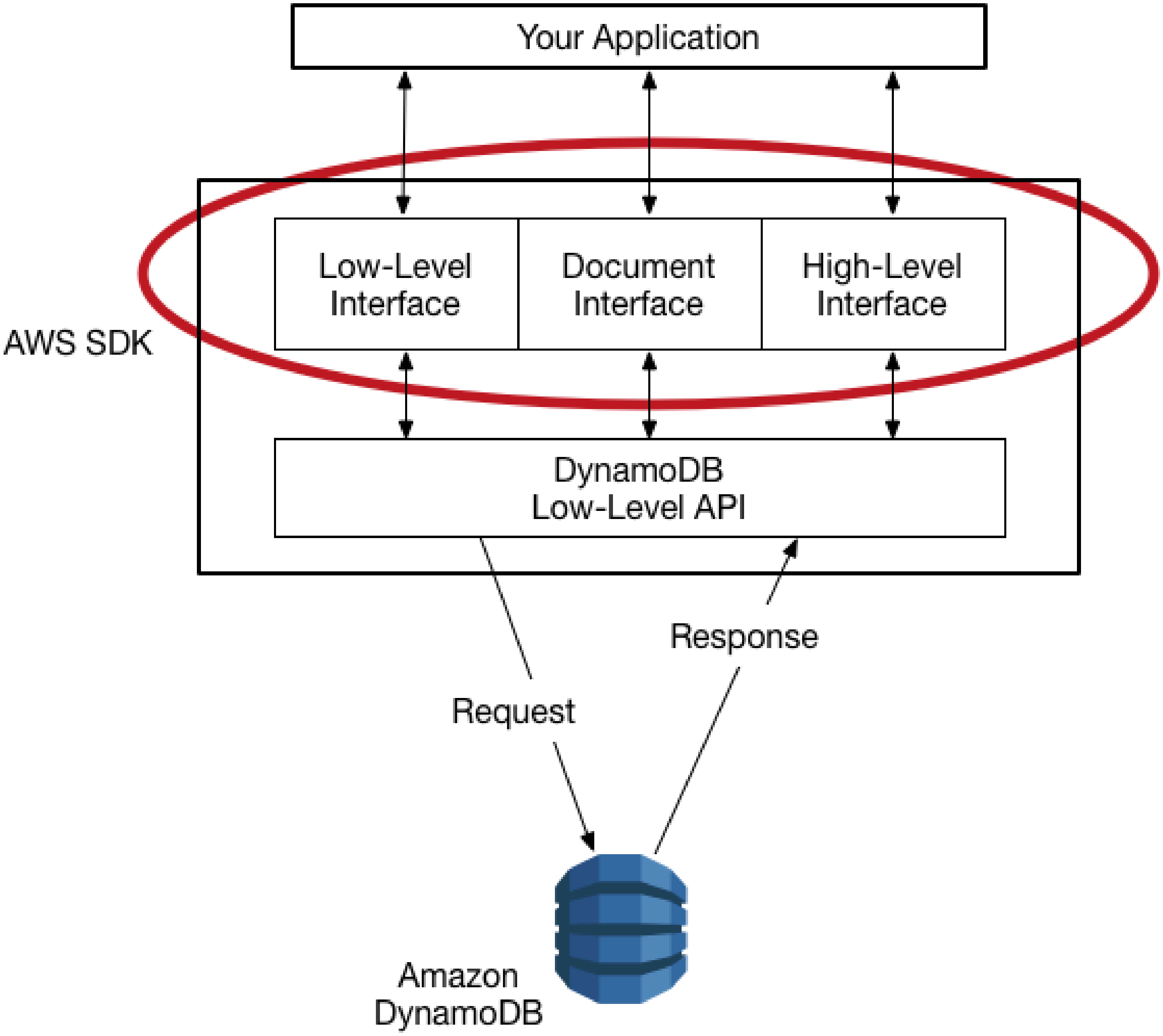
- HTTP (S) リクエストのフォーマットと、リクエストパラメータのシリアル化。
- リクエストごとの暗号化署名の生成。
- DynamoDB エンドポイントへのリクエストの転送と、DynamoDB からのレスポンスの受信。
- これらの応答からの結果の抽出。
- エラーが発生した場合の基本的な再試行ロジックの実装。

これらのタスクでは、コードを記述する必要はありません。

Note

AWS SDK のインストール方法とドキュメントの詳細については、「アマゾン ウェブ サービス用ツール」を参照してください。

プログラミングインターフェイス



各 AWS SDK には、Amazon DynamoDB を操作するプログラミングインターフェイスが 1 つ以上用意されています。これらのインターフェイスは、シンプルな低レベルの DynamoDB ラッパーから、オブジェクト指向の永続レイヤーまで、多岐にわたります。使用可能なインターフェイスは、AWS SDK およびプログラミング言語によって異なります。

次のセクションでは、AWS SDK for Java を例として使用して、使用可能なインターフェイスをいくつかハイライトします。(すべてのインターフェイスが、どの AWS SDK でも使用できるわけではありません。)

トピック

- 低レベルインターフェイス
- ドキュメントインターフェイス
- オブジェクト永続性インターフェイス

低レベルインターフェイス

それぞれの言語特有の AWS SDK は、低レベルの DynamoDB API リクエストによく似たメソッドを持つ、Amazon DynamoDB 用の低レベルインターフェイスを提供します。

場合によっては データ型記述子 を使用して、文字列型の S や、数値型の N など、属性のデータ型を識別する必要があります。

> Note
>
> すべての言語特有の AWS SDK で、低レベルインターフェイスを使用できます。

次の Java プログラムは、AWS SDK for Java の低レベルインターフェイスを使用しています。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName, String
 key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem = ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

ドキュメントインターフェイス

多くの AWS SDK はドキュメントインターフェイスを備えており、テーブルやインデックスに対してデータプレーンオペレーション (作成、読み取り、更新、削除) を実行できます。ドキュメントインターフェイスでは、データ型記述子 を指定する必要はありません。データ型は、データ自体のセマンティクスによって暗示されています。これらの AWS SDK には、JSON ドキュメントとネイティブの Amazon DynamoDB データ型とを簡単に変換するメソッドも用意されています。

Note

ドキュメントインターフェイスは、AWS SDKs for Java、.NET、Node.js、およびブラウザの JavaScript で使用できます。

次の Java プログラムは、AWS SDK for Java のドキュメントインターフェイスを使用します。プログラムが Music テーブルを表す Table オブジェクトを作成し、そのオブジェクトに GetItem の使用を依頼して、曲を取得します。その後プログラムが、曲がリリースされた年を出力します。

com.amazonaws.services.dynamodbv2.document.DynamoDB クラスは DynamoDB ドキュメントインターフェイスを実装します。低レベルクライアント (AmazonDynamoDB) のラッパーとして、DynamoDB がどのような役割を果たすかに注目してください。

```
package com.amazonaws.codesamples.gsg;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");
        System.out.println("The song was released in " + year);

    }
}
```

オブジェクト永続性インターフェイス

一部の AWS SDK には、データプレーンオペレーションを直接実行しない、オブジェクト永続性インターフェイスが用意されています。代わりに、Amazon DynamoDB のテーブルとインデックスの項目を表すオブジェクトを作成し、それらのオブジェクトのみとやり取りを行います。これにより、データベースを意識するのではなく、オブジェクトを中心にコードを作成していくことができます。

Note

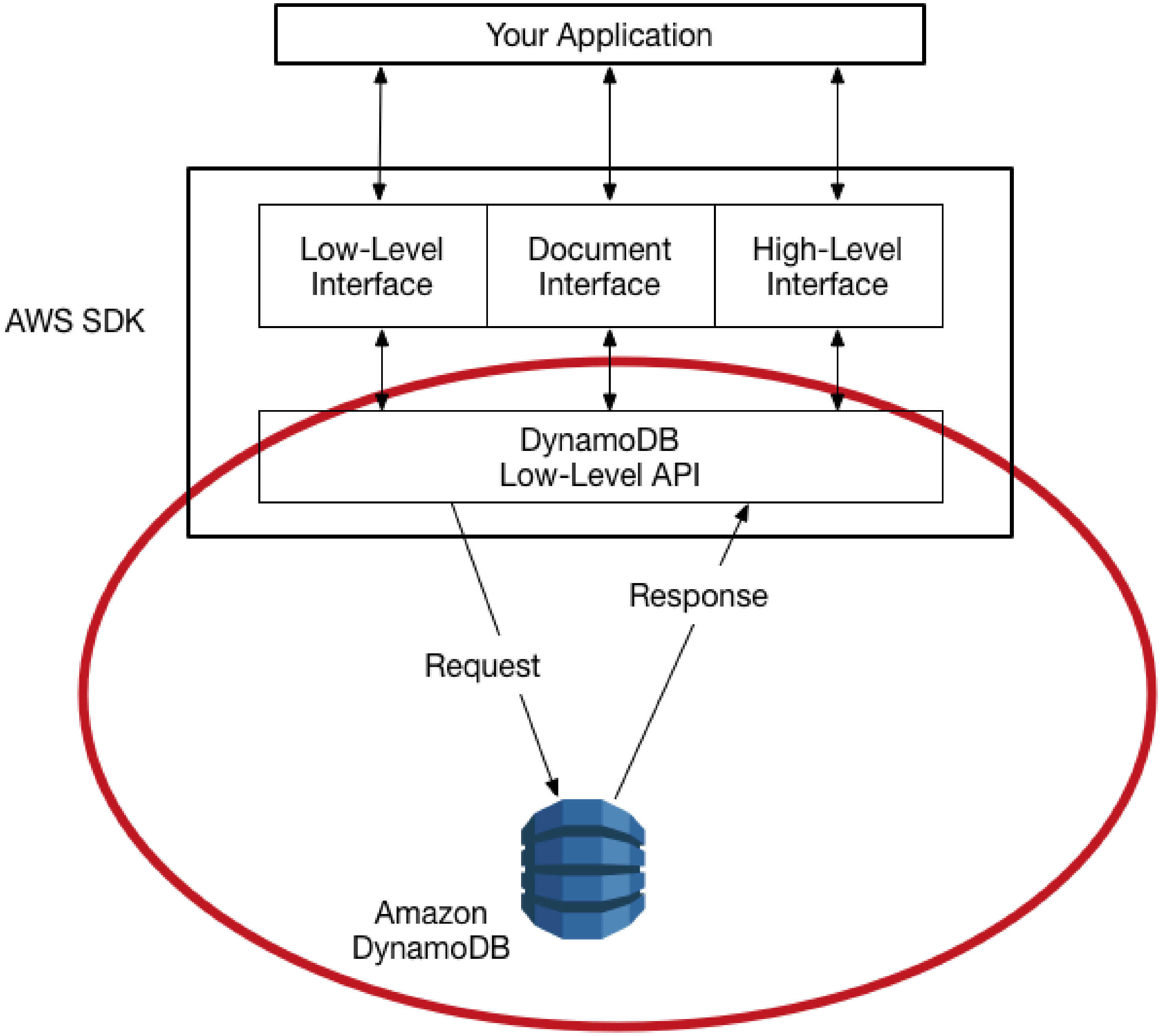
オブジェクト永続性インターフェイスは、AWS SDK for Java と for .NET で利用できます。詳細については、DynamoDB の「DynamoDB の上位レベルのプログラミングインターフェイス」を参照してください。

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;

/*
 * Before running this code example, create an Amazon DynamoDB table named Customer
 with these columns:
 *   - id - the id of the record that is the key. Be sure one of the id values is
 `id101`
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table. These
 values
 *                        need to be in the form of `YYYY-MM-DDTHH:mm:ssZ`, such as
 2022-07-11T00:00:00Z
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class EnhancedGetItem {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

        getItem(enhancedClient);
        ddb.close();
    }

    public static String getItem(DynamoDbEnhancedClient enhancedClient) {
        Customer result = null;
        try {
            DynamoDbTable<Customer> table = enhancedClient.table("Customer",
 TableSchema.fromBean(Customer.class));
            Key key = Key.builder()
                    .partitionValue("id101").sortValue("tred@noserver.com")
                    .build();

            // Get the item by using the key.
            result = table.getItem(
                    (GetItemEnhancedRequest.Builder requestBuilder) ->
 requestBuilder.key(key));
            System.out.println("******* The description value is " +
 result.getCustName());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return result.getCustName();
    }
}
```

DynamoDB 低レベル API

トピック



Amazon DynamoDB の低レベル API は、DynamoDB 用のプロトコルレベルのインターフェイスです。このレベルでは、すべての HTTP リクエストは、適切な形式で有効なデジタル署名がある必要があります。

AWS SDK は、低レベル DynamoDB API リクエストをユーザーに代わって作成し、DynamoDB からのレスポンスを処理します。これにより、低レベルの詳細ではなく、アプリケーションロジックに専念することができます。ただし、低レベル DynamoDB API の動作方法についての基本的な知識も役立ちます。

低レベル DynamoDB API の詳細については、Amazon DynamoDB API リファレンスを参照してください。

> Note
>
> DynamoDB Streams には、DynamoDB とは別に独自の低レベル API があり、AWS SDK で完全にサポートされています。
> 詳細については、「DynamoDB Streams の変更データキャプチャ」を参照してください。低レベル DynamoDB Streams API については、Amazon DynamoDB Streams API リファレンスを参照してください。

低レベル DynamoDB API は、ワイヤプロトコル形式として、JavaScript Object Notation (JSON) を使用しています。JSON では、データ値とデータ構造が両方同時にわかるように、データが階層で示されます。名前と値のペアは、`name:value` の形式で定義されます。データ階層は、名前と値のペアをブラケットで囲み、ネストする形で定義します。

DynamoDB は、ストレージ形式としてではなく、トランスポートプロトコルとしてのみ、JSON を使用しています。JSON を使用して AWS SDK が DynamoDB にデータを送信し、DynamoDB が JSON で応答します。DynamoDB は、JSON 形式でデータを永続的に保存しません。

> Note
>
> JSON の詳細については、`JSON.org` ウェブサイトで「JSON の入門」を参照してください。

リクエストの形式

DynamoDB 低レベル API は、HTTP(S) `POST` リクエストを入力として受け付けます。AWS SDK はこれらのリクエストを作成します。

`Pets` という名のテーブルに、`AnimalType` (パーティションキー)、`Name` (ソートキー) によって構成されるキースキーマがあるとします。これらの属性はいずれも、`string` 型になります。`Pets` から項目を取得するために、AWS SDK は次のリクエストを作成します。

```
POST / HTTP/1.1
Host: dynamodb.<region>.<domain>;
```

```
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Pets",
    "Key": {
        "AnimalType": {"S": "Dog"},
        "Name": {"S": "Fido"}
    }
}
```

このリクエストに関して以下の点に注意してください。

- `Authorization` ヘッダーには、DynamoDB がリクエストを認証するのに必要な情報が含まれています。詳細は、「Amazon Web Services 全般のリファレンス」の「AWS API リクエストへの署名」と「署名バージョン 4 の署名プロセス」を参照してください。
- `X-Amz-Target` ヘッダーには、DynamoDB オペレーションの名前である `GetItem` が含まれます 。(これは、低レベル API バージョンと共に示されます。この場合は `20120810` となります。)
- リクエストのペイロード (本文) には、JSON 形式で、オペレーションのパラメーターが含まれます。`GetItem` オペレーションでは、パラメーターは `TableName` と `Key` です。

レスポンスの形式

リクエストを受け取ったら、DynamoDB が処理しレスポンスを返します。前の例に示したように、HTTP レスポンスペイロードには、オペレーションからの結果が含まれます。

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
    "Item": {
        "Age": {"N": "8"},
        "Colors": {
            "L": [
                {"S": "White"},
                {"S": "Brown"},
                {"S": "Black"}
            ]
        },
        "Name": {"S": "Fido"},
        "Vaccinations": {
            "M": {
                "Rabies": {
                    "L": [
                        {"S": "2009-03-17"},
                        {"S": "2011-09-21"},
                        {"S": "2014-07-08"}
                    ]
                },
                "Distemper": {"S": "2015-10-13"}
            }
        },
        "Breed": {"S": "Beagle"},
        "AnimalType": {"S": "Dog"}
    }
}
```

この時点で、AWS SDK は、さらに処理するためにアプリケーションに応答データを返します。

> Note
>
> リクエストを処理できない場合は、DynamoDB より HTTP エラーコードとメッセージが返ります。AWS SDK は、これらを例外形式でアプリケーションに伝達します。詳細については、「DynamoDB でのエラー処理」を参照してください。

データ型記述子

低レベル DynamoDB API のプロトコルは、各属性がデータ型記述子に伴われる必要があります。データ型記述子は、各属性を解釈する方法を DynamoDB に伝えるトークンです。

リクエストの形式とレスポンスの形式には、データ型記述子が使用されている例が示されています。GetItem リクエストでは、S キースキーマ属性 (Pets と AnimalType) に Name を string 型

で指定します。GetItem レスポンスには、string (S)、number (N)、map (M)、list (L) 型の属性を持つ、Pets 項目が含まれます。

DynamoDB データ型記述子の一覧を次に示します。

- **S** — 文字列
- **N** — 数値
- **B** — バイナリ
- **BOOL** — ブール
- **NULL** — Null
- **M** — マップ
- **L** — リスト
- **SS** — 文字列セット
- **NS** — 数値セット
- **BS** — バイナリセット

> Note
>
> DynamoDB データ型の詳細な説明については、「データ型」を参照してください。

数値データ

プログラミング言語により、提供される JSON のサポートのレベルが異なります。場合によっては、JSONドキュメントを検証し解析するにあたり、サードパーティーのライブラリを使用することもできます。

JSON Number 型に基づいて構築されたサードパーティーライブラリもあり、int や long、double など独自の型を提供しています。ただし、他のデータ型に正確にマッピングされない DynamoDB のネイティブ数値データ型が使用されるため、このようなデータ型の区別が競合の原因になる可能性があります。加えて、多くの JSON ライブラリでは固定精度の数値は処理されず、小数点を含む数字列は自動的に倍精度浮動小数点データ型であると推定されます。

これらの問題を解決するために、DynamoDB ではデータ損失のない単一の数値型が用意されています。誤って倍精度の値に暗黙的変換が行われないように、DynamoDB では数値のデータ転送には文

字列が使用されます。この方法によって、01、2、03 などの値を適切な順序で配置するなど、適切な並べ替えセマンティクスを維持しながら、柔軟に属性値を更新することが可能になります。

数値の精度がアプリケーションにとって重要な場合は、数値を文字列に変換してから、DynamoDB に渡します。

バイナリデータ

DynamoDB ではバイナリ属性がサポートされています。ただし JSON では、ネイティブではバイナリデータのエンコードがサポートされていません。リクエストでバイナリデータを送信するには、base64 形式でエンコードする必要があります。DynamoDB はリクエストを受け取ると、base64 データをバイナリに復号します。

DynamoDB で使用される base64 エンコーディングスキームは、Internet Engineering Task Force (IETF) ウェブサイトの「RFC 4648」に記載されています。

DynamoDB でのエラー処理

このセクションでは、ランタイムエラーとその処理方法について説明します。同時に、Amazon DynamoDB に特有のエラーメッセージとコードについて説明します。AWS のすべてのサービスに共通するエラーのリストについては、「アクセス管理」を参照してください。

トピック

エラーコンポーネント

プログラムがリクエストを送信すると、DynamoDB はその処理を実行するよう試みます。リクエストが成功した場合、DynamoDB はそのオペレーションが出力した結果とともに、 HTTP の成功ステータスコード (`200 OK`) を返します。

リクエストが正常に行われなかった場合、DynamoDB はエラーを返します。それぞれのエラーには、次の三つのコンポーネントがあります:

- HTTP ステータスコード (400 など)。
- 例外の名前 (ResourceNotFoundException など)。
- エラーメッセージ (Requested resource not found: Table: *tablename* not found など)。

AWS SDK によりアプリケーションにエラーが伝達されるため、適切なアクションを実行できます。たとえば、Java プログラムでは、 try-catch を処理する ResourceNotFoundException ロジックを記述できます。

AWS SDK を使用していない場合は、DynamoDB からの低レベルのレスポンスの内容を、ユーザー側で解析する必要があります。以下に、そのようなレスポンスの例を示します。

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 240
Date: Thu, 15 Mar 2012 23:56:23 GMT

{"__type":"com.amazonaws.dynamodb.v20120810#ResourceNotFoundException",
"message":"Requested resource not found: Table: tablename not found"}
```

トランザクションエラー

トランザクションエラーの詳細については、「DynamoDB でのトランザクション競合の処理」を参照してください。

エラーメッセージおよびコード

以下に、DynamoDB によって返される例外のリストを、HTTP ステータスコードでグループ分けして示します。再試行してもいいですかがはいであれば、同じリクエストを再度送信できます。再試行してもいいですかがいいえであれば、新しいリクエストを送信する前に、クライアント側で問題を修正する必要があります。

HTTP ステータスコード 400

HTTP ステータスコード400は、認証の失敗、必須パラメータの欠落、またはテーブルにプロビジョニングされているスループットの超過などのリクエストに関連した問題があることを示しています。リクエストを再度送信する前に、アプリケーションで問題を修正する必要があります。

AccessDeniedException

メッセージ: アクセスが拒否されました。

クライアントがリクエストに正しく署名しませんでした。AWS SDK を使用している場合には、リクエストへの署名が自動的に実行されます。使用していない場合は、「AWS 全般のリファレンス」の「署名バージョン 4 の署名プロセス」に進んでください。

再試行してもいいですか。いいえ

ConditionalCheckFailedException

メッセージ: 条件付きリクエストが失敗しました。

false と評価された条件を指定しました。たとえば、項目に条件付き更新を実行しようとしたかもしれませんが、属性の実際の値は、条件の予期される値と一致しませんでした。

再試行してもいいですか。いいえ

IncompleteSignatureException

メッセージ: リクエストの署名が AWS スタンダードに適合しません。

リクエストの署名に、必要なすべての要素が含まれていませんでした。AWS SDK を使用している場合には、リクエストへの署名が自動的に実行されます。使用していない場合は、「AWS 全般のリファレンス」の「署名バージョン 4 の署名プロセス」に進んでください。

再試行してもいいですか。いいえ

ItemCollectionSizeLimitExceededException

メッセージ: コレクションサイズが超過しました。

ローカルセカンダリインデックスがあるテーブルにおいて、同じパーティションキー値を持つ項目のグループが占めるサイズが、最大 10 GB の上限を超過しました。項目コレクションの詳細については、「ローカルセカンダリインデックス内の項目コレクション」を参照してください。

再試行してもいいですか。はい

LimitExceededException

メッセージ: 特定の受信者に対するオペレーションが多すぎます。

同時オペレーションのコントロールプレーンが多すぎます。`CREATING`、`DELETING`、または `UPDATING` の状態のテーブルやインデックスの累積数が、500 を超えることはできません。

再試行してもいいですか。はい

MissingAuthenticationTokenException

メッセージ: リクエストには、有効な (登録済みメンバー) AWS アクセスキー ID が含まれている必要があります。

リクエストに、必要な認証ヘッダーが含まれていないか、または正しい形式ではありません。「」を参照してくださいDynamoDB 低レベル API

再試行してもいいですか。いいえ

ProvisionedThroughputExceededException

メッセージ: 1 つのテーブルまたは 1 つ以上のグローバルセカンダリインデックスで、プロビジョン済みスループットが、許容されている最大量を超えました。プロビジョンドスループットと消費スループットのパフォーマンスメトリクスを表示するには、「Amazon CloudWatch コンソール」を参照してください。

例: リクエストの頻度が多すぎます。DynamoDB 用の AWS SDK では、この例外を受け取ったリクエストを自動的に再試行します。リクエストは最終的に成功しますが、再試行キューが大きすぎて終了しない場合もあります。エラーの再試行とエクスポネンシャルバックオフ を使用して、リクエストの頻度を少なくします。

再試行してもいいですか。はい

RequestLimitExceeded

メッセージ: スループットがアカウントの現在のスループット制限を超えています。制限の引き上げをリクエストするには、https://aws.amazon.com/support で AWS サポートにお問い合わせください。

例: オンデマンドリクエストの速度が、許容されているアカウントスループットを超えていて、これ以上テーブルをスケールできません。

再試行してもいいですか。はい

ResourceInUseException

メッセージ: 変更しようとしているリソースは使用中です。

例: 既存のテーブルを再作成しようとしたか、CREATING 状態にあるテーブルを削除しようとしました。

再試行してもいいですか。いいえ

ResourceNotFoundException

メッセージ: リクエストされたリソースは存在しません。

例: リクエストされたテーブルが存在しないか、ごく初期の CREATING 状態にあります。

再試行してもいいですか。いいえ

ThrottlingException

メッセージ: リクエストの速度が、許容されているスループットを超えています。

この例外は、AmazonServiceException レスポンスとして、TROTTLING_EXCEPTION ステータスコードとともに返されます。この例外は、コントロールプレーン API オペレーションの実行が速すぎる場合に返される可能性があります。

オンデマンドモードを使用するテーブルの場合、リクエストレートが高すぎると、データプレーン API オペレーションでこの例外が返されることがあります。オンデマンドスケーリングの詳細については、「初期スループットとスケーリングのプロパティ」を参照してください。

再試行してもいいですか。はい

UnrecognizedClientException

メッセージ: アクセスキー ID またはセキュリティトークンが無効です。

リクエスト署名が間違っています。最も可能性の高い原因は、AWS アクセスキー ID またはシークレットキーが無効であることです。

再試行してもいいですか。はい

ValidationException

メッセージ: 発生した特定のエラーにより異なります

このエラーは、必須パラメータが指定されていない、値が範囲外である、データ型が一致しない、などいくつかの理由で発生します。エラーメッセージに、エラーを引き起こしたリクエストの特定部分に関する詳細が含まれています。

再試行してもいいですか。いいえ

HTTP ステータスコード 5xx

HTTP ステータスコード `5xx` は、AWS で解決する必要のある問題を示しています。これは一時的なエラーかもしれず、その場合はリクエストを再試行することで成功する場合があります。それ以外の場合、サービスにオペレーション上の問題があるかどうかを確認するために、AWS Service Health Dashboard を参照してください。

詳細については、「Amazon DynamoDB の HTTP 5xx エラーを解決する方法を教えてください。」を参照してください。

InternalServerError (HTTP 500)

DynamoDB はリクエストを処理できませんでした。

再試行してもいいですか。はい

> **Note**
>
> 項目の操作中に内部サーバーエラーが発生することがあります。これはテーブルの存続期間中に発生すると予想されます。失敗したリクエストは速やかに再試行できます。
> 書き込みオペレーションでステータスコード 500 を受け取った場合、オペレーションは成功した可能性も、失敗した可能性もあります。書き込みオペレーションが `TransactWriteItem` リクエストである場合、オペレーションを再試行できる状態です。書き込みオペレーションが単一項目の書き込み要求 (`PutItem`、`UpdateItem`、または `DeleteItem` など) である場合、アプリケーションはオペレーションを再試行する前に項目の状態を読み取り、条件式 を使用して、前のオペレーションが成功したか失敗したかにかかわらず、再試行後も項目が正しい状態のままであることを確認します。冪等性が書き込みオペレーションの要件である場合は、`TransactWriteItem` を使用して

ください。これは、同じアクションを実行する複数の試行を明確にするために自動的に ClientRequestToken を指定することにより、冪等リクエストをサポートします。

ServiceUnavailable (HTTP 503)

DynamoDB は現在利用できません。(これは一時的な状態です。)

再試行してもいいですか。はい

アプリケーションのエラー処理

アプリケーションをスムーズに実行するには、エラーを見つけ、エラーに対応するロジックを組み込む必要があります。一般的な方法には、try-catchブロックやif-thenステートメントの使用などがあります。

AWS SDK は独自に再試行とエラーチェックを実行します。AWS SDK の使用中にエラーが発生した場合は、エラーコードと説明が問題のトラブルシューティングに役立ちます。

また、レスポンスにRequest IDが表示されます。Request ID は、AWS サポートを使用して問題を診断することが必要な場合に便利です。

エラーの再試行とエクスポネンシャルバックオフ

特定のリクエストの処理中には、DNS サーバー、スイッチ、ロードバランサーなど、ネットワーク上のさまざまなコンポーネントが原因でエラーが発生する可能性があります。ネットワーク環境でこれらのエラー応答を処理する通常の方法は、クライアントアプリケーションで再試行を実装することです。この手法により、アプリケーションの信頼性が向上します。

各 AWS SDK は、自動的に再試行ロジックを実装しています。必要に応じて再試行パラメータを変更できます。たとえば、エラーが発生したら再試行が許可されない、Fail-Fast 方式を要求する Java アプリケーションについて考えてみましょう。AWS SDK for Java では、ClientConfiguration クラスを使用して maxErrorRetry の値を 0 に設定することで、再試行を無効にできます。詳細については、AWS SDK ドキュメントのご使用のプログラミング言語の項目を参照してください。

AWS SDK を使用していない場合は、サーバーエラー (5xx) を受け取る元のリクエストを再試行する必要があります。ただし、クライアントエラー (4xx、ThrottlingException または ProvisionedThroughputExceededException 以外) は、再試行する前にリクエスト自体を修正して問題を解決する必要があることを示しています。

簡単な再試行に加えて、各 AWS SDK はエクスポネンシャルバックオフアルゴリズムを実装し、フロー制御を改善します。エクスポネンシャルバックオフは、再試行間の待機時間を累進的に長くして、連続的なエラー応答を受信するという概念に基づいています。たとえば、1 回目の再試行の前に最大 50 ミリ秒、2 回目の前に最大 100 ミリ秒、3 回目の前に最大 200 ミリ秒のようになります。ただし 1 分を経過してもリクエストが成功しない場合は、問題の原因はリクエストの速度ではなく、プロビジョニングしたスループットをリクエストのサイズが超えたためである可能性があります。1 分程度で再試行が停止するように最大回数を設定します。リクエストが失敗した場合は、プロビジョニングされたスループットのオプションを調べてください。

> Note
>
> AWS SDK には自動再試行ロジックとエクスポネンシャルバックオフが実装されています。

ほとんどのエクスポネンシャルバックオフアルゴリズムは、連続した衝突を防ぐためにジッター (ランダム化された遅延) を使用します。この場合は、そうした衝突を回避しようとしていないので、乱数を使用する必要はありません。ただし、同時クライアントを使用する場合、ジッターはリクエストをより速く成功させるのに役立ちます。詳細については、「エクスポネンシャルバックオフとジッター」に関するブログ投稿を参照してください。

バッチオペレーションとエラー処理

DynamoDB の低レベル API では、読み込み/書き込みのバッチオペレーションがサポートされています。`BatchGetItem` は 1 つ以上のテーブルからの項目の読み込みを処理し、`BatchWriteItem` は、1 つ以上のテーブルでの項目の入力や削除を処理します。これらのバッチオペレーションは、DynamoDB のバッチではない他のオペレーションのラッパーとして実装されています。つまり、`BatchGetItem` は、バッチの項目ごとに `GetItem` を一度呼び出します。同様に、`BatchWriteItem` は、`DeleteItem` または `PutItem` を必要に応じてバッチの項目ごとに呼び出します。

バッチオペレーションではバッチの個々のリクエストのエラーが許容されます。たとえば、5 つの項目を読み込む `BatchGetItem` リクエストの場合を考えます。基礎となる `GetItem` リクエストの一部が失敗した場合でも、`BatchGetItem` オペレーション全体が失敗することはありません。ただし、5 つのオペレーションにすべて失敗する場合は、`BatchGetItem` 全体が失敗します。

バッチオペレーションでは、失敗した個々のリクエストについて情報が返されるため、問題を診断し、オペレーションを再試行できます。`BatchGetItem` の場合、問題のあるテーブルとプライマリキーがレスポンスの `UnprocessedKeys` 値で返されます。`BatchWriteItem` の場合、同様の情報は `UnprocessedItems` で返されます。

読み込みまたは書き込みの失敗の原因として最も可能性が高いのは、スロットリングです。`BatchGetItem` の場合、バッチリクエストの 1 つ以上のテーブルに、オペレーションをサポートするための十分なプロビジョンド読み込みキャパシティーがなくなります。`BatchWriteItem` の場合、1 つ以上のテーブルに、十分なプロビジョンド書き込みキャパシティーがなくなります。

DynamoDB によって未処理の項目が返された場合は、それらの項目に対してバッチオペレーションを再試行する必要があります。ただし、エクスポネンシャルバックオフアルゴリズムを使用することを強くお勧めします。すぐにバッチオペレーションを再試行した場合、基礎となる読み込みまたは書き込みリクエストはやはり、個々のテーブルに対する帯域幅調整により失敗することがあります。エクスポネンシャルバックオフアルゴリズムを使用してバッチオペレーションを遅らせた場合は、バッチの個々のリクエストが成功する可能性がはるかに高くなります。

DynamoDB の上位レベルのプログラミングインターフェイス

AWS SDK は、Amazon DynamoDB を操作するための下位レベルのインターフェイスをアプリケーションに提供します。これらのクライアント側のクラスとメソッドは、下位レベルの DynamoDB API と直接対応しています。しかし、これは多くのデベロッパーにとって、複雑なデータ型をデータベーステーブル内の項目にマップする必要がある場合の接続性の悪さ、つまりインピーダンスの不整合の原因となります。下位レベルのデータベースインタフェースを使用するデベロッパーは、オブジェクトデータをデータベーステーブルに対して読み書きするためのメソッドを記述する必要があります。オブジェクトタイプとデータベーステーブルの組み合わせに、個別に対応するための余分なコードが、圧倒的な量になることもあります。

AWS SDK for Java および for .NET では、開発業務を簡素化するために、高度に抽象化されたインターフェイスが用意されています。DynamoDB の上位レベルのインターフェイスを使用すると、プログラム内のオブジェクトと、それらのオブジェクトのデータを格納するためのデータベーステーブル間の関係を定義できます。このマッピングを定義した後、`save`、`load`、`delete` などの単純なオブジェクトメソッドを呼び出すと、ユーザーに代わって、基盤となる下位レベルの DynamoDB オペレーションの呼び出しが自動化されます。これにより、データベースを意識するのではなく、オブジェクトを中心にコードを作成していくことができます。

DynamoDB の上位レベルのプログラミングインターフェイスは、Java および .NET 用の AWS SDK で使用できます。

Java

- Java 1.x: DynamoDBMapper
- Java 2.x: DynamoDB 拡張クライアント

.NET

- .NET ドキュメントモデル
- .NET: オブジェクト永続性モデル

Java 1.x: DynamoDBMapper

AWS SDK for Java には DynamoDBMapper クラスが用意されているため、クライアント側のクラスを Amazon DynamoDB テーブルにマッピングできます。DynamoDBMapper を使用するには、DynamoDB テーブル内の項目と、それが対応するコード内のオブジェクトインスタンスの間での関係性を定義します。DynamoDBMapper クラスを使用すると、項目に対する作成、読み込み、更新、削除 (CRUD) の各オペレーションの実行、およびテーブルに対するクエリやスキャンを行うことができます。

トピック

- DynamoDB Mapper for Java でサポートされるデータ型
- DynamoDB 用の Java アノテーション
- DynamoDBMapper クラス
- DynamoDBMapper のオプションの構成設定
- バージョン番号を使用した楽観的ロック
- 任意データのマッピング
- DynamoDBMapper の例

Note

DynamoDBMapperクラスでは、テーブルを作成、更新、または削除することはできません。これらのタスクを実行するには、代わりに SDK for Java の下位レベルインターフェイスを使用します。詳細については、「」を参照してくださいJava での DynamoDB テーブルの使用

SDK for Java には、クラスをテーブルにマッピングするための、一連のアノテーションタイプが用意されています。たとえば、パーティションキーとして ProductCatalog を含む Id テーブルがあるとします。

```
ProductCatalog(Id, ...)
```

次の Java コードに示すように、クライアントアプリケーション内のクラスを `ProductCatalog` テーブルにマッピングすることができます。このコードでは、`CatalogItem` という名前の Plain Old Java Object (POJO) を定義しています。このオブジェクトは、アノテーションを使用して、オブジェクトフィールドを DynamoDB の属性名にマッピングします。

Example

```
package com.amazonaws.codesamples;

import java.util.Set;

import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIgnore;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) {this.id = id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN; }

    @DynamoDBAttribute(attributeName="Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp; }
    public void setSomeProp(String someProp) { this.someProp = someProp; }
}
```

前述のコードでは、@DynamoDBTable 注釈によって、CatalogItem クラスが ProductCatalog テーブルにマッピングされています。個々のクラスインスタンスは、テーブル内の項目として格納できます。クラス定義では、@DynamoDBHashKey 注釈によって Id プロパティがプライマリキーにマッピングされます。

デフォルトでは、クラスプロパティはテーブル内の同じ名前属性にマッピングされます。プロパティ Title および ISBN は、テーブル内の同じ名前属性にマッピングされます。

DynamoDB の属性名がクラスで宣言されたプロパティ名と一致する場合、@DynamoDBAttribute アノテーションの使用はオプションです。これらの名前が異なる場合には、attributeName パラメータを指定しながらこのアノテーションを使用し、プロパティが DynamoDB のどの属性に対応しているかを指定します。

前述の例では、各プロパティに @DynamoDBAttribute 注釈を追加することで、プロパティ名が「DynamoDB でのコード例用のテーブルの作成とデータのロード」で作成したテーブルに確実に一致し、このガイド内の他のコード例で使用されている属性名との整合性がとられています。

クラス定義には、テーブル内のどの属性にもマッピングされないプロパティを含めることもできます。これらのプロパティを特定するには、@DynamoDBIgnore 注釈を追加します。前述の例では、SomeProp プロパティが @DynamoDBIgnore 注釈によってマーキングされています。CatalogItem インスタンスをテーブルにアップロードしたとき、DynamoDBMapper インスタンスに SomeProp プロパティは追加されません。また、このマッパーは、テーブルから項目を取り出すときにこの属性を返しません。

マッピングクラスを定義したら、DynamoDBMapper メソッドを使用して、そのクラスのインスタンスを Catalog テーブルの対応する項目に書き込むことができます。次のサンプルコードでは、この技術を示しています。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapper mapper = new DynamoDBMapper(client);

CatalogItem item = new CatalogItem();
item.setId(102);
item.setTitle("Book 102 Title");
item.setISBN("222-2222222222");
item.setBookAuthors(new HashSet<String>(Arrays.asList("Author 1", "Author 2")));
item.setSomeProp("Test");

mapper.save(item);
```

次のサンプルコードでは、項目を取得し、その属性の一部にアクセスする方法を示します。

```
CatalogItem partitionKey = new CatalogItem();

partitionKey.setId(102);
DynamoDBQueryExpression<CatalogItem> queryExpression = new
 DynamoDBQueryExpression<CatalogItem>()
    .withHashKeyValues(partitionKey);

List<CatalogItem> itemList = mapper.query(CatalogItem.class, queryExpression);

for (int i = 0; i < itemList.size(); i++) {
    System.out.println(itemList.get(i).getTitle());
    System.out.println(itemList.get(i).getBookAuthors());
}
```

DynamoDBMapper は、Java 内で DynamoDB データを操作するための直観的で自然な方法を提供します。また、オプティミスティックロック、ACID トランザクション、自動生成されるパーティションキーとソートキーの値、オブジェクトのバージョニングなど、複数の組み込み機能があります。

DynamoDB Mapper for Java でサポートされるデータ型

このセクションでは、Amazon DynamoDB でサポートされているプリミティブな Java データ型、コレクション、および任意のデータ型について説明します。

Amazon DynamoDB では、次のプリミティブな Java データ型とプリミティブなラッパークラスがサポートされています。

- String
- Boolean, boolean
- Byte, byte
- Date (ISO_8601 ミリ秒精度文字列、UTC に転換)
- Calendar (ISO_8601 ミリ秒精度文字列、UTC に転換)
- Long, long

- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`
- `BigInteger`

> Note
>
> - DynamoDB での命名規則およびサポートされている各データ型の詳細については、「Amazon DynamoDB でサポートされるデータ型と命名規則」を参照してください。
> - 空のバイナリ値は、DynamoDBMapper でサポートされます。
> - 空の文字列値は、AWS SDK for Java 2.x でサポートされます。
>
>   AWS SDK for Java 1.x では、DynamoDBMapper による空の文字列属性値の読み取りをサポートしています。ただし、空の文字列属性値の書き込みは、その属性がリクエストから削除されるため実行されません。

DynamoDB では、Java の Set、List、および Map コレクションタイプがサポートされています。次の表に、これらの Java 型が DynamoDB 型にどのようにマッピングされるかを示します。

Java 型	DynamoDB 型
すべての数値型	N (数値型)
文字列	S (文字列型)
Boolean	BOOL (ブール型)、0 または 1。
ByteBuffer	B (バイナリ型)
日付	S (文字列型)。Date の値は、ISO-8601 形式の文字列として格納されます。
Set コレクション型	SS (文字列セット) 型、NS (数値セット) 型、または BS (バイナリセット) 型。

DynamoDBTypeConverter インターフェイスでは、独自の任意データ型を、DynamoDB でネイティブにサポートされているデータ型にマッピングすることができます。詳細については、「」を参照してください任意データのマッピング

DynamoDB 用の Java アノテーション

このセクションでは、クラスとプロパティを Amazon DynamoDB のテーブルと属性にマッピングするためのアノテーションについて説明します。

関連する Javadoc ドキュメントについては、AWS SDK for Java API リファレンスの「アノテーションタイプの概要」を参照してください。

> Note
>
> 次の注釈では、DynamoDBTable と DynamoDBHashKey だけが必須です。

トピック

- DynamoDBAttribute
- DynamoDBAutoGeneratedKey
- DynamoDBAutoGeneratedTimestamp
- DynamoDBDocument
- DynamoDBHashKey
- DynamoDBIgnore
- DynamoDBIndexHashKey
- DynamoDBIndexRangeKey
- DynamoDBRangeKey
- DynamoDBTable
- DynamoDBTypeConverted
- DynamoDBTyped
- DynamoDBVersionAttribute

DynamoDBAttribute

テーブルの属性にプロパティをマッピングします。デフォルトでは、各クラスのプロパティが、同じ名前の項目属性にマッピングされます。ただし名前が同じでない場合は、この注釈を

使用して属性にプロパティをマッピングすることができます。次の Java コードスニペットでは、DynamoDBAttribute によって、BookAuthors プロパティがテーブル内の 属性名にマッピングされています。Authors

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

DynamoDBMapper では、テーブルにオブジェクトを保存する際に、 を属性名として使用しています。Authors

DynamoDBAutoGeneratedKey

パーティションキーまたはソートキーのプロパティは、自動生成済みとしてマーキングされます。DynamoDBMapper では、これらの属性を保存するときにランダムな UUID が生成されます。String プロパティには、自動生成済みのキーとしてマーキングできます。

次の例は、自動生成されたパラメータを使用する方法を示しています。

```
@DynamoDBTable(tableName="AutoGeneratedKeysExample")
public class AutoGeneratedKeys {
    private String id;
    private String payload;

    @DynamoDBHashKey(attributeName = "Id")
    @DynamoDBAutoGeneratedKey
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    @DynamoDBAttribute(attributeName="payload")
    public String getPayload() { return this.payload; }
    public void setPayload(String payload) { this.payload = payload; }

    public static void saveItem() {
        AutoGeneratedKeys obj = new AutoGeneratedKeys();
        obj.setPayload("abc123");

        // id field is null at this point
        DynamoDBMapper mapper = new DynamoDBMapper(dynamoDBClient);
        mapper.save(obj);
        System.out.println("Object was saved with id " + obj.getId());
    }
}
```

DynamoDBAutoGeneratedTimestamp

タイムスタンプを自動生成します。

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

オプションで、戦略属性を指定して自動生成戦略を定義できます。デフォルト: `ALWAYS`。

DynamoDBDocument

Amazon DynamoDB ドキュメントにあるように、クラスは連番で作成できます。

例えば、JSON ドキュメントを Map 型 (`M`) の DynamoDB 属性にマッピングしたい場合を考えます。次のサンプルコードでは、マップ型の入れ子の属性 (Pictures) を含む項目を定義します。

```
public class ProductCatalogItem {

    private Integer id;  //partition key
    private Pictures pictures;
    /* ...other attributes omitted... */

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id;}
    public void setId(Integer id) {this.id = id;}

    @DynamoDBAttribute(attributeName="Pictures")
    public Pictures getPictures() { return pictures;}
    public void setPictures(Pictures pictures) {this.pictures = pictures;}

    // Additional properties go here.

    @DynamoDBDocument
    public static class Pictures {
        private String frontView;
        private String rearView;
        private String sideView;

        @DynamoDBAttribute(attributeName = "FrontView")
        public String getFrontView() { return frontView; }
        public void setFrontView(String frontView) { this.frontView = frontView; }

        @DynamoDBAttribute(attributeName = "RearView")
        public String getRearView() { return rearView; }
        public void setRearView(String rearView) { this.rearView = rearView; }

        @DynamoDBAttribute(attributeName = "SideView")
        public String getSideView() { return sideView; }
        public void setSideView(String sideView) { this.sideView = sideView; }

     }
}
```

以下の例に示されているように、ProductCatalog と合わせて、新しい Pictures 項目を保存することができます。

```
ProductCatalogItem item = new ProductCatalogItem();

Pictures pix = new Pictures();
pix.setFrontView("http://example.com/products/123_front.jpg");
pix.setRearView("http://example.com/products/123_rear.jpg");
pix.setSideView("http://example.com/products/123_left_side.jpg");
item.setPictures(pix);

item.setId(123);

mapper.save(item);
```

その結果、ProductCatalog 項目は次のようになります (JSON 形式)。

```
{
  "Id" : 123
  "Pictures" : {
    "SideView" : "http://example.com/products/123_left_side.jpg",
    "RearView" : "http://example.com/products/123_rear.jpg",
    "FrontView" : "http://example.com/products/123_front.jpg"
  }
}
```

DynamoDBHashKey

テーブルのパーティションキーにクラスプロパティをマッピングします。このプロパティは、スカラー文字列型、数値型、バイナリ型のいずれかである必要があります。コレクション型は使用できません。

`ProductCatalog` がプライマリキーである `Id` テーブルがあるとします。次の Java コードでは、`CatalogItem` クラスを定義し、その `Id` プロパティを `ProductCatalog` タグを使用して `@DynamoDBHashKey` テーブルのプライマリキーにマッピングしています。

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {
    private Integer Id;
   @DynamoDBHashKey(attributeName="Id")
   public Integer getId() {
        return Id;
   }
   public void setId(Integer Id) {
        this.Id = Id;
   }
   // Additional properties go here.
}
```

DynamoDBIgnore

`DynamoDBMapper` インスタンスに対して、関連するプロパティを無視するように指示します。テーブルにデータを保存する場合、`DynamoDBMapper` ではこのプロパティがテーブルに保存されません。

モデリングされていないプロパティの getter メソッドまたはクラスフィールドに適用されます。注釈がクラスフィールドに直接適用されている場合、対応する getter および setter が同じクラスで宣言される必要があります。

DynamoDBIndexHashKey

グローバルセカンダリインデックスのパーティションキーに、クラスプロパティをマッピングします。このプロパティは、スカラー文字列型、数値型、バイナリ型のいずれかである必要があります。コレクション型は使用できません。

グローバルセカンダリインデックスを `Query` したい場合に、このアノテーションを使用します。インデックス名（`globalSecondaryIndexName`）を指定する必要があります。クラスプロパ

ティの名前がインデックスのパーティションキーと異なる場合、そのインデックス属性の名前 (attributeName) も指定する必要があります。

DynamoDBIndexRangeKey

グローバルセカンダリインデックスまたはローカルセカンダリインデックスのソートキーにクラスプロパティをマッピングします。このプロパティは、スカラー文字列型、数値型、バイナリ型のいずれかである必要があります。コレクション型は使用できません。

このアノテーションは、ローカルセカンダリインデックスもしくはグローバルセカンダリインデックスを Query し、その結果をインデックスのソートキーを使用して絞り込む必要がある場合に使用します。インデックス名 (globalSecondaryIndexName または localSecondaryIndexName) を指定する必要があります。クラスプロパティの名前がインデックスのソートキーと異なる場合、そのインデックス属性の名前 (attributeName) も指定する必要があります。

DynamoDBRangeKey

テーブルのソートキーにクラスプロパティをマッピングします。このプロパティは、スカラー文字列型、数値型、バイナリ型のいずれかである必要があります。コレクション型は使用できません。

プライマリキーが複合の場合 (パーティションキーとソートキー)、このタグを使用して、クラスフィールドをソートキーにマッピングできます。たとえば、フォーラムスレッドからの返信を格納する Reply テーブルがあるとします。各スレッドには多数の返信がある可能性があります。したがってこのテーブルのプライマリキーは、ThreadId と ReplyDateTime の両方になります。パーティションキーは ThreadId で、ソートキーは ReplyDateTime です。

たとえば次の Java コードでは、Reply クラスを定義して、Reply テーブルにマッピングしています。ここでは、@DynamoDBHashKey タグと @DynamoDBRangeKey タグの両方を使用して、プライマリキーにマッピングされるクラスプロパティを識別しています。

```
@DynamoDBTable(tableName="Reply")
public class Reply {
    private Integer id;
    private String replyDateTime;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    @DynamoDBRangeKey(attributeName="ReplyDateTime")
    public String getReplyDateTime() { return replyDateTime; }
    public void setReplyDateTime(String replyDateTime) { this.replyDateTime =
 replyDateTime; }

   // Additional properties go here.
}
```

DynamoDBTable

DynamoDB 内でターゲットテーブルを識別します。例えば、次の Java コードでは Developer クラスを定義して、DynamoDB の People テーブルにマッピングしています。

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

@DynamoDBTable 注釈は継承できます。Developer クラスから継承された新しいクラスも、People テーブルにマッピングされます。たとえば、Lead クラスから継承された Developer クラスを作成したとします。Developer クラスを People テーブルにマッピングしたことで、Lead クラスオブジェクトも同じテーブルに格納されます。

@DynamoDBTable もオーバーライドできます。デフォルトで Developer クラスから継承された新しいクラスは、同じ People テーブルにマッピングされます。ただし、このデフォルトのマッピングはオーバーライドできます。たとえば、Developer クラスから継承したクラスを作成した場合には、次の Java サンプルコードに示すように、@DynamoDBTable 注釈を追加することで、別のテーブルに明示的にマッピングできます。

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDBTypeConverted

カスタム型コンバーターの使用時にプロパティをマークする注釈を使用できます。DynamoDBTypeConverter に追加のプロパティを渡すために、ユーザーが定義した注釈でコメントすることもできます。

DynamoDBTypeConverter インターフェイスでは、独自の任意データ型を、DynamoDB でネイティブにサポートされているデータ型にマッピングすることができます。詳細については、「任意データのマッピング」を参照してください。

DynamoDBTyped

標準属性タイプの追加を上書きする注釈を使用できます。そのタイプでデフォルト属性のバインドを適用する場合、スタンダードの種類には注釈を必要としません。

DynamoDBVersionAttribute

オプティミスティックロックのバージョン番号を格納するためのクラスプロパティを識別します。`DynamoDBMapper` は、新しい項目を保存するときにこのプロパティにバージョン番号を割り当てます。バージョン番号は項目を更新するたびに増えていきます。サポートされているのは番号によるスカラー型だけです。 のデータ型の詳細については、「」を参照してくださいデータ型 バージョニングの詳細については、「バージョン番号を使用した楽観的ロック」を参照してください。

DynamoDBMapper クラス

`DynamoDBMapper` クラスは、Amazon DynamoDB のエントリポイントとなります。このクラスは、DynamoDB エンドポイントへのアクセスを提供し、さまざまなテーブルのデータを使用できるようにします。また、アイテムに対する作成、読み込み、更新、削除 (CRUD) の各オペレーションの実行、およびテーブルに対するクエリやスキャンを行うことができます。このクラスでは、DynamoDB を操作するために以下のメソッドが提供されます。

対応する Javadoc ドキュメントについては、AWS SDK for Java API リファレンスの「DynamoDBMapper」を参照してください。

トピック

- batchDelete
- バッチ書き込み
- transactionWrite
- transactionLoad
- count
- generateCreateTableRequest
- createS3Link
- getS3ClientCache

save

指定したオブジェクトがテーブルに保存されます。このメソッドで必要なパラメータは、保存するオブジェクトだけです。DynamoDBMapperConfig オブジェクトを使用して、オプションの設定パラメータを入力できます。

同じプライマリキーを持つ項目が存在しない場合は、このメソッドによってテーブル内に新しい項目が作成されます。同じプライマリキーを持つ項目が存在する場合は、その既存の項目が更新されます。パーティションキーとソートキーが String 型で、@DynamoDBAutoGeneratedKey によって注釈が付けられている場合、初期化しなければ、ランダムな UUID (Universally Unique Identifier) が与えられます。@DynamoDBVersionAttribute で注釈が付けられたバージョンフィールドは、バージョンが 1 ずつ増えていきます。さらに、バージョンフィールドが更新されるかキーが生成されると、オペレーションの結果として、渡されたオブジェクトが更新されます。

デフォルトでは、マップされたクラスプロパティに対応する属性のみが更新されます。アイテムのその他の既存の属性には影響はありません。ただし、SaveBehavior.CLOBBER を指定すると、項目が完全に上書きされるようにすることができます。

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

バージョニングを有効にした場合は、クライアント側とサーバー側で項目のバージョンが一致する必要があります。ただし、SaveBehavior.CLOBBER オプションを使用する場合は、バージョンを一致させる必要はありません。バージョニングの詳細については、「バージョン番号を使用した楽観的ロック」を参照してください。

load

テーブルから項目を取り出します。取得する項目のプライマリキーを入力する必要があります。DynamoDBMapperConfig オブジェクトを使用して、オプションの設定パラメータを入力できます。たとえば次の Java ステートメントに示すように、オプションで強力な整合性のある読み込みをリクエストして、このメソッドによって最新の項目の値だけを取り出すようにすることができます。

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

DynamoDB のデフォルトでは、結果整合性のある値を持つ項目が返されます。DynamoDB の結果整合性モデルの詳細については、「読み込み整合性」を参照してください。

削除

テーブルから項目を削除します。マッピングされたクラスのオブジェクトインスタンスを渡す必要があります。

バージョニングを有効にした場合は、クライアント側とサーバー側で項目のバージョンが一致する必要があります。ただし、SaveBehavior.CLOBBER オプションを使用する場合は、バージョンを一致させる必要はありません。バージョニングの詳細については、「バージョン番号を使用した楽観的ロック」を参照してください。

query

テーブルまたはセカンダリインデックスをクエリする

フォーラムスレッドの返信を格納する Reply というテーブルがあるとします。各スレッドの件名については、0 以上の返信を受け取ることができます。Reply テーブルのプライマリキーは、Id および ReplyDateTime フィールドで構成されます。ここで、Id はプライマリキーのパーティションキー、ReplyDateTime はソートキーを表します。

```
Reply ( Id, ReplyDateTime, ... )
```

Reply クラスと、それに対応する DynamoDB 内の Reply テーブルの間で、マッピングを作成したとします。次の Java コードでは、DynamoDBMapper を使用して特定のスレッド件名に対する過去 2 週間のすべての返信を検索しています。

Example

```
String forumName = "&DDB;";
String forumSubject = "&DDB; Thread 1";
String partitionKey = forumName + "#" + forumSubject;

long twoWeeksAgoMilli = (new Date()).getTime() - (14L*24L*60L*60L*1000L);
Date twoWeeksAgo = new Date();
twoWeeksAgo.setTime(twoWeeksAgoMilli);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
String twoWeeksAgoStr = df.format(twoWeeksAgo);

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS(partitionKey));
eav.put(":v2",new AttributeValue().withS(twoWeeksAgoStr.toString()));

DynamoDBQueryExpression<Reply> queryExpression = new DynamoDBQueryExpression<Reply>()
    .withKeyConditionExpression("Id = :v1 and ReplyDateTime > :v2")
    .withExpressionAttributeValues(eav);

List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);
```

このクエリでは、Reply オブジェクトのコレクションが返されます。

デフォルトでは、query メソッドによって、「遅延ロード」されたコレクションが返されます。最初に結果が 1 ページのみ返され、必要に応じて、さらに次ページを要求するサービス呼び出しが行われます。一致するすべての項目を取得するには、latestReplies コレクションを反復処理します。

コレクションで size() メソッドを呼び出すと、正確なカウントを提供するためにすべての結果がロードされます。これにより、プロビジョニングされたスループットが大量に消費され、非常に大きなテーブルでは JVM 内のすべてのメモリが消費されることさえあります。

インデックスにクエリを実行するには、最初にインデックスをマッパークラスとしてモデリングする必要があります。今、Reply テーブルに、PostedBy-Message-Index という名前のグローバルセカンダリインデックスが存在すると仮定します。このインデックスのパーティションキーは PostedBy キーで、ソートキーは Message です。インデックス内の項目のクラス定義は次のようになります。

```
@DynamoDBTable(tableName="Reply")
public class PostedByMessage {
    private String postedBy;
    private String message;

    @DynamoDBIndexHashKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "PostedBy")
    public String getPostedBy() { return postedBy; }
    public void setPostedBy(String postedBy) { this.postedBy = postedBy; }

    @DynamoDBIndexRangeKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "Message")
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

   // Additional properties go here.
}
```

`@DynamoDBTable` 注釈は、このインデックスが `Reply` テーブルに関連付けられていることを示します。`@DynamoDBIndexHashKey` 注釈はインデックスのパーティションキー (PostedBy) を示し、`@DynamoDBIndexRangeKey` はインデックスのソートキー (Message) を示します。

ここで、`DynamoDBMapper` を使用してインデックスにクエリを実行し、特定のユーザーによって投稿されたメッセージのサブセットを取得できます。テーブル間やインデックス間でマッピングが競合せず、マッピングが既にマッパーで行われている場合は、インデックス名を指定する必要はありません。マッパーは、プライマリキーとソートキーに基づいて推論します。次のコードでは、グローバルセカンダリインデックスをクエリしています。グローバルセカンダリインデックスは、結果整合性のある読み込みをサポートしますが、強力な整合性のある読み込みはサポートしていないため、`withConsistentRead(false)` を指定する必要があります。

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1",  new AttributeValue().withS("User A"));
eav.put(":v2",  new AttributeValue().withS("DynamoDB"));

DynamoDBQueryExpression<PostedByMessage> queryExpression = new
 DynamoDBQueryExpression<PostedByMessage>()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false)
    .withKeyConditionExpression("PostedBy = :v1 and begins_with(Message, :v2)")
    .withExpressionAttributeValues(eav);

List<PostedByMessage> iList =  mapper.query(PostedByMessage.class, queryExpression);
```

このクエリでは、`PostedByMessage` オブジェクトのコレクションが返されます。

queryPage

テーブルまたはセカンダリインデックスのクエリを実行し、一致した結果を 1 ページ返します。query メソッドと同様、パーティションキー値とソートキー属性に適用されるクエリフィルタを指定する必要があります。ただし queryPage では、データの最初の "ページ"、つまり 1 MB 以内に収まるデータ量のみが返されます。

scan

テーブル全体またはセカンダリインデックスをスキャンします。オプションで FilterExpression を指定して結果セットをフィルタリングできます。

フォーラムスレッドの返信を格納する Reply というテーブルがあるとします。各スレッドの件名については、0 以上の返信を受け取ることができます。Reply テーブルのプライマリキーは、Id および ReplyDateTime フィールドで構成されます。ここで、Id はプライマリキーのパーティションキー、ReplyDateTime はソートキーを表します。

```
Reply ( Id, ReplyDateTime, ... )
```

Reply テーブルに Java クラスをマッピングした場合は、DynamoDBMapper を使用してテーブルをスキャンできます。たとえば、以下の Java コードは、Reply テーブル全体をスキャンし、特定の年の返信のみを返します。

Example

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS("2015"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("begins_with(ReplyDateTime,:v1)")
    .withExpressionAttributeValues(eav);

List<Reply> replies =  mapper.scan(Reply.class, scanExpression);
```

デフォルトでは、scan メソッドによって、「遅延ロード」されたコレクションが返されます。最初に結果が 1 ページのみ返され、必要に応じて、さらに次ページを要求するサービス呼び出しが行われます。一致するすべての項目を取得するには、replies コレクションを反復処理します。

コレクションで size() メソッドを呼び出すと、正確なカウントを提供するためにすべての結果がロードされます。これにより、プロビジョニングされたスループットが大量に消費され、非常に大きなテーブルでは JVM 内のすべてのメモリが消費されることさえあります。

インデックスをスキャンするには、最初にインデックスをマッパークラスとしてモデリングする必要があります。今、Reply テーブルには、PostedBy-Message-Index という名前のグローバルセカンダリインデックスがあると仮定します。このインデックスのパーティションキーは PostedBy キーで、ソートキーは Message です。このインデックスのマッパークラスは、query セクションに示されています。また、@DynamoDBIndexHashKey と @DynamoDBIndexRangeKey の注釈を使用して、インデックスパーティションキーとソートキーを指定します。

以下のサンプルコードでは PostedBy-Message-Index をスキャンします。スキャンフィルタを使用しないので、インデックス内のすべての項目が返されます。

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

テーブルまたはセカンダリインデックスがスキャンされ、一致する結果が 1 ページ返されます。scan メソッドと同様に、オプションで FilterExpression を指定して結果セットをフィルタリングできます。ただし、scanPage では、データの最初の "ページ"、つまり、1 MB 以内に収まるデータ量のみが返されます。

parallelScan

テーブル全体、またはセカンダリインデックスの並列スキャンが実行されます。テーブルの論理セグメントの数と、結果をフィルタするスキャン式を指定します。parallelScan では、スキャンタスクが複数のワーカーに分割され、論理セグメントごとに 1 つのワーカーが割り当てられます。ワーカーは、データを並列に処理し、結果を返します。

次の Java コード例では、Product テーブルに対して並列スキャンを実行します。

```
int numberOfThreads = 4;

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":n", new AttributeValue().withN("100"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("Price <= :n")
    .withExpressionAttributeValues(eav);
```

```
List<Product> scanResult = mapper.parallelScan(Product.class, scanExpression,
 numberOfThreads);
```

parallelScan の使用を示す Java コード例については、「DynamoDBMapper クエリおよびスキャンオペレーション」を参照してください。

batchSave

AmazonDynamoDB.batchWriteItem メソッドに対する 1 つ以上の呼び出しを使用して、1 つ以上のテーブルにオブジェクトを保存します。このメソッドでは、トランザクション保証はなされません。

次の Java コードでは、2 つの項目 (書籍) を ProductCatalog テーブルに保存します。

```
Book book1 = new Book();
book1.setId(901);
book1.setProductCategory("Book");
book1.setTitle("Book 901 Title");

Book book2 = new Book();
book2.setId(902);
book2.setProductCategory("Book");
book2.setTitle("Book 902 Title");

mapper.batchSave(Arrays.asList(book1, book2));
```

batchLoad

テーブルのプライマリキーを使用して、1 つ以上のテーブルから複数の項目を取り出します。

次の Java コードでは、2 つの異なるテーブルから 2 つの項目を取得します。

```
ArrayList<Object> itemsToGet = new ArrayList<Object>();

ForumItem forumItem = new ForumItem();
forumItem.setForumName("Amazon DynamoDB");
itemsToGet.add(forumItem);

ThreadItem threadItem = new ThreadItem();
threadItem.setForumName("Amazon DynamoDB");
threadItem.setSubject("Amazon DynamoDB thread 1 message text");
```

```
itemsToGet.add(threadItem);

Map<String, List<Object>> items = mapper.batchLoad(itemsToGet);
```

batchDelete

AmazonDynamoDB.batchWriteItem メソッドに対する 1 つ以上の呼び出しを使用して、1 つ以上のテーブルからオブジェクトを削除します。このメソッドでは、トランザクション保証はなされません。

次の Java コードでは、2 つの項目 (書籍) を ProductCatalog テーブルから削除します。

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

バッチ書き込み

AmazonDynamoDB.batchWriteItem メソッドに対する 1 つ以上の呼び出しを使用して、1 つ以上のテーブルに対してオブジェクトの保存および削除を行います。このメソッドではトランザクション保証はなされず、バージョニング (条件付き入力または削除) もサポートされません。

次の Java コードでは、新しい項目を Forum テーブルと Thread テーブルに書き込み、ProductCatalog テーブルから項目を削除します。

```
// Create a Forum item to save
Forum forumItem = new Forum();
forumItem.setName("Test BatchWrite Forum");

// Create a Thread item to save
Thread threadItem = new Thread();
threadItem.setForumName("AmazonDynamoDB");
threadItem.setSubject("My sample question");

// Load a ProductCatalog item to delete
Book book3 = mapper.load(Book.class, 903);

List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
List<Book> objectsToDelete = Arrays.asList(book3);

mapper.batchWrite(objectsToWrite, objectsToDelete);
```

transactionWrite

AmazonDynamoDB.transactWriteItems メソッドに対する 1 回の呼び出しを使用して、1 つまたは複数のテーブルに対してオブジェクトの保存および削除を行います。

トランザクション固有の例外のリストについては、「TransactWriteItems エラー」を参照してください。

DynamoDB トランザクション、および提供されるアトミック性、整合性、分離、耐久性 (ACID) の保証の詳細については、「DynamoDB トランザクションで複雑なワークフローを管理する」を参照してください。

> Note
>
> このメソッドでは、以下をサポートしていません。
>
> - DynamoDBMapperConfig.SaveBehavior

次の Java コードでは、Forum と Thread の各テーブルに、トランザクションとして新しい項目を書き込みます。

```
Thread s3ForumThread = new Thread();
s3ForumThread.setForumName("S3 Forum");
s3ForumThread.setSubject("Sample Subject 1");
s3ForumThread.setMessage("Sample Question 1");

Forum s3Forum = new Forum();
s3Forum.setName("S3 Forum");
s3Forum.setCategory("Amazon Web Services");
s3Forum.setThreads(1);

TransactionWriteRequest transactionWriteRequest = new TransactionWriteRequest();
transactionWriteRequest.addPut(s3Forum);
transactionWriteRequest.addPut(s3ForumThread);
mapper.transactionWrite(transactionWriteRequest);
```

transactionLoad

AmazonDynamoDB.transactGetItems メソッドへの 1 回の呼び出しを使用して、1 つまたは複数のテーブルからオブジェクトをロードします。

トランザクション固有の例外のリストについては、「TransactGetItems エラー」を参照してください。

DynamoDB トランザクション、および提供されるアトミック性、整合性、分離、耐久性 (ACID) の保証の詳細については、「DynamoDB トランザクションで複雑なワークフローを管理する」を参照してください。

次の Java コードは、 Forum と Thread の各テーブルから、トランザクションとして一方の項目をロードします。

```
Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB Forum");

TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();
transactionLoadRequest.addLoad(dynamodbForum);
transactionLoadRequest.addLoad(dynamodbForumThread);
mapper.transactionLoad(transactionLoadRequest);
```

count

指定されたスキャン式の値を求め、一致する項目数を返します。項目データは返されません。

generateCreateTableRequest

DynamoDB テーブルに対応する POJO クラスを解析し、そのテーブルの CreateTableRequest を返します。

createS3Link

Amazon S3 内にあるオブジェクトへのリンクを作成します。バケット名とキー名を指定する必要があります。キー名によって、バケット内のオブジェクトを一意に識別します。

createS3Link を使用するには、マッパークラスでゲッターメソッドとセッターメソッドを定義する必要があります。次のサンプルコードでは、これを示しており、新しい属性と getter/setter メソッドを CatalogItem クラスに追加しています。

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    ...
    public S3Link productImage;

    ....

    @DynamoDBAttribute(attributeName = "ProductImage")
    public S3Link getProductImage() {
            return productImage;
    }

    public void setProductImage(S3Link productImage) {
        this.productImage = productImage;
    }

...
}
```

次の Java コードでは、Product テーブルに書き込まれる新しい項目を定義しています。この項目には、製品イメージへのリンクが含まれ、そのイメージデータは Amazon S3 にアップロードされています。

```
CatalogItem item = new CatalogItem();

item.setId(150);
item.setTitle("Book 150 Title");

String myS3Bucket = "myS3bucket";
String myS3Key = "productImages/book_150_cover.jpg";
item.setProductImage(mapper.createS3Link(myS3Bucket, myS3Key));

item.getProductImage().uploadFrom(new File("/file/path/book_150_cover.jpg"));

mapper.save(item);
```

S3Link クラスには、他にも、Amazon S3 のオブジェクトを操作するためのさまざまなメソッドが用意されています。詳細については、「Javadocs for S3Link」を参照してください。

getS3ClientCache

Amazon S3 にアクセスする際の基盤となる S3ClientCache を返します。S3ClientCache は、AmazonS3Client オブジェクトのスマートマップです。複数のクライアントがある場合、S3ClientCache によって、AWS リージョン別にクライアントを整理しやすくなり、新しい Amazon S3 クライアントをオンデマンドで作成できるようになります。

DynamoDBMapper のオプションの構成設定

DynamoDBMapper のインスタンスを作成すると、そのインスタンスには、特定のデフォルトの動作があります。DynamoDBMapperConfig クラスを使用して、このデフォルトの動作をオーバーライドできます。

次のコードスニペットでは、カスタム設定を使用して DynamoDBMapper を作成します。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapperConfig mapperConfig = DynamoDBMapperConfig.builder()
        .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
        .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
        .withTableNameOverride(null)

 .withPaginationLoadingStrategy(DynamoDBMapperConfig.PaginationLoadingStrategy.EAGER_LOADING)
    .build();

DynamoDBMapper mapper = new DynamoDBMapper(client, mapperConfig);
```

詳細については、AWS SDK for Java API リファレンスの「DynamoDBMapperConfig」を参照してください。

DynamoDBMapperConfig のインスタンスに対して、次の引数を指定できます。

- DynamoDBMapperConfig.ConsistentReads 列挙値:
  - EVENTUAL – マッパーインスタンスは、結果整合性のある読み込みリクエストを使用します。
  - CONSISTENT – マッパーインスタンスは、強力な整合性のある読み取りリクエストを使用します。このオプションの設定は、load、query、または scan オペレーションに対して使用できます。強力な整合性のある読み込みは、パフォーマンスと請求に影響を与えます。詳細については、DynamoDB の製品の詳細ページを参照してください。

  マッパーインスタンスに読み込み整合性の設定を指定しない場合、デフォルトは EVENTUAL になります。

  > **Note**
  >
  > この値は、DynamoDBMapper の query、querypage、load、batch load の各オペレーションに適用されます。

- DynamoDBMapperConfig.PaginationLoadingStrategy 列挙値 – query または scan からの結果など、ページ分割されたデータのリストをマッパーインスタンスが処理する方法を制御します。
    - LAZY_LOADING – マッパーインスタンスは、可能な場合にデータをロードし、ロードされたすべての結果をメモリに保持します。
    - EAGER_LOADING – マッパーインスタンスは、リストが初期化されるとすぐにデータをロードします。
    - ITERATION_ONLY – リストからの読み込みに Iterator のみを使用することができます。反復中、リストは、前のすべての結果をクリアしてから、次のページをロードすることで、最大 1 ページのロードされた結果をメモリに維持するようにします。つまり、リストを反復できるのは 1 回だけです。メモリのオーバーヘッドを低減するために、大きい項目を処理するときにこの方法を使用することをお勧めします。

  マッパーインスタンスにページ分割ロードの方法を指定しない場合、デフォルトは LAZY_LOADING になります。

- DynamoDBMapperConfig.SaveBehavior 列挙値 – 保存オペレーション中にマッパーインスタンスが属性を処理する方法を指定します。
    - UPDATE – 保存オペレーション中、すべてのモデル化された属性が更新され、モデル化されていない属性は影響を受けません。プリミティブな数値型（byte、int、long）は 0 に設定されます。オブジェクト型は null に設定されます。
    - CLOBBER – 保存オペレーション中、モデル化されていない属性も含め、すべての属性をクリアし置き換えます。このオペレーションを行うには、項目を削除し、再作成します。また、バージョン付きフィールドの制約は無視されます。

  マッパーインスタンスに保存動作を指定しない場合、デフォルトは UPDATE になります。

> **Note**
>
> DynamoDBMapper トランザクションオペレーションは DynamoDBMapperConfig.SaveBehavior 列挙をサポートしません。

- DynamoDBMapperConfig.TableNameOverride オブジェクト – クラスの DynamoDBTable アノテーションによって指定されたテーブル名を無視し、代わりに指定した別のテーブル名を使用するように、マッパーインスタンスに指示します。これは、実行時にデータを複数のテーブルに分割する場合に役立ちます。

必要に応じて、オペレーションごとに DynamoDBMapper のデフォルトの設定オブジェクトをオーバーライドできます。

バージョン番号を使用した楽観的ロック

オプティミスティックロックとは、更新 (または削除) しているクライアント側の項目と、Amazon DynamoDB の項目を確実に同じになるようにするための手段です。この方法を使用すると、データベースの書き込みは、他のユーザーの書き込みによって上書きされないように保護されます。逆の場合も同様に保護されます。

オプティミスティックロックを使用する場合、各項目には、バージョン番号として機能する属性があります。項目をテーブルから取り出すと、アプリケーションは、その項目のバージョン番号を記録します。サーバー側のバージョン番号が変更されていない場合のみ、項目を更新できます。バージョンの不一致がある場合は、前に他のユーザーによってそのアイテムが変更されたことを意味します。アイテムの古いバージョンがあるため、更新の試行は失敗します。その場合は更新をやり直します。もう一度項目を取得して、更新してください。オプティミスティックロックでは、他のユーザーが行った変更を誤って上書きできないようにします。また、お客様が行った変更を他のユーザーが誤って変更することを防ぐこともできます。

独自の楽観的ロック戦略を実装することもできますが、AWS SDK for Java には @DynamoDBVersionAttribute アノテーションが用意されています。テーブルのマッピングクラスで、バージョン番号を保存する 1 つのプロパティを指定し、この注釈を使用してそのプロパティをマーキングします。オブジェクトを保存すると、DynamoDB テーブル内の対応する項目に、バージョン番号を格納するための属性が追加されます。DynamoDBMapper では、最初にオブジェクトを保存したときにバージョン番号が割り当てられ、項目を更新するたびにバージョン番号が自動的にインクリメントされます。更新または削除リクエストは、クライアント側オブジェクトのバージョン番号が、DynamoDB テーブルで対応する項目のバージョン番号に一致する場合のみ成功します。

ConditionalCheckFailedException以下の場合、 はスローされます。

- オプティミスティックロックを @DynamoDBVersionAttribute と共に使用しており、サーバーのバージョンの値が、クライアント側の値とは異なる場合。
- DynamoDBMapper で DynamoDBSaveExpression を使用してデータを保存するときに、独自の条件付き制約を指定し、それらの制約に障害が生じた場合。

Note

- DynamoDB のグローバルテーブルはグローバルテーブルでは、同時更新間の「最新書き込み」の照合を行います。グローバルテーブルを使用する場合、最後に書き込まれたポリシーが採用されます。したがってこの場合、ロック戦略は想定通りに機能しません。
- DynamoDBMapper トランザクションの書き込みオペレーションは、同じオブジェクトでの @DynamoDBVersionAttribute 注釈と条件式をサポートしていません。トランザクション書き込み内のオブジェクトに @DynamoDBVersionAttribute の注釈が付けられ、条件式も含まれている場合、SdkClientException がスローされます。

たとえば次の Java コードでは、複数のプロパティを持つ CatalogItem クラスを定義しています。Version プロパティは @DynamoDBVersionAttribute 注釈でタグ付けされています。

Example

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;
    private Long version;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer Id) { this.id = Id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN;}

    @DynamoDBAttribute(attributeName = "Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp;}
    public void setSomeProp(String someProp) {this.someProp = someProp;}

    @DynamoDBVersionAttribute
    public Long getVersion() { return version; }
    public void setVersion(Long version) { this.version = version;}
}
```

`@DynamoDBVersionAttribute` 注釈は、`Long` や `Integer` などの、null が許容された型を提供するプリミティブラッパークラスによって得られた、null が許容された型に適用できます。

オプティミスティックロックは、次の `DynamoDBMapper` メソッドに対して次のような影響があります。

- `save` – `DynamoDBMapper` は、新しい項目に対して初期バージョン番号 1 を割り当てます。項目を取得し、その項目の 1 つ以上のプロパティを更新して変更を保存する場合には、クライアント側とサーバー側のバージョン番号が一致する場合のみ、保存が成功します。`DynamoDBMapper` によってバージョン番号が自動的にインクリメントされます。
- `delete` – `delete` メソッドは 1 つのオブジェクトをパラメータとして指定し、項目を削除する前に `DynamoDBMapper` によるバージョンチェックが実行されます。バージョンチェックは、リクエスト内で `DynamoDBMapperConfig.SaveBehavior.CLOBBER` を指定して無効にすることができます。

  `DynamoDBMapper` 内のオプティミスティックロックの内部実装では、DynamoDB の条件付き更新と条件付き削除機能が使用されます。
- `transactionWrite` —
  - `Put` – `DynamoDBMapper` は、新しい項目に対して初期バージョン番号 1 を割り当てます。項目を取得し、その項目の 1 つ以上のプロパティを更新して変更を保存する場合には、クライアント側とサーバー側のバージョン番号が一致する場合のみ、put オペレーションが成功します。`DynamoDBMapper` によってバージョン番号が自動的にインクリメントされます。
  - `Update` – `DynamoDBMapper` は、新しい項目に対して初期バージョン番号 1 を割り当てます。項目を取得し、その項目の 1 つ以上のプロパティを更新して変更を保存する場合には、クライアント側とサーバー側のバージョン番号が一致する場合のみ、更新オペレーションが成功します。`DynamoDBMapper` によってバージョン番号が自動的にインクリメントされます。

- Delete – DynamoDBMapper は、項目を削除する前にバージョンチェックを実行します。削除オペレーションは、クライアント側とサーバー側のバージョン番号が一致する場合にのみ成功します。
- ConditionCheck – @DynamoDBVersionAttribute アノテーションは、ConditionCheck オペレーションではサポートされていません。ConditionCheck 項目に @DynamoDBVersionAttribute の注釈が付いている場合、SdkClientException がスローされます。

楽観的ロックの無効化

オプティミスティックロックを無効にするには、DynamoDBMapperConfig.SaveBehavior 列挙値を UPDATE から CLOBBER に変更します。その場合は、バージョンチェックを省略する DynamoDBMapperConfig インスタンスを作成して、そのインスタンスをすべてのリクエストで使用します。DynamoDBMapperConfig.SaveBehavior とその他のオプションの DynamoDBMapper パラメータの詳細については、「DynamoDBMapper のオプションの構成設定 」を参照してください。

ロック動作は、特定のオペレーションだけに設定することもできます。たとえば次の Java コードスニペットではDynamoDBMapper を使用してカタログ項目を保存しています。オプションの DynamoDBMapperConfig.SaveBehavior パラメータを DynamoDBMapperConfig メソッドに追加することで、save を指定しています。

> **Note**
>
> transactionWrite メソッドは、DynamoDBMapperConfig.SaveBehavior 構成をサポートしていません。transactionWrite のオプティミスティックロックの無効化はサポートされていません。

Example

```
DynamoDBMapper mapper = new DynamoDBMapper(client);

// Load a catalog item.
CatalogItem item = mapper.load(CatalogItem.class, 101);
item.setTitle("This is a new title for the item");
...
// Save the item.
mapper.save(item,
    new DynamoDBMapperConfig(
        DynamoDBMapperConfig.SaveBehavior.CLOBBER));
```

任意データのマッピング

サポートされている Java 型 (「DynamoDB Mapper for Java でサポートされるデータ型」を参照) に加えて、Amazon DynamoDB 型に直接マッピングされない、アプリケーション内の型を使用できます。これらの型をマッピングするには、複合型を DynamoDB 対応の型に (またその反対に) 変換する処理を実装し、@DynamoDBTypeConverted アノテーションを使用して複合型のアクセサーメソッドに注釈付けを行う必要があります。コンバーターコードは、オブジェクトが保存またはロードされるときにデータを変換します。これは、複合型を消費するすべてのオペレーションでも使用されます。クエリおよびスキャンオペレーションの中でデータを比較する場合、その比較は DynamoDB に格納されているデータに対して行われることにご注意ください。

たとえば、プロパティ CatalogItem (Dimension 型) を定義する、次の DimensionType クラスがあるとします。このプロパティには、項目の寸法として高さ、幅、厚さが格納されています。これらの項目のディメンションを DynamoDB の中に文字列 (8.5x11x.05 など) として格納しようとしている場合を考えます。次の例は、DimensionType オブジェクトを文字列に変換し、文字列を DimensionType に変換するコンバーターコードを示しています。

> Note
>
> このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
> 以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

Example

```
public class DynamoDBMapperExample {

    static AmazonDynamoDB client;

    public static void main(String[] args) throws IOException {

        // Set the AWS region you want to access.
        Regions usWest2 = Regions.US_WEST_2;
        client = AmazonDynamoDBClientBuilder.standard().withRegion(usWest2).build();

        DimensionType dimType = new DimensionType();
        dimType.setHeight("8.00");
        dimType.setLength("11.0");
        dimType.setThickness("1.0");

        Book book = new Book();
        book.setId(502);
        book.setTitle("Book 502");
        book.setISBN("555-5555555555");
        book.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));
        book.setDimensions(dimType);

        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(book);

        Book bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Book info: " + "\n" + bookRetrieved);

        bookRetrieved.getDimensions().setHeight("9.0");
        bookRetrieved.getDimensions().setLength("12.0");
        bookRetrieved.getDimensions().setThickness("2.0");

        mapper.save(bookRetrieved);

        bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Updated book info: " + "\n" + bookRetrieved);
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
        private String ISBN;
        private Set<String> bookAuthors;
        private DimensionType dimensionType;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }
        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Authors")
        public Set<String> getBookAuthors() {
            return bookAuthors;
        }

        public void setBookAuthors(Set<String> bookAuthors) {
            this.bookAuthors = bookAuthors;
        }

        @DynamoDBTypeConverted(converter = DimensionTypeConverter.class)
        @DynamoDBAttribute(attributeName = "Dimensions")
        public DimensionType getDimensions() {
            return dimensionType;
        }

        @DynamoDBAttribute(attributeName = "Dimensions")
        public void setDimensions(DimensionType dimensionType) {
            this.dimensionType = dimensionType;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ", 
 dimensionType= "
                    + dimensionType.getHeight() + " X " + dimensionType.getLength() + " 
 X "
                    + dimensionType.getThickness()
                    + ", Id=" + id + ", Title=" + title + "]";
        }
    }

    static public class DimensionType {

        private String length;
        private String height;
        private String thickness;

        public String getLength() {
            return length;
        }

        public void setLength(String length) {
            this.length = length;
        }

        public String getHeight() {
            return height;
        }

        public void setHeight(String height) {
            this.height = height;
        }

        public String getThickness() {
            return thickness;
        }

        public void setThickness(String thickness) {
            this.thickness = thickness;
        }
    }

    // Converts the complex type DimensionType to a string and vice-versa.
    static public class DimensionTypeConverter implements DynamoDBTypeConverter<String, 
 DimensionType> {
        @Override
        public String convert(DimensionType object) {
            DimensionType itemDimensions = (DimensionType) object;
            String dimension = null;
            try {
                if (itemDimensions != null) {
                    dimension = String.format("%s x %s x %s",
 itemDimensions.getLength(), itemDimensions.getHeight(),
                            itemDimensions.getThickness());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return dimension;
        }

        @Override
        public DimensionType unconvert(String s) {

            DimensionType itemDimension = new DimensionType();
            try {
                if (s != null && s.length() != 0) {
                    String[] data = s.split("x");
                    itemDimension.setLength(data[0].trim());
                    itemDimension.setHeight(data[1].trim());
                    itemDimension.setThickness(data[2].trim());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return itemDimension;
        }
    }
}
```

DynamoDBMapper の例

以下の Java コード例は、DynamoDBMapper クラスを使用してさまざまなオペレーションを実行する方法を示しています。これらの例を使用して、CRUD、クエリ、スキャン、バッチ、およびトランザクションの各オペレーションを実行できます。

トピック

- DynamoDBMapper CRUD オペレーション
- DynamoDBMapper クエリおよびスキャンオペレーション
- DynamoDBMapper バッチオペレーション
- DynamoDBMapper トランザクションオペレーション

DynamoDBMapper CRUD オペレーション

次の Java コード例では、`CatalogItem`、`Id`、`Title`、および `ISBN` プロパティを指定して、`Authors` クラスを宣言しています。このクラスでは、アノテーションにより前出のプロパティを DynamoDB の `ProductCatalog` テーブルにマッピングしています。この例では、次に `DynamoDBMapper` を使用して、書籍オブジェクトを保存、取得、更新し、書籍項目を削除します。

> **Note**
>
> このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
> 以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

インポート

```
import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapperConfig;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
```

Code

```
public class DynamoDBMapperCRUDExample {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

    public static void main(String[] args) throws IOException {
        testCRUDOperations();
        System.out.println("Example complete!");
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class CatalogItem {
        private Integer id;
        private String title;
        private String ISBN;
        private Set<String> bookAuthors;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public Integer getId() {
            return id;
        }

        public void setId(Integer id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Authors")
        public Set<String> getBookAuthors() {
            return bookAuthors;
        }

        public void setBookAuthors(Set<String> bookAuthors) {
            this.bookAuthors = bookAuthors;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ", id=" + id
 + ", title=" + title + "]";
        }
    }

    private static void testCRUDOperations() {

        CatalogItem item = new CatalogItem();
        item.setId(601);
        item.setTitle("Book 601");
        item.setISBN("611-1111111111");
        item.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));

        // Save the item (book).
        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(item);

        // Retrieve the item.
        CatalogItem itemRetrieved = mapper.load(CatalogItem.class, 601);
        System.out.println("Item retrieved:");
        System.out.println(itemRetrieved);

        // Update the item.
        itemRetrieved.setISBN("622-2222222222");
        itemRetrieved.setBookAuthors(new HashSet<String>(Arrays.asList("Author1",
 "Author3")));
        mapper.save(itemRetrieved);
        System.out.println("Item updated:");
        System.out.println(itemRetrieved);

        // Retrieve the updated item.
        DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
                .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
                .build();
        CatalogItem updatedItem = mapper.load(CatalogItem.class, 601, config);
        System.out.println("Retrieved the previously updated item:");
        System.out.println(updatedItem);

        // Delete the item.
        mapper.delete(updatedItem);

        // Try to retrieve deleted item.
        CatalogItem deletedItem = mapper.load(CatalogItem.class, updatedItem.getId(),
 config);
        if (deletedItem == null) {
            System.out.println("Done - Sample item is deleted.");
        }
    }
}
```

DynamoDBMapper クエリおよびスキャンオペレーション

このセクションの Java の例では、次のクラスを定義して、Amazon DynamoDB 内のテーブルにマッピングします。サンプルテーブル作成の詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

- Book クラスは、ProductCatalog テーブルにマッピングされます。
- Forum、Thread、および Reply クラスは、同じ名前のテーブルにマッピングされます。

この例ではさらに、DynamoDBMapper インスタンスを使用して、次のクエリとスキャンオペレーションを実行します。

- Id によって書籍を取得します。

  ProductCatalog テーブルには、プライマリキーとして Id があります。プライマリキーの一部にソートキーは含まれていません。したがって、テーブルのクエリを行うことはできません。項目は Id 値を使用して取得できます。
- Reply テーブルに対して次のクエリを実行します。

  Reply テーブルのプライマリキーは、Id および ReplyDateTime 属性で構成されています。ReplyDateTime はソートキーです。したがってこのテーブルではクエリを実行できます。
  - 過去 15 日間に投稿されたフォーラムスレッドに対する返信を検索します。

  - 特定の日付範囲の間に投稿されたフォーラムスレッドに対する返信を検索します。
- ProductCatalog テーブルをスキャンし、価格が指定した値より低い書籍を検索します。

  パフォーマンス上の理由から、スキャンオペレーションではなくクエリオペレーションを使用するようにしてください。ただし、場合によってはテーブルをスキャンする必要があります。データ入力エラーがあり、書籍の価格の 1 つが 0 未満に設定されたとします。この例では、ProductCategory テーブルをスキャンして、価格が 0 未満である書籍項目 (ProductCategory は書籍) を検索しています。
- ProductCatalog テーブルのパラレルスキャンを実行して特定のタイプの自転車を検索します。

> **Note**
>
> このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
> 以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

インポート

```
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBQueryExpression;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBScanExpression;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
```

Code

```
public class DynamoDBMapperQueryScanExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

    public static void main(String[] args) throws Exception {
        try {

            DynamoDBMapper mapper = new DynamoDBMapper(client);

            // Get a book - Id=101
            GetBook(mapper, 101);
            // Sample forum and thread to test queries.
            String forumName = "Amazon DynamoDB";
            String threadSubject = "DynamoDB Thread 1";
            // Sample queries.
            FindRepliesInLast15Days(mapper, forumName, threadSubject);
            FindRepliesPostedWithinTimePeriod(mapper, forumName, threadSubject);

            // Scan a table and find book items priced less than specified
            // value.
            FindBooksPricedLessThanSpecifiedValue(mapper, "20");

            // Scan a table with multiple threads and find bicycle items with a
            // specified bicycle type
            int numberOfThreads = 16;
            FindBicyclesOfSpecificTypeWithMultipleThreads(mapper, numberOfThreads,
 "Road");

            System.out.println("Example complete!");

        } catch (Throwable t) {
            System.err.println("Error running the DynamoDBMapperQueryScanExample: " +
 t);
            t.printStackTrace();
        }
    }

    private static void GetBook(DynamoDBMapper mapper, int id) throws Exception {
        System.out.println("GetBook: Get book Id='101' ");
        System.out.println("Book table has no sort key. You can do GetItem, but not
 Query.");
        Book book = mapper.load(Book.class, id);
        System.out.format("Id = %s Title = %s, ISBN = %s %n", book.getId(),
 book.getTitle(), book.getISBN());
    }

    private static void FindRepliesInLast15Days(DynamoDBMapper mapper, String
 forumName, String threadSubject)
            throws Exception {
        System.out.println("FindRepliesInLast15Days: Replies within last 15 days.");

        String partitionKey = forumName + "#" + threadSubject;

        long twoWeeksAgoMilli = (new Date()).getTime() - (15L * 24L * 60L * 60L *
 1000L);
        Date twoWeeksAgo = new Date();
        twoWeeksAgo.setTime(twoWeeksAgoMilli);
        SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");
        dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
        String twoWeeksAgoStr = dateFormatter.format(twoWeeksAgo);

        Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
        eav.put(":val1", new AttributeValue().withS(partitionKey));
        eav.put(":val2", new AttributeValue().withS(twoWeeksAgoStr.toString()));

        DynamoDBQueryExpression<Reply> queryExpression = new
 DynamoDBQueryExpression<Reply>()
                .withKeyConditionExpression("Id = :val1 and ReplyDateTime
 > :val2").withExpressionAttributeValues(eav);

        List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);

        for (Reply reply : latestReplies) {
            System.out.format("Id=%s, Message=%s, PostedBy=%s %n, ReplyDateTime=%s %n",
 reply.getId(),
                    reply.getMessage(), reply.getPostedBy(), reply.getReplyDateTime());
        }
    }

    private static void FindRepliesPostedWithinTimePeriod(DynamoDBMapper mapper, String
 forumName, String threadSubject)
            throws Exception {
        String partitionKey = forumName + "#" + threadSubject;

        System.out.println(
                "FindRepliesPostedWithinTimePeriod: Find replies for thread Message =
 'DynamoDB Thread 2' posted within a period.");
        long startDateMilli = (new Date()).getTime() - (14L * 24L * 60L * 60L *
 1000L); // Two

 // weeks

 // ago.
        long endDateMilli = (new Date()).getTime() - (7L * 24L * 60L * 60L * 1000L); //
 One
                                                                                      //
 week
                                                                                      //
 ago.
        SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");
        dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
        String startDate = dateFormatter.format(startDateMilli);
        String endDate = dateFormatter.format(endDateMilli);

        Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
        eav.put(":val1", new AttributeValue().withS(partitionKey));
        eav.put(":val2", new AttributeValue().withS(startDate));
        eav.put(":val3", new AttributeValue().withS(endDate));

        DynamoDBQueryExpression<Reply> queryExpression = new
 DynamoDBQueryExpression<Reply>()
                .withKeyConditionExpression("Id = :val1 and ReplyDateTime between :val2
 and :val3")
                .withExpressionAttributeValues(eav);

        List<Reply> betweenReplies = mapper.query(Reply.class, queryExpression);

        for (Reply reply : betweenReplies) {
            System.out.format("Id=%s, Message=%s, PostedBy=%s %n, PostedDateTime=%s
 %n", reply.getId(),
                    reply.getMessage(), reply.getPostedBy(), reply.getReplyDateTime());
        }

    }

    private static void FindBooksPricedLessThanSpecifiedValue(DynamoDBMapper mapper,
 String value) throws Exception {
        System.out.println("FindBooksPricedLessThanSpecifiedValue: Scan
 ProductCatalog.");

        Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
        eav.put(":val1", new AttributeValue().withN(value));
        eav.put(":val2", new AttributeValue().withS("Book"));

        DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
                .withFilterExpression("Price < :val1 and ProductCategory
 = :val2").withExpressionAttributeValues(eav);

        List<Book> scanResult = mapper.scan(Book.class, scanExpression);

        for (Book book : scanResult) {
            System.out.println(book);
        }
    }

    private static void FindBicyclesOfSpecificTypeWithMultipleThreads(DynamoDBMapper
 mapper, int numberOfThreads,
            String bicycleType) throws Exception {

        System.out.println("FindBicyclesOfSpecificTypeWithMultipleThreads: Scan
 ProductCatalog With Multiple Threads.");
        Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
        eav.put(":val1", new AttributeValue().withS("Bicycle"));
        eav.put(":val2", new AttributeValue().withS(bicycleType));

        DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
                .withFilterExpression("ProductCategory = :val1 and BicycleType
 = :val2")
                .withExpressionAttributeValues(eav);

        List<Bicycle> scanResult = mapper.parallelScan(Bicycle.class, scanExpression,
 numberOfThreads);
        for (Bicycle bicycle : scanResult) {
            System.out.println(bicycle);
        }
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
```

```
        private String ISBN;
        private int price;
        private int pageCount;
        private String productCategory;
        private boolean inPublication;

        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Price")
        public int getPrice() {
            return price;
        }

        public void setPrice(int price) {
            this.price = price;
        }

        @DynamoDBAttribute(attributeName = "PageCount")
        public int getPageCount() {
            return pageCount;
        }

        public void setPageCount(int pageCount) {
            this.pageCount = pageCount;
        }

        @DynamoDBAttribute(attributeName = "ProductCategory")
        public String getProductCategory() {
            return productCategory;
        }

        public void setProductCategory(String productCategory) {
            this.productCategory = productCategory;
        }

        @DynamoDBAttribute(attributeName = "InPublication")
        public boolean getInPublication() {
            return inPublication;
        }

        public void setInPublication(boolean inPublication) {
            this.inPublication = inPublication;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", price=" + price + ", product category=" +
 productCategory + ", id=" + id
                    + ", title=" + title + "]";
        }

    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Bicycle {
        private int id;
        private String title;
        private String description;
        private String bicycleType;
        private String brand;
        private int price;
        private List<String> color;
        private String productCategory;
```

```
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "Description")
        public String getDescription() {
            return description;
        }

        public void setDescription(String description) {
            this.description = description;
        }

        @DynamoDBAttribute(attributeName = "BicycleType")
        public String getBicycleType() {
            return bicycleType;
        }

        public void setBicycleType(String bicycleType) {
            this.bicycleType = bicycleType;
        }

        @DynamoDBAttribute(attributeName = "Brand")
        public String getBrand() {
            return brand;
        }

        public void setBrand(String brand) {
            this.brand = brand;
        }

        @DynamoDBAttribute(attributeName = "Price")
        public int getPrice() {
            return price;
        }

        public void setPrice(int price) {
            this.price = price;
        }

        @DynamoDBAttribute(attributeName = "Color")
        public List<String> getColor() {
            return color;
        }

        public void setColor(List<String> color) {
            this.color = color;
        }

        @DynamoDBAttribute(attributeName = "ProductCategory")
        public String getProductCategory() {
            return productCategory;
        }

        public void setProductCategory(String productCategory) {
            this.productCategory = productCategory;
        }

        @Override
        public String toString() {
            return "Bicycle [Type=" + bicycleType + ", color=" + color + ", price=" +
 price + ", product category="
                    + productCategory + ", id=" + id + ", title=" + title + "]";
        }

    }

    @DynamoDBTable(tableName = "Reply")
    public static class Reply {
        private String id;
        private String replyDateTime;
        private String message;
        private String postedBy;
        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public String getId() {
            return id;
        }

        public void setId(String id) {
            this.id = id;
        }

        // Range key
        @DynamoDBRangeKey(attributeName = "ReplyDateTime")
        public String getReplyDateTime() {
            return replyDateTime;
        }

        public void setReplyDateTime(String replyDateTime) {
            this.replyDateTime = replyDateTime;
        }

        @DynamoDBAttribute(attributeName = "Message")
        public String getMessage() {
            return message;
        }

        public void setMessage(String message) {
            this.message = message;
        }

        @DynamoDBAttribute(attributeName = "PostedBy")
        public String getPostedBy() {
            return postedBy;
        }

        public void setPostedBy(String postedBy) {
            this.postedBy = postedBy;
        }
    }

    @DynamoDBTable(tableName = "Thread")
    public static class Thread {
        private String forumName;
        private String subject;
        private String message;
        private String lastPostedDateTime;
        private String lastPostedBy;
        private Set<String> tags;
        private int answered;
        private int views;
        private int replies;

        // Partition key
        @DynamoDBHashKey(attributeName = "ForumName")
        public String getForumName() {
            return forumName;
        }

        public void setForumName(String forumName) {
            this.forumName = forumName;
        }

        // Range key
        @DynamoDBRangeKey(attributeName = "Subject")
        public String getSubject() {
            return subject;
        }

        public void setSubject(String subject) {
            this.subject = subject;
        }

        @DynamoDBAttribute(attributeName = "Message")
        public String getMessage() {
            return message;
        }

        public void setMessage(String message) {
            this.message = message;
        }

        @DynamoDBAttribute(attributeName = "LastPostedDateTime")
        public String getLastPostedDateTime() {
            return lastPostedDateTime;
        }

        public void setLastPostedDateTime(String lastPostedDateTime) {
            this.lastPostedDateTime = lastPostedDateTime;
        }

        @DynamoDBAttribute(attributeName = "LastPostedBy")
        public String getLastPostedBy() {
            return lastPostedBy;
        }

        public void setLastPostedBy(String lastPostedBy) {
            this.lastPostedBy = lastPostedBy;
        }

        @DynamoDBAttribute(attributeName = "Tags")
        public Set<String> getTags() {
            return tags;
        }

        public void setTags(Set<String> tags) {
            this.tags = tags;
        }

        @DynamoDBAttribute(attributeName = "Answered")
        public int getAnswered() {
            return answered;
        }

        public void setAnswered(int answered) {
            this.answered = answered;
        }

        @DynamoDBAttribute(attributeName = "Views")
        public int getViews() {
            return views;
        }

        public void setViews(int views) {
            this.views = views;
        }

        @DynamoDBAttribute(attributeName = "Replies")
        public int getReplies() {
            return replies;
        }

        public void setReplies(int replies) {
            this.replies = replies;
        }

    }

    @DynamoDBTable(tableName = "Forum")
    public static class Forum {
        private String name;
        private String category;
        private int threads;

        @DynamoDBHashKey(attributeName = "Name")
        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        @DynamoDBAttribute(attributeName = "Category")
        public String getCategory() {
            return category;
        }

        public void setCategory(String category) {
            this.category = category;
        }

        @DynamoDBAttribute(attributeName = "Threads")
        public int getThreads() {
            return threads;
        }

        public void setThreads(int threads) {
            this.threads = threads;
        }
    }
}
```

DynamoDBMapper バッチオペレーション

この Java サンプルコードでは、Book、Forum、Thread、および Reply クラスが宣言され、それらは DynamoDBMapper クラスを使用して Amazon DynamoDB テーブルにマッピングされています。

このコードには、次のバッチ書き込みオペレーションが含まれます。

- batchSave は、書籍項目を ProductCatalog テーブルに挿入します。
- batchDelete は、項目を ProductCatalog テーブルから削除します。
- batchWrite は、Forum および Thread テーブルから書籍項目を挿入したり、そこから削除したりします。

この例で使用されているテーブルの詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。以下の例をテストするための詳細な手順については、「Java コードの例」を参照してください。

インポート

```
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapperConfig;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
```

Code

```
public class DynamoDBMapperBatchWriteExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
```

```
   public static void main(String[] args) throws Exception {
       try {

           DynamoDBMapper mapper = new DynamoDBMapper(client);

           testBatchSave(mapper);
           testBatchDelete(mapper);
           testBatchWrite(mapper);

           System.out.println("Example complete!");

       } catch (Throwable t) {
           System.err.println("Error running the DynamoDBMapperBatchWriteExample: " +
t);
           t.printStackTrace();
       }
   }

   private static void testBatchSave(DynamoDBMapper mapper) {

       Book book1 = new Book();
       book1.setId(901);
       book1.setInPublication(true);
       book1.setISBN("902-11-11-1111");
       book1.setPageCount(100);
       book1.setPrice(10);
       book1.setProductCategory("Book");
       book1.setTitle("My book created in batch write");

       Book book2 = new Book();
       book2.setId(902);
       book2.setInPublication(true);
       book2.setISBN("902-11-12-1111");
       book2.setPageCount(200);
       book2.setPrice(20);
       book2.setProductCategory("Book");
       book2.setTitle("My second book created in batch write");

       Book book3 = new Book();
       book3.setId(903);
       book3.setInPublication(false);
       book3.setISBN("902-11-13-1111");
       book3.setPageCount(300);
       book3.setPrice(25);
```

```
        book3.setProductCategory("Book");
        book3.setTitle("My third book created in batch write");

        System.out.println("Adding three books to ProductCatalog table.");
        mapper.batchSave(Arrays.asList(book1, book2, book3));
    }

    private static void testBatchDelete(DynamoDBMapper mapper) {

        Book book1 = mapper.load(Book.class, 901);
        Book book2 = mapper.load(Book.class, 902);
        System.out.println("Deleting two books from the ProductCatalog table.");
        mapper.batchDelete(Arrays.asList(book1, book2));
    }

    private static void testBatchWrite(DynamoDBMapper mapper) {

        // Create Forum item to save
        Forum forumItem = new Forum();
        forumItem.setName("Test BatchWrite Forum");
        forumItem.setThreads(0);
        forumItem.setCategory("Amazon Web Services");

        // Create Thread item to save
        Thread threadItem = new Thread();
        threadItem.setForumName("AmazonDynamoDB");
        threadItem.setSubject("My sample question");
        threadItem.setMessage("BatchWrite message");
        List<String> tags = new ArrayList<String>();
        tags.add("batch operations");
        tags.add("write");
        threadItem.setTags(new HashSet<String>(tags));

        // Load ProductCatalog item to delete
        Book book3 = mapper.load(Book.class, 903);

        List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
        List<Book> objectsToDelete = Arrays.asList(book3);

        DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
                .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
                .build();

        mapper.batchWrite(objectsToWrite, objectsToDelete, config);
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
        private String ISBN;
        private int price;
        private int pageCount;
        private String productCategory;
        private boolean inPublication;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Price")
        public int getPrice() {
            return price;
        }
        public void setPrice(int price) {
            this.price = price;
        }

        @DynamoDBAttribute(attributeName = "PageCount")
        public int getPageCount() {
            return pageCount;
        }

        public void setPageCount(int pageCount) {
            this.pageCount = pageCount;
        }

        @DynamoDBAttribute(attributeName = "ProductCategory")
        public String getProductCategory() {
            return productCategory;
        }

        public void setProductCategory(String productCategory) {
            this.productCategory = productCategory;
        }

        @DynamoDBAttribute(attributeName = "InPublication")
        public boolean getInPublication() {
            return inPublication;
        }

        public void setInPublication(boolean inPublication) {
            this.inPublication = inPublication;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", price=" + price + ", product category=" +
productCategory + ", id=" + id
                    + ", title=" + title + "]";
        }

    }

    @DynamoDBTable(tableName = "Reply")
    public static class Reply {
        private String id;
        private String replyDateTime;
        private String message;
        private String postedBy;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public String getId() {
            return id;
        }

        public void setId(String id) {
            this.id = id;
        }

        // Sort key
        @DynamoDBRangeKey(attributeName = "ReplyDateTime")
        public String getReplyDateTime() {
            return replyDateTime;
        }

        public void setReplyDateTime(String replyDateTime) {
            this.replyDateTime = replyDateTime;
        }

        @DynamoDBAttribute(attributeName = "Message")
        public String getMessage() {
            return message;
        }

        public void setMessage(String message) {
            this.message = message;
        }

        @DynamoDBAttribute(attributeName = "PostedBy")
        public String getPostedBy() {
            return postedBy;
        }

        public void setPostedBy(String postedBy) {
            this.postedBy = postedBy;
        }
    }

    @DynamoDBTable(tableName = "Thread")
    public static class Thread {
        private String forumName;
        private String subject;
        private String message;
        private String lastPostedDateTime;
        private String lastPostedBy;
        private Set<String> tags;
        private int answered;
        private int views;
        private int replies;

        // Partition key
        @DynamoDBHashKey(attributeName = "ForumName")
        public String getForumName() {
            return forumName;
        }

        public void setForumName(String forumName) {
            this.forumName = forumName;
        }

        // Sort key
        @DynamoDBRangeKey(attributeName = "Subject")
        public String getSubject() {
            return subject;
        }

        public void setSubject(String subject) {
            this.subject = subject;
        }

        @DynamoDBAttribute(attributeName = "Message")
        public String getMessage() {
            return message;
        }

        public void setMessage(String message) {
            this.message = message;
        }

        @DynamoDBAttribute(attributeName = "LastPostedDateTime")
        public String getLastPostedDateTime() {
            return lastPostedDateTime;
        }
        public void setLastPostedDateTime(String lastPostedDateTime) {
            this.lastPostedDateTime = lastPostedDateTime;
        }

        @DynamoDBAttribute(attributeName = "LastPostedBy")
        public String getLastPostedBy() {
            return lastPostedBy;
        }

        public void setLastPostedBy(String lastPostedBy) {
            this.lastPostedBy = lastPostedBy;
        }

        @DynamoDBAttribute(attributeName = "Tags")
        public Set<String> getTags() {
            return tags;
        }

        public void setTags(Set<String> tags) {
            this.tags = tags;
        }

        @DynamoDBAttribute(attributeName = "Answered")
        public int getAnswered() {
            return answered;
        }

        public void setAnswered(int answered) {
            this.answered = answered;
        }

        @DynamoDBAttribute(attributeName = "Views")
        public int getViews() {
            return views;
        }

        public void setViews(int views) {
            this.views = views;
        }

        @DynamoDBAttribute(attributeName = "Replies")
        public int getReplies() {
            return replies;
        }

        public void setReplies(int replies) {
            this.replies = replies;
        }

    }

    @DynamoDBTable(tableName = "Forum")
    public static class Forum {
        private String name;
        private String category;
        private int threads;

        // Partition key
        @DynamoDBHashKey(attributeName = "Name")
        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        @DynamoDBAttribute(attributeName = "Category")
        public String getCategory() {
            return category;
        }

        public void setCategory(String category) {
            this.category = category;
        }

        @DynamoDBAttribute(attributeName = "Threads")
        public int getThreads() {
            return threads;
        }

        public void setThreads(int threads) {
            this.threads = threads;
        }
    }
}
```

DynamoDBMapper トランザクションオペレーション

次の Java サンプルコードでは、Forum および Thread クラスが宣言され、それらは DynamoDBMapper クラスにより、DynamoDB テーブルにマッピングされています。

このコードは、次のトランザクションオペレーションを示しています。

- transactionWrite で 1 つのトランザクションで 1 つまたは複数のテーブルから複数の項目を追加、更新、削除します。
- 1 つのトランザクションで 1 つまたは複数のテーブルから複数の項目を取得する transactionLoad

インポート

```
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMappingException;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import
 com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTransactionLoadExpression;
import
 com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTransactionWriteExpression;
import com.amazonaws.services.dynamodbv2.datamodeling.TransactionLoadRequest;
import com.amazonaws.services.dynamodbv2.datamodeling.TransactionWriteRequest;
import com.amazonaws.services.dynamodbv2.model.InternalServerErrorException;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import com.amazonaws.services.dynamodbv2.model.TransactionCanceledException;
```

Code

```
public class DynamoDBMapperTransactionExample {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDBMapper mapper;

    public static void main(String[] args) throws Exception {
        try {

            mapper = new DynamoDBMapper(client);

            testPutAndUpdateInTransactionWrite();
            testPutWithConditionalUpdateInTransactionWrite();
            testPutWithConditionCheckInTransactionWrite();
            testMixedOperationsInTransactionWrite();
            testTransactionLoadWithSave();
            testTransactionLoadWithTransactionWrite();
            System.out.println("Example complete");

        } catch (Throwable t) {
            System.err.println("Error running the 
 DynamoDBMapperTransactionWriteExample: " + t);
            t.printStackTrace();
        }
    }

    private static void testTransactionLoadWithSave() {
        // Create new Forum item for DynamoDB using save
        Forum dynamodbForum = new Forum();
        dynamodbForum.setName("DynamoDB Forum");
        dynamodbForum.setCategory("Amazon Web Services");
        dynamodbForum.setThreads(0);
        mapper.save(dynamodbForum);

        // Add a thread to DynamoDB Forum
        Thread dynamodbForumThread = new Thread();
        dynamodbForumThread.setForumName("DynamoDB Forum");
        dynamodbForumThread.setSubject("Sample Subject 1");
        dynamodbForumThread.setMessage("Sample Question 1");
        mapper.save(dynamodbForumThread);

        // Update DynamoDB Forum to reflect updated thread count
        dynamodbForum.setThreads(1);
        mapper.save(dynamodbForum);

        // Read DynamoDB Forum item and Thread item at the same time in a serializable
```

```
       // manner
       TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();

       // Read entire item for DynamoDB Forum
       transactionLoadRequest.addLoad(dynamodbForum);

       // Only read subject and message attributes from Thread item
       DynamoDBTransactionLoadExpression loadExpressionForThread = new
DynamoDBTransactionLoadExpression()
               .withProjectionExpression("Subject, Message");
       transactionLoadRequest.addLoad(dynamodbForumThread, loadExpressionForThread);

       // Loaded objects are guaranteed to be in same order as the order in which they
       // are
       // added to TransactionLoadRequest
       List<Object> loadedObjects = executeTransactionLoad(transactionLoadRequest);
       Forum loadedDynamoDBForum = (Forum) loadedObjects.get(0);
       System.out.println("Forum: " + loadedDynamoDBForum.getName());
       System.out.println("Threads: " + loadedDynamoDBForum.getThreads());
       Thread loadedDynamodbForumThread = (Thread) loadedObjects.get(1);
       System.out.println("Subject: " + loadedDynamodbForumThread.getSubject());
       System.out.println("Message: " + loadedDynamodbForumThread.getMessage());
   }

   private static void testTransactionLoadWithTransactionWrite() {
       // Create new Forum item for DynamoDB using save
       Forum dynamodbForum = new Forum();
       dynamodbForum.setName("DynamoDB New Forum");
       dynamodbForum.setCategory("Amazon Web Services");
       dynamodbForum.setThreads(0);
       mapper.save(dynamodbForum);

       // Update Forum item for DynamoDB and add a thread to DynamoDB Forum, in
       // an ACID manner using transactionWrite

       dynamodbForum.setThreads(1);
       Thread dynamodbForumThread = new Thread();
       dynamodbForumThread.setForumName("DynamoDB New Forum");
       dynamodbForumThread.setSubject("Sample Subject 2");
       dynamodbForumThread.setMessage("Sample Question 2");
       TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
       transactionWriteRequest.addPut(dynamodbForumThread);
       transactionWriteRequest.addUpdate(dynamodbForum);
       executeTransactionWrite(transactionWriteRequest);

       // Read DynamoDB Forum item and Thread item at the same time in a serializable
       // manner
       TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();

       // Read entire item for DynamoDB Forum
       transactionLoadRequest.addLoad(dynamodbForum);

       // Only read subject and message attributes from Thread item
       DynamoDBTransactionLoadExpression loadExpressionForThread = new
DynamoDBTransactionLoadExpression()
               .withProjectionExpression("Subject, Message");
       transactionLoadRequest.addLoad(dynamodbForumThread, loadExpressionForThread);

       // Loaded objects are guaranteed to be in same order as the order in which they
       // are
       // added to TransactionLoadRequest
       List<Object> loadedObjects = executeTransactionLoad(transactionLoadRequest);
       Forum loadedDynamoDBForum = (Forum) loadedObjects.get(0);
       System.out.println("Forum: " + loadedDynamoDBForum.getName());
       System.out.println("Threads: " + loadedDynamoDBForum.getThreads());
       Thread loadedDynamodbForumThread = (Thread) loadedObjects.get(1);
       System.out.println("Subject: " + loadedDynamodbForumThread.getSubject());
       System.out.println("Message: " + loadedDynamodbForumThread.getMessage());
   }

   private static void testPutAndUpdateInTransactionWrite() {
       // Create new Forum item for S3 using save
       Forum s3Forum = new Forum();
       s3Forum.setName("S3 Forum");
       s3Forum.setCategory("Core Amazon Web Services");
       s3Forum.setThreads(0);
       mapper.save(s3Forum);

       // Update Forum item for S3 and Create new Forum item for DynamoDB using
       // transactionWrite
       s3Forum.setCategory("Amazon Web Services");
       Forum dynamodbForum = new Forum();
       dynamodbForum.setName("DynamoDB Forum");
       dynamodbForum.setCategory("Amazon Web Services");
       dynamodbForum.setThreads(0);
       TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
```

```
       transactionWriteRequest.addUpdate(s3Forum);
       transactionWriteRequest.addPut(dynamodbForum);
       executeTransactionWrite(transactionWriteRequest);
   }

   private static void testPutWithConditionalUpdateInTransactionWrite() {
       // Create new Thread item for DynamoDB forum and update thread count in
DynamoDB
       // forum
       // if the DynamoDB Forum exists
       Thread dynamodbForumThread = new Thread();
       dynamodbForumThread.setForumName("DynamoDB Forum");
       dynamodbForumThread.setSubject("Sample Subject 1");
       dynamodbForumThread.setMessage("Sample Question 1");

       Forum dynamodbForum = new Forum();
       dynamodbForum.setName("DynamoDB Forum");
       dynamodbForum.setCategory("Amazon Web Services");
       dynamodbForum.setThreads(1);

       DynamoDBTransactionWriteExpression transactionWriteExpression = new
DynamoDBTransactionWriteExpression()
               .withConditionExpression("attribute_exists(Category)");

       TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
       transactionWriteRequest.addPut(dynamodbForumThread);
       transactionWriteRequest.addUpdate(dynamodbForum, transactionWriteExpression);
       executeTransactionWrite(transactionWriteRequest);
   }

   private static void testPutWithConditionCheckInTransactionWrite() {
       // Create new Thread item for DynamoDB forum and update thread count in
DynamoDB
       // forum if a thread already exists
       Thread dynamodbForumThread2 = new Thread();
       dynamodbForumThread2.setForumName("DynamoDB Forum");
       dynamodbForumThread2.setSubject("Sample Subject 2");
       dynamodbForumThread2.setMessage("Sample Question 2");

       Thread dynamodbForumThread1 = new Thread();
       dynamodbForumThread1.setForumName("DynamoDB Forum");
       dynamodbForumThread1.setSubject("Sample Subject 1");
       DynamoDBTransactionWriteExpression conditionExpressionForConditionCheck = new
DynamoDBTransactionWriteExpression()
               .withConditionExpression("attribute_exists(Subject)");

       Forum dynamodbForum = new Forum();
       dynamodbForum.setName("DynamoDB Forum");
       dynamodbForum.setCategory("Amazon Web Services");
       dynamodbForum.setThreads(2);

       TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
       transactionWriteRequest.addPut(dynamodbForumThread2);
       transactionWriteRequest.addConditionCheck(dynamodbForumThread1,
conditionExpressionForConditionCheck);
       transactionWriteRequest.addUpdate(dynamodbForum);
       executeTransactionWrite(transactionWriteRequest);
   }

   private static void testMixedOperationsInTransactionWrite() {
       // Create new Thread item for S3 forum and delete "Sample Subject 1" Thread
from
       // DynamoDB forum if
       // "Sample Subject 2" Thread exists in DynamoDB forum
       Thread s3ForumThread = new Thread();
       s3ForumThread.setForumName("S3 Forum");
       s3ForumThread.setSubject("Sample Subject 1");
       s3ForumThread.setMessage("Sample Question 1");

       Forum s3Forum = new Forum();
       s3Forum.setName("S3 Forum");
       s3Forum.setCategory("Amazon Web Services");
       s3Forum.setThreads(1);

       Thread dynamodbForumThread1 = new Thread();
       dynamodbForumThread1.setForumName("DynamoDB Forum");
       dynamodbForumThread1.setSubject("Sample Subject 1");

       Thread dynamodbForumThread2 = new Thread();
       dynamodbForumThread2.setForumName("DynamoDB Forum");
       dynamodbForumThread2.setSubject("Sample Subject 2");
       DynamoDBTransactionWriteExpression conditionExpressionForConditionCheck = new
DynamoDBTransactionWriteExpression()
               .withConditionExpression("attribute_exists(Subject)");
```

```
        Forum dynamodbForum = new Forum();
        dynamodbForum.setName("DynamoDB Forum");
        dynamodbForum.setCategory("Amazon Web Services");
        dynamodbForum.setThreads(1);

        TransactionWriteRequest transactionWriteRequest = new
TransactionWriteRequest();
        transactionWriteRequest.addPut(s3ForumThread);
        transactionWriteRequest.addUpdate(s3Forum);
        transactionWriteRequest.addDelete(dynamodbForumThread1);
        transactionWriteRequest.addConditionCheck(dynamodbForumThread2,
conditionExpressionForConditionCheck);
        transactionWriteRequest.addUpdate(dynamodbForum);
        executeTransactionWrite(transactionWriteRequest);
    }

    private static List<Object> executeTransactionLoad(TransactionLoadRequest
transactionLoadRequest) {
        List<Object> loadedObjects = new ArrayList<Object>();
        try {
            loadedObjects = mapper.transactionLoad(transactionLoadRequest);
        } catch (DynamoDBMappingException ddbme) {
            System.err.println("Client side error in Mapper, fix before retrying.
Error: " + ddbme.getMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.err.println("One of the tables was not found, verify table exists
before retrying. Error: "
                    + rnfe.getMessage());
        } catch (InternalServerErrorException ise) {
            System.err.println(
                    "Internal Server Error, generally safe to retry with back-off.
Error: " + ise.getMessage());
        } catch (TransactionCanceledException tce) {
            System.err.println(
                    "Transaction Canceled, implies a client issue, fix before retrying.
Error: " + tce.getMessage());
        } catch (Exception ex) {
            System.err.println(
                    "An exception occurred, investigate and configure retry strategy.
Error: " + ex.getMessage());
        }
        return loadedObjects;
    }
```

```
   private static void executeTransactionWrite(TransactionWriteRequest
transactionWriteRequest) {
       try {
           mapper.transactionWrite(transactionWriteRequest);
       } catch (DynamoDBMappingException ddbme) {
           System.err.println("Client side error in Mapper, fix before retrying.
Error: " + ddbme.getMessage());
       } catch (ResourceNotFoundException rnfe) {
           System.err.println("One of the tables was not found, verify table exists
before retrying. Error: "
                   + rnfe.getMessage());
       } catch (InternalServerErrorException ise) {
           System.err.println(
                   "Internal Server Error, generally safe to retry with back-off.
Error: " + ise.getMessage());
       } catch (TransactionCanceledException tce) {
           System.err.println(
                   "Transaction Canceled, implies a client issue, fix before retrying.
Error: " + tce.getMessage());
       } catch (Exception ex) {
           System.err.println(
                   "An exception occurred, investigate and configure retry strategy.
Error: " + ex.getMessage());
       }
   }

   @DynamoDBTable(tableName = "Thread")
   public static class Thread {
       private String forumName;
       private String subject;
       private String message;
       private String lastPostedDateTime;
       private String lastPostedBy;
       private Set<String> tags;
       private int answered;
       private int views;
       private int replies;

       // Partition key
       @DynamoDBHashKey(attributeName = "ForumName")
       public String getForumName() {
           return forumName;
       }
```

```
        public void setForumName(String forumName) {
            this.forumName = forumName;
        }

        // Sort key
        @DynamoDBRangeKey(attributeName = "Subject")
        public String getSubject() {
            return subject;
        }

        public void setSubject(String subject) {
            this.subject = subject;
        }

        @DynamoDBAttribute(attributeName = "Message")
        public String getMessage() {
            return message;
        }

        public void setMessage(String message) {
            this.message = message;
        }

        @DynamoDBAttribute(attributeName = "LastPostedDateTime")
        public String getLastPostedDateTime() {
            return lastPostedDateTime;
        }

        public void setLastPostedDateTime(String lastPostedDateTime) {
            this.lastPostedDateTime = lastPostedDateTime;
        }

        @DynamoDBAttribute(attributeName = "LastPostedBy")
        public String getLastPostedBy() {
            return lastPostedBy;
        }

        public void setLastPostedBy(String lastPostedBy) {
            this.lastPostedBy = lastPostedBy;
        }

        @DynamoDBAttribute(attributeName = "Tags")
        public Set<String> getTags() {
            return tags;
        }

        public void setTags(Set<String> tags) {
            this.tags = tags;
        }

        @DynamoDBAttribute(attributeName = "Answered")
        public int getAnswered() {
            return answered;
        }

        public void setAnswered(int answered) {
            this.answered = answered;
        }

        @DynamoDBAttribute(attributeName = "Views")
        public int getViews() {
            return views;
        }

        public void setViews(int views) {
            this.views = views;
        }

        @DynamoDBAttribute(attributeName = "Replies")
        public int getReplies() {
            return replies;
        }

        public void setReplies(int replies) {
            this.replies = replies;
        }

    }

    @DynamoDBTable(tableName = "Forum")
    public static class Forum {
        private String name;
        private String category;
        private int threads;

        // Partition key
        @DynamoDBHashKey(attributeName = "Name")
        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        @DynamoDBAttribute(attributeName = "Category")
        public String getCategory() {
            return category;
        }

        public void setCategory(String category) {
            this.category = category;
        }

        @DynamoDBAttribute(attributeName = "Threads")
        public int getThreads() {
            return threads;
        }

        public void setThreads(int threads) {
            this.threads = threads;
        }
    }
}
```

Java 2.x: DynamoDB 拡張クライアント

DynamoDB 拡張クライアントは、AWS SDK for Java バージョン 2 (v2) の一部である高レベルのライブラリです。クライアント側のクラスを DynamoDB テーブルにマッピングする簡単な方法を提供します。テーブルおよび対応するモデルクラスの間の関係をコードで定義します。関係を定義したら、DynamoDB のテーブルまたは項目に対して、さまざまな作成、読み取り、更新、または削除 (CRUD) オペレーションを直感的に実行できます。

DynamoDB で拡張クライアントを使用する方法の詳細については、「AWS SDK for Java 2.x での DynamoDB 拡張クライアントの使用」を参照してください。

.NET ドキュメントモデル

AWS SDK for .NET では、低レベル Amazon DynamoDB オペレーションの一部をまとめるドキュメントモデルクラスを使用して、コーディングをさらに簡略化することができま

す。ドキュメントモデルのプライマリクラスは Table と Document です。Table クラスでは、PutItem、GetItem、DeleteItem などのデータオペレーション方法を使用できます。Query および Scan メソッドも使用できます。Document クラスは、テーブル内の単一の項目を表します。

前述のドキュメントモデルクラスは、Amazon.DynamoDBv2.DocumentModel 名前空間で使用できます。

> Note
>
> ドキュメントモデルクラスは、テーブルの作成、更新、削除に使用することはできません。ただし、ドキュメントモデルは、ほとんどの一般的なデータオペレーションをサポートしています。

トピック

- サポートされている データ型
- AWS SDK for .NET ドキュメントモデルを使用して DynamoDB の項目の操作します
- 例: AWS SDK for .NET ドキュメントモデルを使用した CRUD オペレーション
- 例: AWS SDK for .NET ドキュメントモデル API を使用したバッチオペレーション
- AWS SDK for .NET ドキュメントモデルを使用した DynamoDB のテーブルの操作

サポートされている データ型

ドキュメントモデルは、プリミティブ .NET データ型とコレクションデータ型のセットをサポートします。このモデルでは、次のプリミティブデータ型がサポートされています。

- bool
- byte
- char
- DateTime
- decimal
- double
- float

- `Guid`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

次の表に、前述の .NET 型が DynamoDB の型にどのようにマッピングされるのかをまとめています。

.NET プリミティブ型	DynamoDB 型
すべての数値型	N（数値型）
すべての文字列型	S (文字列型)
MemoryStream、byte[]	B（バイナリ型）
ブール	N (数値型)。0 は false、1 は true を表します。
DateTime	S (文字列型)。`DateTime` の値は、ISO-8601 形式の文字列として格納されます。
Guid	S (文字列型)。
コレクション型 (リスト、ハッシュセット、配列)	BS（バイナリセット）型、SS（文字列セット）型、NS（数値セット）型

AWS SDK for .NET は、DynamoDB のブール型、null 型、リスト型、およびマップ型を .NET ドキュメントモデル API にマッピングするための型を定義します。

- ブール型には `DynamoDBBool` 使用します。
- null 型には `DynamoDBNull` を使用します。

- リスト型には DynamoDBList を使用します。
- マップ型には Document を使用します。

Note

- 空のバイナリ値がサポートされています。
- 空の文字列値の読み取りがサポートされています。空の文字列属性値は、DynamoDB への書き込み中に、文字列セット型の属性値内でサポートされます。文字列型の空の文字列属性値と、リスト型またはマップ型に含まれる空の文字列値が書き込みリクエストから削除されます。

AWS SDK for .NET ドキュメントモデルを使用して DynamoDB の項目の操作します

以下のコード例は、AWS SDK for .NET ドキュメントモデルを使用してさまざまなオペレーションを実行する方法を示しています。これらの例を使用して、CRUD、バッチ、トランザクションの各オペレーションを実行できます。

トピック

ドキュメントモデルを使用してデータオペレーションを実行するには、最初に Table.LoadTable メソッドを呼び出して、特定のテーブルを表す Table クラスのインスタンスを作成します。次の C# の例では、Amazon DynamoDB の ProductCatalog テーブルに対応している Table オブジェクトを作成しています。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
```

> Note
>
> 通常、LoadTable メソッドはアプリケーションの開始時に 1 回使用するだけです。このメソッドによって、DynamoDB へのラウンドトリップに DescribeTable 呼び出しが追加されるためです。

その後、Table オブジェクトを使用して、さまざまなデータオペレーションを実行することができます。各データオペレーションには 2 種類のオーバーロードがあります。1 つは最小限必要なパラメーターを受け取り、もう 1 つはオプションのオペレーション固有の設定情報を受け取ります。たとえば項目を取得するには、テーブルのプライマリキーの値を入力する必要がありますが、その場合は次の GetItem オーバーロードを使用できます。

Example

```
// Get the item from a table that has a primary key that is composed of only a
 partition key.
Table.GetItem(Primitive partitionKey);
// Get the item from a table whose primary key is composed of both a partition key and
 sort key.
Table.GetItem(Primitive partitionKey, Primitive sortKey);
```

これらのメソッドには、オプションのパラメータを渡すこともできます。たとえば前述の GetItem では、すべての属性を含む項目全体が返されます。オプションで、取り出す属性のリストを指定することもできます。この場合は、オペレーション固有の設定オブジェクトパラメータをとる、次の GetItem オーバーロードを使用します。

Example

```


// Configuration object that specifies optional parameters.
GetItemOperationConfig config = new GetItemOperationConfig()
{
  AttributesToGet = new List<string>() { "Id", "Title" },
};
// Pass in the configuration to the GetItem method.
// 1. Table that has only a partition key as primary key.
Table.GetItem(Primitive partitionKey, GetItemOperationConfig config);
// 2. Table that has both a partition key and a sort key.
```

```
Table.GetItem(Primitive partitionKey, Primitive sortKey, GetItemOperationConfig
 config);
```

設定オブジェクトを使用することで、特定の属性リストのリクエストやページサイズ（ページあたりの項目数）の指定など、複数のオプションパラメータを指定できます。それぞれのデータオペレーションメソッドには、独自の設定クラスがあります。たとえば、GetItemOperationConfig クラスを使用して、GetItem オペレーションのオプションを指定します。PutItemOperationConfig クラスを使用して、PutItem オペレーション用にオプションのパラメータを指定することができます。

以下のセクションでは、Table クラスでサポートされているそれぞれのデータオペレーションについて説明します。

項目の入力 – Table.PutItem メソッド

PutItem メソッドは、入力された Document インスタンスをテーブルにアップロードします。入力された Document で指定されているプライマリキーを持つ項目がテーブル内に存在する場合は、PutItem オペレーションによって、既存の項目全体が置換されます。新しい項目は、Document メソッドに指定した PutItem オブジェクトと同じになります。元の項目にそれ以外の属性が存在していたとしても、新しい項目には存在しません。

次に、AWS SDK for .NET ドキュメントモデルを使用してテーブルに新しい項目を入力するステップを示します。

1. 項目を追加するテーブル名を生成するために、Table.LoadTable メソッドを実行します。
2. 属性名と値のリストが含まれる、Document オブジェクトを作成します。
3. Document インスタンスをパラメータとして指定しながら Table.PutItem を実行します。

以下の C# コード例は、前述のタスクの例です。この例では、項目を ProductCatalog テーブルにアップロードします。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
book["Id"] = 101;
book["Title"] = "Book 101 Title";
```

```
book["ISBN"] = "11-11-11-11";
book["Authors"] = new List<string> { "Author 1", "Author 2" };
book["InStock"] = new DynamoDBBool(true);
book["QuantityOnHand"] = new DynamoDBNull();

table.PutItem(book);
```

前述の例では、Document インスタンスで、Number、String、String Set、Boolean、および Null 属性を含む項目が作成されます。(Null は、この製品の QuantityOnHand が不明なことを示す場合に使用されます) Booleanと Null の場合は、コンストラクタメソッド DynamoDBBool および DynamoDBNull を使用します。

DynamoDB では、データ型 List および Map に、他のデータ型で構成された要素を含むこができます。これらのデータ型をドキュメントモデル API にマッピングする方法を以下に示します。

- List – DynamoDBList コンストラクタを使用します。
- Map – Document コンストラクタを使用します。

項目に List 属性を追加するように、前述の例を変更することができます。これを行うには、次のサンプルコードに示すように、DynamoDBList コンストラクタを使用します。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
book["Id"] = 101;

/*other attributes omitted for brevity...*/

var relatedItems = new DynamoDBList();
relatedItems.Add(341);
relatedItems.Add(472);
relatedItems.Add(649);
book.Add("RelatedItems", relatedItems);

table.PutItem(book);
```

書籍に Map 属性を追加するには、別の Document を定義します。次のコード例はこれを行う方法を示しています。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
book["Id"] = 101;

/*other attributes omitted for brevity...*/

var pictures = new Document();
pictures.Add("FrontView", "http://example.com/products/101_front.jpg" );
pictures.Add("RearView", "http://example.com/products/101_rear.jpg" );

book.Add("Pictures", pictures);

table.PutItem(book);
```

これらの例は、式を使用する時の項目属性の指定 に示された項目に基づいています。ドキュメントモデルを使用すると、導入事例の ProductReviews 属性のように複雑な入れ子になった属性を作成することができます。

オプションパラメータの指定

PutItem オペレーションに PutItemOperationConfig パラメータを追加することで、オプションパラメータを設定できます。すべてのオプションパラメータのリストは、「PutItem」でご確認ください。次の C# のサンプルコードでは、ProductCatalog テーブルに項目を配置します。次のオプションパラメータが指定されています。

- 条件付き入力リクエストとする ConditionalExpression パラメータ。例では、置き換える項目に ISBN 属性が存在し、それが特定の値でなくてはならないことを指定する式を作成します。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();
book["Id"] = 555;
book["Title"] = "Book 555 Title";
book["Price"] = "25.00";
book["ISBN"] = "55-55-55-55";
book["Name"] = "Item 1 updated";
```

```
book["Authors"] = new List<string> { "Author x", "Author y" };
book["InStock"] = new DynamoDBBool(true);
book["QuantityOnHand"] = new DynamoDBNull();

// Create a condition expression for the optional conditional put operation.
Expression expr = new Expression();
expr.ExpressionStatement = "ISBN = :val";
expr.ExpressionAttributeValues[":val"] = "55-55-55-55";

PutItemOperationConfig config = new PutItemOperationConfig()
{
  // Optional parameter.
     ConditionalExpression = expr
};

table.PutItem(book, config);
```

項目の取得 – Table.GetItem

GetItem オペレーションでは、項目を Document インスタンスとして取り出します。次の C# サンプルコードに示すように、取得する項目のプライマリキーを入力する必要があります。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
Document document = table.GetItem(101); // Primary key 101.
```

GetItem オペレーションを実行すると、項目のすべての属性が返され、デフォルトで結果整合性のある読み込み (「読み込み整合性」を参照) が実行されます。

オプションパラメータの指定

GetItem パラメータを追加することで、GetItemOperationConfig オペレーションに追加オプションを設定できます。すべてのオプションパラメータのリストは、「GetItem」でご確認ください。次の C# のサンプルコードでは、ProductCatalog から項目を取得します。次のオプションパラメータを使用して GetItemOperationConfig が指定されています。

- 指定された属性だけを取り出す AttributesToGet パラメータ。
- 指定されたすべての属性に対する最新の値をリクエストする ConsistentRead パラメータ データの整合性の詳細については、「読み込み整合性」を参照してください。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

GetItemOperationConfig config = new GetItemOperationConfig()
{
  AttributesToGet = new List<string>() { "Id", "Title", "Authors", "InStock",
 "QuantityOnHand" },
  ConsistentRead = true
};
Document doc = table.GetItem(101, config);
```

ドキュメントモデル API を使用して項目を取得すると、以下の例に示されるように、返る `Document` オブジェクト内の個々の要素にアクセスできます。

Example

```
int id = doc["Id"].AsInt();
string title = doc["Title"].AsString();
List<string> authors = doc["Authors"].AsListOfString();
bool inStock = doc["InStock"].AsBoolean();
DynamoDBNull quantityOnHand = doc["QuantityOnHand"].AsDynamoDBNull();
```

`List` 型または `Map` 型の属性の場合に、属性をドキュメントモデル API にマッピングする方法を以下に示します。

- `List` – `AsDynamoDBList` メソッドを使用します。
- `Map` – `AsDocument` メソッドを使用します。

次のサンプルコードは、`List` (RelatedItems) および `Map` (Pictures) を `Document` オブジェクトから取得する方法を示します。

Example

```
DynamoDBList relatedItems = doc["RelatedItems"].AsDynamoDBList();

Document pictures = doc["Pictures"].AsDocument();
```

項目の削除 – Table.DeleteItem

DeleteItem オペレーションは、テーブルから項目を削除します。項目のプライマリキーをパラメータとして渡すことができます。また、以下の C# のサンプルコードに示されているように、すでに項目を読み取っていて対応する Document オブジェクトを取得している場合には、それをパラメータとして DeleteItem メソッドに渡すこともできます。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

// Retrieve a book (a Document instance)
Document document = table.GetItem(111);

// 1) Delete using the Document instance.
 table.DeleteItem(document);

// 2) Delete using the primary key.
int partitionKey = 222;
table.DeleteItem(partitionKey)
```

オプションパラメータの指定

Delete パラメータを追加することで、DeleteItemOperationConfig オペレーションに追加オプションを設定できます。すべてのオプションパラメータのリストは、「DeleteTable」でご確認ください。次の C# サンプルコードでは、以下の 2 つのオプションパラメータを指定します。

- ISBN 属性に特定の値が含まれている書籍項目が削除されるようにする ConditionalExpression パラメータ。
- 削除された項目が ReturnValues メソッドによって返されることをリクエストする Delete パラメータ。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
int partitionKey = 111;

Expression expr = new Expression();
expr.ExpressionStatement = "ISBN = :val";
expr.ExpressionAttributeValues[":val"] = "11-11-11-11";
```

```
// Specify optional parameters for Delete operation.
DeleteItemOperationConfig config = new DeleteItemOperationConfig
{
     ConditionalExpression = expr,
     ReturnValues = ReturnValues.AllOldAttributes // This is the only supported value
 when using the document model.
};

// Delete the book.
Document d = table.DeleteItem(partitionKey, config);
```

項目の更新 – Table.UpdateItem

`UpdateItem` オペレーションは、既存の項目があればそれを更新します。プライマリキーが指定されている項目がない場合は、`UpdateItem` オペレーションによって新しい項目が追加されます。

`UpdateItem` オペレーションを使用して、既存の属性値を更新するか、既存のコレクションに新しい属性を追加するか、既存のコレクションから属性を削除することができます。実行する更新を記述する `Document` インスタンスを作成して、これらの更新を提供します。

`UpdateItem` アクションは、以下のガイドラインに従います。

- 項目が存在しない場合、`UpdateItem` は入力で指定されたプライマリキーを使用して、新しい項目を追加します。
- 項目が存在する場合、`UpdateItem` は次のように更新を適用します。
  - 既存の属性値を更新に含まれる値に置き換えます。
  - 入力で指定された属性が存在しない場合は、新しい属性を項目に追加します。
  - 入力された属性値が null である場合は、属性を削除します。

> **Note**
>
> この中間レベルの `UpdateItem` オペレーションでは、基盤となる DynamoDB オペレーションでサポートされている `Add` アクション (「UpdateItem」を参照) はサポートされていません。

> **Note**
>
> PutItem オペレーション (項目の入力 – Table.PutItem メソッド) により、更新を実行できます。PutItem を呼び出して項目をアップロードするときにプライマリキーが存在する場合は、PutItem オペレーションによって項目全体が置き換わります。既存の項目内に属性があり、入力された Document でそれらの属性が指定されていない場合、それらの属性は、PutItem オペレーションによって削除されます。ただし、UpdateItem が更新するのは指定された入力属性だけです。その項目では、その他の既存の属性は変更されません。

AWS SDK for .NET ドキュメントモデルを使用して項目を更新するステップは次のとおりです。

1. 更新オペレーションを実行するテーブルの名前を指定しながら、Table.LoadTable メソッドを実行します。
2. 実行するすべての更新を指定して、Document インスタンスを作成します。

   既存の属性を削除するには、その属性値に null を指定します。
3. Table.UpdateItem メソッドを呼び出し、Document インスタンスを入力パラメータとして指定します。

   プライマリキーを、Document インスタンスで、または明示的にパラメータとして指定する必要があります。

以下の C# コード例は、前述のタスクの例です。このサンプルコードでは、Book テーブルの項目を更新します。UpdateItem オペレーションによって、既存の Authors 属性が更新され、PageCount 属性が削除され、新しい属性 XYZ が追加されます。Document インスタンスには、更新する書籍のプライマリキーが含まれています。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");

var book = new Document();

// Set the attributes that you wish to update.
book["Id"] = 111; // Primary key.
// Replace the authors attribute.
book["Authors"] = new List<string> { "Author x", "Author y" };
```

```
// Add a new attribute.
book["XYZ"] = 12345;
// Delete the existing PageCount attribute.
book["PageCount"] = null;

table.Update(book);
```

オプションパラメータの指定

UpdateItem パラメータを追加することで、UpdateItemOperationConfig オペレーションに追加オプションを設定できます。すべてのオプションパラメータのリストは、「UpdateItem」でご確認ください。

次の C# サンプルコードでは、書籍項目の価格が 25 に更新されています。次の 2 つのオプションパラメータが指定されています。

- 存在すると予測される ConditionalExpression 属性の値が Price であることを指定する 20 パラメータ。
- 更新された項目が返される ReturnValues オペレーションをリクエストする UpdateItem パラメータ。

Example

```
Table table = Table.LoadTable(client, "ProductCatalog");
string partitionKey = "111";

var book = new Document();
book["Id"] = partitionKey;
book["Price"] = 25;

Expression expr = new Expression();
expr.ExpressionStatement = "Price = :val";
expr.ExpressionAttributeValues[":val"] = "20";

UpdateItemOperationConfig config = new UpdateItemOperationConfig()
{
    ConditionalExpression = expr,
    ReturnValues = ReturnValues.AllOldAttributes
};

Document d1 = table.Update(book, config);
```

バッチ書き込み – 複数の項目の書き込みおよび削除

バッチ書き込みは、複数の項目の書き込みと削除をバッチで行うことを意味します。このオペレーションでは、単一の 呼び出しの 1 つ以上のテーブルから、複数の項目の書き込みと削除を行うことができます。AWS SDK for .NET ドキュメントモデル API を使用して、テーブルで複数の項目の書き込みまたは削除を行うステップを次に示します。

1. `Table` オブジェクトを作成するには、バッチオペレーションを実行するテーブルの名前を指定して、`Table.LoadTable` メソッドを実行します。
2. 前のステップで作成したテーブルインスタンスで `createBatchWrite` メソッドを実行し、`DocumentBatchWrite` オブジェクトを作成します。
3. `DocumentBatchWrite` オブジェクトメソッドを使用して、アップロードまたは削除するドキュメントを指定します。
4. `DocumentBatchWrite.Execute` メソッドを呼び出してバッチオペレーションを実行します。

   ドキュメントモデル API を使用する場合は、1 つのバッチに任意の数のオペレーションを指定できます。ただし DynamoDB では、1 つのバッチ内のオペレーションの数と、1 つのバッチオペレーションでのバッチの合計サイズが制限されています。具体的な制限事項については、「BatchWriteItem」を参照してください。許可されている書き込みリクエスト数をバッチ書き込みリクエストが超えたこと、またはバッチの HTTP ペイロードサイズが `BatchWriteItem` で許可されている制限を超えたことをドキュメントモデル API が検出した場合には、バッチが複数の小さなバッチに分割されます。さらに、バッチ書き込みに対する応答で、未処理の項目が返された場合には、ドキュメントモデル API がそれら未処理の項目を使用して、別のバッチリクエストを自動的に送信します。

以下の C# サンプルコードは、前述のステップの例です。この例では、バッチ書き込みオペレーションを使用して、書籍項目のアップロードと別の書籍項目の削除という 2 つの書き込みが実行されています。

```
Table productCatalog = Table.LoadTable(client, "ProductCatalog");
var batchWrite = productCatalog.CreateBatchWrite();

var book1 = new Document();
book1["Id"] = 902;
book1["Title"] = "My book1 in batch write using .NET document model";
book1["Price"] = 10;
book1["Authors"] = new List<string> { "Author 1", "Author 2", "Author 3" };
```

```
book1["InStock"] = new DynamoDBBool(true);
book1["QuantityOnHand"] = 5;

batchWrite.AddDocumentToPut(book1);
// specify delete item using overload that takes PK.
batchWrite.AddKeyToDelete(12345);

batchWrite.Execute();
```

実例については、「例: AWS SDK for .NET ドキュメントモデル API を使用したバッチオペレーション」を参照してください。

batchWrite オペレーションを使用すると、複数のテーブルで入力および削除オペレーションを実行できます。AWS SDK for .NET ドキュメントモデルを使用して、複数のテーブルで複数の項目の書き込みまたは削除を行うステップを次に示します。

1. 前述の手順で示したように、複数の項目の入力または削除を行う各テーブルについて DocumentBatchWrite インスタンスを作成します。
2. MultiTableDocumentBatchWrite のインスタンスを作成し、その中に個々の DocumentBatchWrite オブジェクトを追加します。
3. MultiTableDocumentBatchWrite.Execute メソッドを実行します。

以下の C# サンプルコードは、前述のステップの例です。この例では、バッチ書き込みオペレーションを使用して、次の書き込みオペレーションを実行しています。

- Forum テーブル項目内に新しい項目を入力します。
- Thread テーブル内に項目を入力し、同じテーブルから項目を削除します。

```
// 1. Specify item to add in the Forum table.
Table forum = Table.LoadTable(client, "Forum");
var forumBatchWrite = forum.CreateBatchWrite();

var forum1 = new Document();
forum1["Name"] = "Test BatchWrite Forum";
forum1["Threads"] = 0;
forumBatchWrite.AddDocumentToPut(forum1);
```

```
// 2a. Specify item to add in the Thread table.
Table thread = Table.LoadTable(client, "Thread");
var threadBatchWrite = thread.CreateBatchWrite();

var thread1 = new Document();
thread1["ForumName"] = "Amazon S3 forum";
thread1["Subject"] = "My sample question";
thread1["Message"] = "Message text";
thread1["KeywordTags"] = new List<string>{ "Amazon S3", "Bucket" };
threadBatchWrite.AddDocumentToPut(thread1);

// 2b. Specify item to delete from the Thread table.
threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

// 3. Create multi-table batch.
var superBatch = new MultiTableDocumentBatchWrite();
superBatch.AddBatch(forumBatchWrite);
superBatch.AddBatch(threadBatchWrite);

superBatch.Execute();
```

例: AWS SDK for .NET ドキュメントモデルを使用した CRUD オペレーション

次の C# コード例は、次のアクションを実行します。

- `ProductCatalog` テーブルに書籍項目を作成します。
- 書籍項目を取得します。
- 書籍項目を更新します。このコード例は、新しい属性を追加して既存の属性を更新する、通常の更新を示しています。また、既存の価格とコードで指定されている価格が一致する場合のみ書籍の価格を更新する、条件付き更新も示しています。
- 書籍項目を削除します。

次の例をテストするための詳しい手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class MidlevelItemCRUD
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "ProductCatalog";
        // The sample uses the following id PK value to add book item.
        private static int sampleBookId = 555;

        static void Main(string[] args)
        {
            try
            {
                Table productCatalog = Table.LoadTable(client, tableName);
                CreateBookItem(productCatalog);
                RetrieveBook(productCatalog);
                // Couple of sample updates.
                UpdateMultipleAttributes(productCatalog);
                UpdateBookPriceConditionally(productCatalog);

                // Delete.
                DeleteBook(productCatalog);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        // Creates a sample book item.
        private static void CreateBookItem(Table productCatalog)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document();
            book["Id"] = sampleBookId;
            book["Title"] = "Book " + sampleBookId;
            book["Price"] = 19.99;
            book["ISBN"] = "111-1111111111";
            book["Authors"] = new List<string> { "Author 1", "Author 2", "Author 3" };
            book["PageCount"] = 500;
            book["Dimensions"] = "8.5x11x.5";
            book["InPublication"] = new DynamoDBBool(true);
            book["InStock"] = new DynamoDBBool(false);
            book["QuantityOnHand"] = 0;

            productCatalog.PutItem(book);
        }

        private static void RetrieveBook(Table productCatalog)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");
            // Optional configuration.
            GetItemOperationConfig config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title", "Authors",
 "Price" },
                ConsistentRead = true
            };
            Document document = productCatalog.GetItem(sampleBookId, config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        private static void UpdateMultipleAttributes(Table productCatalog)
        {
            Console.WriteLine("\n*** Executing UpdateMultipleAttributes() ***");
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document();
            book["Id"] = partitionKey;
            // List of attribute updates.
            // The following replaces the existing authors list.
            book["Authors"] = new List<string> { "Author x", "Author y" };
            book["newAttribute"] = "New Value";
            book["ISBN"] = null; // Remove it.

            // Optional parameters.
            UpdateItemOperationConfig config = new UpdateItemOperationConfig
            {
                // Get updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes
            };
            Document updatedBook = productCatalog.UpdateItem(book, config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        private static void UpdateBookPriceConditionally(Table productCatalog)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally() ***");

            int partitionKey = sampleBookId;

            var book = new Document();
            book["Id"] = partitionKey;
            book["Price"] = 29.99;

            // For conditional price update, creating a condition expression.
            Expression expr = new Expression();
            expr.ExpressionStatement = "Price = :val";
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
            UpdateItemOperationConfig config = new UpdateItemOperationConfig
            {
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes
            };
            Document updatedBook = productCatalog.UpdateItem(book, config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose price
was conditionally updated");
            PrintDocument(updatedBook);
        }

        private static void DeleteBook(Table productCatalog)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");
            // Optional configuration.
            DeleteItemOperationConfig config = new DeleteItemOperationConfig
            {
                // Return the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes
            };
            Document document = productCatalog.DeleteItem(sampleBookId, config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");
            PrintDocument(document);
        }

        private static void PrintDocument(Document updatedDocument)
        {
            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];
                if (value is Primitive)
                    stringValue = value.AsPrimitive().Value.ToString();
                else if (value is PrimitiveList)
                    stringValue = string.Join(",", (from primitive
                                    in value.AsPrimitiveList().Entries
                                                    select primitive.Value).ToArray());
                Console.WriteLine("{0} - {1}", attribute, stringValue);
            }
        }
    }
}
```

例: AWS SDK for .NET ドキュメントモデル API を使用したバッチオペレーション

トピック

- 例: AWS SDK for .NET ドキュメントモデルを使用したバッチ書き込み

例: AWS SDK for .NET ドキュメントモデルを使用したバッチ書き込み

次の C# コード例は、単一のテーブルと複数のテーブルに対するバッチ書き込みオペレーションを示しています。この例では次のタスクを実行しています。

- 単一のテーブルバッチ書き込みを示します。ProductCatalog テーブルに 2 つの項目を追加します。
- 複数のテーブルバッチ書き込みを示します。項目 w2お Forum および Thread テーブルの両方に追加し、Thread から項目を削除します。

「DynamoDB でのコード例用のテーブルの作成とデータのロード」のステップに従っていれば、ProductCatalog、Forum、および Thread テーブルは作成済みです。これらのサンプルテーブルは、プログラムで作成することもできます。詳細については、「」を参照してくださいAWS SDK for .NET を使用してサンプルテーブルを作成してデータをアップロードする 以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class MidLevelBatchWriteItem
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        static void Main(string[] args)
        {
            try
            {
                SingleTableBatchWrite();
                MultiTableBatchWrite();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void SingleTableBatchWrite()
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document();
            book1["Id"] = 902;
            book1["Title"] = "My book1 in batch write using .NET helper classes";
            book1["ISBN"] = "902-11-11-1111";
            book1["Price"] = 10;
            book1["ProductCategory"] = "Book";
            book1["Authors"] = new List<string> { "Author 1", "Author 2", "Author 3" };
            book1["Dimensions"] = "8.5x11x.5";
            book1["InStock"] = new DynamoDBBool(true);
            book1["QuantityOnHand"] = new DynamoDBNull(); //Quantity is unknown at this
 time
```

```
            batchWrite.AddDocumentToPut(book1);
            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in SingleTableBatchWrite()");
            batchWrite.Execute();
        }

        private static void MultiTableBatchWrite()
        {
            // 1. Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document();
            forum1["Name"] = "Test BatchWrite Forum";
            forum1["Threads"] = 0;
            forumBatchWrite.AddDocumentToPut(forum1);


            // 2a. Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document();
            thread1["ForumName"] = "S3 forum";
            thread1["Subject"] = "My sample question";
            thread1["Message"] = "Message text";
            thread1["KeywordTags"] = new List<string> { "S3", "Bucket" };
            threadBatchWrite.AddDocumentToPut(thread1);

            // 2b. Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // 3. Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in MultiTableBatchWrite()");
            superBatch.Execute();
        }
    }
}
```

AWS SDK for .NET ドキュメントモデルを使用した DynamoDB のテーブルの操作

トピック

AWS SDK for .NET での Table.Query メソッド

Query メソッドを使用すると、テーブルのクエリを行うことができます。複合プライマリキー (パーティションキーおよびソートキー) があるテーブルにのみ、クエリを実行できます。テーブルのプライマリキーがパーティションキーだけで構成されている場合、Query オペレーションはサポートされていません。デフォルトでは、Query の内部で結果整合性のあるクエリが実行されます。整合性モデルの詳細については、「読み込み整合性」を参照してください。

Query メソッドには 2 つのオーバーロードがあります。Query メソッドで最小限必要なパラメーターは、パーティションキーの値とソートキーフィルタです。次のオーバーロードを使用して、これら最小限必要なパラメータを指定できます。

Example

```
Query(Primitive partitionKey, RangeFilter Filter);
```

たとえば次の C# コードでは、フォーラムで過去 15 日間に投稿されたすべての返信がクエリされます。

Example

```
string tableName = "Reply";
Table table = Table.LoadTable(client, tableName);

DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
RangeFilter filter = new RangeFilter(QueryOperator.GreaterThan, twoWeeksAgoDate);
Search search = table.Query("DynamoDB Thread 2", filter);
```

これによって Search オブジェクトが作成されます。次の C# サンプルコードに示すように、Search.GetNextSet メソッドを反復的に呼び出して、一度に 1 ページずつ結果を取得できるようになりました。このコードでは、クエリによって返される各項目の属性値が出力されます。

Example

```
List<Document> documentSet = new List<Document>();
do
{
  documentSet = search.GetNextSet();
  foreach (var document in documentSet)
    PrintDocument(document);
} while (!search.IsDone);

    private static void PrintDocument(Document document)
{
  Console.WriteLine();
  foreach (var attribute in document.GetAttributeNames())
  {
    string stringValue = null;
    var value = document[attribute];
    if (value is Primitive)
      stringValue = value.AsPrimitive().Value;
    else if (value is PrimitiveList)
      stringValue = string.Join(",", (from primitive
                                     in value.AsPrimitiveList().Entries
                                   select primitive.Value).ToArray());
    Console.WriteLine("{0} - {1}", attribute, stringValue);
  }
}
```

オプションパラメータの指定

また、取り出す属性リスト、強力な整合性のある読み込み、ページサイズ、ページごとに返される項目数などを指定する、Query のオプションパラメータを指定することもできます。すべてのパラメータのリストは、「クエリ」でご確認ください。オプションパラメータを指定するには、次のオーバーロードを使用して QueryOperationConfig オブジェクトを指定する必要があります。

Example

```
Query(QueryOperationConfig config);
```

前出の例でクエリを実行する (過去 15 日間に投稿されたフォーラムの返信を取り出す) 場合を考えます。ただし、特定の属性だけを取り出し、強力な整合性のある読み込みをリクエ

ストする、オプションのクエリパラメータを指定するとします。次の C# サンプルコードでは、QueryOperationConfig オブジェクトを使用してリクエストを作成しています。

Example

```
Table table = Table.LoadTable(client, "Reply");
DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
QueryOperationConfig config = new QueryOperationConfig()
{
  HashKey = "DynamoDB Thread 2", //Partition key
  AttributesToGet = new List<string>
    { "Subject", "ReplyDateTime", "PostedBy" },
  ConsistentRead = true,
  Filter = new RangeFilter(QueryOperator.GreaterThan, twoWeeksAgoDate)
};

Search search = table.Query(config);
```

例: Table.Query メソッドを使用したクエリ

次の C# コード例では、Table.Query メソッドを使用して次のサンプルクエリを実行しています。

- 次のクエリは、Reply テーブルに対して実行されています。
  - 過去 15 日間に投稿されたフォーラムスレッドの返信を検索しています。

    このクエリは 2 回実行されます。最初の Table.Query の呼び出しの例では、必須のクエリパラメータだけが指定されています。2 回目の Table.Query の呼び出しでは、オプションのクエリパラメータを指定して、強力な整合性のある読み込みと、取得する属性のリストをリクエストしています。
  - 特定の期間中に投稿されたフォーラムスレッドの返信を検索する。

    このクエリでは Between クエリ演算子を使用して、2 つの日付間に投稿された返信が検索されます。
- ProductCatalog から製品を取得します。

  ProductCatalog テーブルにはパーティションキーでしかないプライマリキーがあるため、項目の取得だけが可能で、テーブルのクエリを行うことはできません。この例では、項目 Id を使用して特定の製品項目を取得しています。

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class MidLevelQueryAndScan
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                // Query examples.
                Table replyTable = Table.LoadTable(client, "Reply");
                string forumName = "Amazon DynamoDB";
                string threadSubject = "DynamoDB Thread 2";
                FindRepliesInLast15Days(replyTable, forumName, threadSubject);
                FindRepliesInLast15DaysWithConfig(replyTable, forumName,
 threadSubject);
                FindRepliesPostedWithinTimePeriod(replyTable, forumName,
 threadSubject);

                // Get Example.
                Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
                int productId = 101;
                GetProduct(productCatalogTable, productId);

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = tableName.GetItem(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not exist");
            }
        }

        private static void FindRepliesInLast15Days(Table table, string forumName,
 string threadSubject)
        {
            string Attribute = forumName + "#" + threadSubject;

            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            QueryFilter filter = new QueryFilter("Id", QueryOperator.Equal,
 partitionKey);
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
 twoWeeksAgoDate);

            // Use Query overloads that takes the minimum required query parameters.
            Search search = table.Query(filter);

            List<Document> documentSet = new List<Document>();
            do
            {
                documentSet = search.GetNextSet();
                Console.WriteLine("\nFindRepliesInLast15Days: printing ............");
                foreach (var document in documentSet)
                    PrintDocument(document);
            } while (!search.IsDone);
        }

        private static void FindRepliesPostedWithinTimePeriod(Table table, string
 forumName, string threadSubject)
        {
            DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0, 0));
            DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0, 0));
```

```
           QueryFilter filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadSubject);
           filter.AddCondition("ReplyDateTime", QueryOperator.Between, startDate,
endDate);

           QueryOperationConfig config = new QueryOperationConfig()
           {
               Limit = 2, // 2 items/page.
               Select = SelectValues.SpecificAttributes,
               AttributesToGet = new List<string> { "Message",
                                32"ReplyDateTime",
                                "PostedBy" },
               ConsistentRead = true,
               Filter = filter
           };

           Search search = table.Query(config);

           List<Document> documentList = new List<Document>();

           do
           {
               documentList = search.GetNextSet();
               Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} ............", startDate, endDate);
               foreach (var document in documentList)
               {
                   PrintDocument(document);
               }
           } while (!search.IsDone);
       }

       private static void FindRepliesInLast15DaysWithConfig(Table table, string
forumName, string threadName)
       {
           DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
           QueryFilter filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadName);
           filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);
           // You are specifying optional parameters so use QueryOperationConfig.
           QueryOperationConfig config = new QueryOperationConfig()
           {
               Filter = filter,
                // Optional parameters.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Message", "ReplyDateTime",
                                 "PostedBy" },
                ConsistentRead = true
            };

            Search search = table.Query(config);

            List<Document> documentSet = new List<Document>();
            do
            {
                documentSet = search.GetNextSet();
                Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
 printing ............");
                foreach (var document in documentSet)
                    PrintDocument(document);
            } while (!search.IsDone);
        }

        private static void PrintDocument(Document document)
        {
            //   count++;
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                    stringValue = value.AsPrimitive().Value.ToString();
                else if (value is PrimitiveList)
                    stringValue = string.Join(",", (from primitive
                                    in value.AsPrimitiveList().Entries
                                                    select primitive.Value).ToArray());
                Console.WriteLine("{0} - {1}", attribute, stringValue);
            }
        }
    }
}
```

AWS SDK for .NET での Table.Scan メソッド

Scan メソッドではテーブル全体のスキャンが実行されます。ここでは 2 つのオーバーロードを使用できます。Scan メソッドで必要とされるパラメータは、次のオーバーロードを使用して指定できる、スキャンフィルタだけです。

Example

```
Scan(ScanFilter filter);
```

たとえば、フォーラムスレッドのテーブルを維持していて、そこではスレッドの件名 (プライマリ)、関連メッセージ、スレッドが属するフォーラムの Id、Tags などの情報を追跡しているとします。スレッドの件名がプライマリキーであるとします。

Example

```
Thread(Subject, Message, ForumId, Tags, LastPostedDateTime, .... )
```

これは、AWS フォーラムで見られるフォーラムやスレッドを簡略化したものです (「ディスカッションフォーラム」を参照)。次の C# サンプルコードでは、特定のフォーラム (ForumId = 101) 内で、「sortkey」というタグが付加されたすべてのスレッドをクエリしています。ForumId はプライマリキーではないため、この例ではテーブルをスキャンしています。ScanFilter には 2 つの条件が含まれています。クエリによって、両方の条件を満たすすべてのスレッドが返されます。

Example

```
string tableName = "Thread";
Table ThreadTable = Table.LoadTable(client, tableName);

ScanFilter scanFilter = new ScanFilter();
scanFilter.AddCondition("ForumId", ScanOperator.Equal, 101);
scanFilter.AddCondition("Tags", ScanOperator.Contains, "sortkey");

Search search = ThreadTable.Scan(scanFilter);
```

オプションパラメータの指定

Scan には、取得する属性のリストや、強力な整合性のある読み込みを実行するかどうかなど、オプションのパラメータを指定することもできます。オプションパラメータを指定するには、必須およびオプションのパラメータを含む ScanOperationConfig オブジェクトを作成し、次のオーバーロードを使用する必要があります。

Example

```
Scan(ScanOperationConfig config);
```

次の C# コード例では、前述のものと同じ (ForumId が 101 であり、Tag 属性に「sortkey」キーワードが含まれているフォーラムスレッドを検索する) クエリを実行しています。オプションのパラメータを追加して、特定の属性リストのみ取得するものとします。この場合は、次のサンプルコードに示すように、すべてのパラメータ、必須およびオプションのパラメータを指定して ScanOperationConfig オブジェクトを作成する必要があります。

Example

```
string tableName = "Thread";
Table ThreadTable = Table.LoadTable(client, tableName);

ScanFilter scanFilter = new ScanFilter();
scanFilter.AddCondition("ForumId", ScanOperator.Equal, forumId);
scanFilter.AddCondition("Tags", ScanOperator.Contains, "sortkey");

ScanOperationConfig config = new ScanOperationConfig()
{
  AttributesToGet = new List<string> { "Subject", "Message" } ,
  Filter = scanFilter
};

Search search = ThreadTable.Scan(config);
```

例: Table.Scan メソッドを使用したスキャン

Scan オペレーションでは完全なテーブルスキャンが実行されるため、非効率的なオペレーションになる可能性があります。代わりにクエリを使用してください。ただし場合によっては、テーブルに対してスキャンを実行する必要が生じることがあります。たとえば、次の C# サンプルコードに示すように、製品価格にデータ入力エラーがあるためテーブルのスキャンを行う場合があります。この例では ProductCatalog テーブルをスキャンして、価格の値が 0 未満である製品を検索しています。この例は、2 つの Table.Scan オーバーロードの使い方を示しています。

- Table.Scan オブジェクトをパラメータとする ScanFilter。

  必須パラメータだけを渡す場合は、ScanFilter パラメータを渡すことができます。
- Table.Scan オブジェクトをパラメータとする ScanOperationConfig。

ScanOperationConfig メソッドにオプションパラメータを渡す場合には、Scan パラメータを使用する必要があります。

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

namespace com.amazonaws.codesamples
{
    class MidLevelScanOnly
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // Scan example.
            FindProductsWithNegativePrice(productCatalogTable);
            FindProductsWithNegativePriceWithConfig(productCatalogTable);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void FindProductsWithNegativePrice(Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced < 0.
            ScanFilter scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            Search search = productCatalogTable.Scan(scanFilter);

            List<Document> documentList = new List<Document>();
            do
            {
                documentList = search.GetNextSet();
                Console.WriteLine("\nFindProductsWithNegativePrice: 
 printing ............");
```

```
                foreach (var document in documentList)
                    PrintDocument(document);
            } while (!search.IsDone);
        }

        private static void FindProductsWithNegativePriceWithConfig(Table
 productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced < 0.
            ScanFilter scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            ScanOperationConfig config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" }
            };

            Search search = productCatalogTable.Scan(config);

            List<Document> documentList = new List<Document>();
            do
            {
                documentList = search.GetNextSet();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
 printing ...........");
                foreach (var document in documentList)
                    PrintDocument(document);
            } while (!search.IsDone);
        }

        private static void PrintDocument(Document document)
        {
            //   count++;
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                    stringValue = value.AsPrimitive().Value.ToString();
                else if (value is PrimitiveList)
                    stringValue = string.Join(",", (from primitive
                                          in value.AsPrimitiveList().Entries
                                                              select primitive.Value).ToArray());
                    Console.WriteLine("{0} - {1}", attribute, stringValue);
                }
            }
        }
}
```

.NET: オブジェクト永続性モデル

トピック

AWS SDK for .NET には、クライアント側クラスを Amazon DynamoDB テーブルにマッピングできるオブジェクト永続性モデルを有効にしています。各オブジェクトインスタンスが、対応するテーブルの項目にマッピングされます。クライアント側オブジェクトをテーブルに保存するために、オブジェクト永続性モデルでは、DynamoDB のエントリポイントとして `DynamoDBContext` クラスを使用できます。このクラスでは、DynamoDB に接続してテーブルにアクセスし、各種の CRUD オペレーションやクエリを実行することができます。

オブジェクト永続性モデルには、クライアント側クラスをテーブルにマッピングし、プロパティ/フィールドを属性にマッピングする、属性のセットが用意されています。

ⓘ Note

オブジェクト永続性モデルには、テーブルを作成、更新、または削除するための API はありません。データオペレーションだけが可能になっています。テーブルの作成、更新、削除が可能なのは AWS SDK for .NET の低レベル API だけです。詳細については、「」を参照してください.NET での DynamoDB テーブルの使用

次の例は、オブジェクト永続性モデルの動作を示しています。ProductCatalog テーブルから開始します。ここでは Id がプライマリキーになっています。

```
ProductCatalog(Id, ...)
```

Book、Title、ISBN プロパティを持つ Authors クラスがあるとします。次の C# サンプルコードに示すように、オブジェクト永続性モデルで定義された属性を追加することで、Book クラスを ProductCatalog テーブルにマッピングできます。

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]
    public int Id { get; set; }

    public string Title { get; set; }
    public int ISBN { get; set; }

    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }

    [DynamoDBIgnore]
    public string CoverPage { get; set; }
  }
```

前述の例では、DynamoDBTable 属性によって、Book クラスが ProductCatalog テーブルにマッピングされています。

オブジェクト永続性モデルでは、クラスプロパティとテーブル属性との間で、明示的なマッピングとデフォルトのマッピングの両方がサポートされています。

- 明示的なマッピング - プライマリキーにプロパティをマッピングするには、オブジェクト永続性モデル属性の DynamoDBHashKey および DynamoDBRangeKey を使用する必要があります。さらに、非プライマリキー属性については、クラス内のプロパティ名と、マッピング先の対応するテーブル属性が同じでない場合は、DynamoDBProperty 属性を明示的に追加してマッピングを定義する必要があります。

  前述の例では、Id プロパティが同じ名前のプライマリキーにマッピングされ、BookAuthors プロパティが Authors テーブル内の ProductCatalog 属性にマッピングされています。
- デフォルトのマッピング – デフォルトでは、オブジェクト永続性モデルによって、クラスプロパティがテーブル内の同じ名前の属性にマッピングされます。

  前述の例では、Title および ISBN プロパティが、ProductCatalog テーブル内の同じ名前の属性にマッピングされています。

すべてのクラスプロパティをマッピングする必要はありません。これらのプロパティを特定するには、DynamoDBIgnore 属性を追加します。Book インスタンスをテーブルに保存する場合、DynamoDBContext には CoverPage プロパティは含まれません。このプロパティは、書籍インスタンスを取り出す場合にも返されません。

int や string など、.NET プリミティブ型のプロパティをマッピングできます。また、任意データをいずれかの DynamoDB 型にマッピングする適切なコンバーターがある限り、任意のデータ型をマッピングすることもできます。任意の型のマッピングについては、「AWS SDK for .NET のオブジェクト永続性モデルを使用した、DynamoDB での任意データのマッピング」を参照してください。

オブジェクト永続性モデルでは、オプティミスティックロックがサポートされています。それによって、更新オペレーションで、更新する項目の最新のコピーを確実に使用することができます。詳細については、「」を参照してくださいDynamoDB で AWS SDK for .NET のオブジェクト永続性モデルを使用した、バージョン番号による楽観的ロック

DynamoDB 属性

このセクションでは、クラスとプロパティを DynamoDB のテーブルや属性にマッピングできるように、オブジェクト永続性モデルで提供されている属性を説明します。

> Note
>
> 次の属性では、DynamoDBTable と DynamoDBHashKey だけが必須です。

DynamoDBGlobalSecondaryIndexHashKey

グローバルセカンダリインデックスのパーティションキーに、クラスプロパティをマッピングします。この属性は、グローバルセカンダリインデックスを Query する必要がある場合に使用します。

DynamoDBGlobalSecondaryIndexRangeKey

グローバルセカンダリインデックスのソートキーにクラスプロパティをマッピングします。この属性は、グローバルセカンダリインデックスを Query し、インデックスソートキーを使用して結果を絞り込む必要がある場合に使用します。

DynamoDBHashKey

テーブルのプライマリキーのパーティションキーにクラスプロパティをマッピングします。プライマリキーの属性をコレクション型にすることはできません。

次の C# サンプルコードでは、Book クラスを ProductCatalog テーブルに、Id プロパティをテーブルのプライマリキーのパーティションキーにマッピングしています。

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

DynamoDBIgnore

関連するプロパティを無視するように指示します。クラスプロパティを保存しない場合は、この属性を追加することで、テーブルにオブジェクトを保存するときにこのプロパティを含めないように DynamoDBContext に指示できます。

DynamoDBLocalSecondaryIndexRangeKey

ローカルセカンダリインデックスのソートキーにクラスプロパティをマッピングします。この属性は、ローカルセカンダリインデックスを Query し、インデックスソートキーを使用して結果を絞り込む必要がある場合に使用します。

DynamoDBProperty

テーブルの属性にクラスプロパティをマッピングします。クラスプロパティを同じ名前のテーブル属性にマッピングする場合は、この属性を指定する必要はありません。ただし名前が異なる場合は、このタグを使用してマッピングを指定できます。次の C# ステートメントでは、DynamoDBProperty によって、BookAuthors プロパティがテーブル内の Authors 属性にマッピングされています。

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

DynamoDBContext はこのマッピング情報を使用して、対応するテーブルにオブジェクトデータを保存するときに Authors 属性を作成します。

DynamoDBRenamable

クラスプロパティの代替名を指定します。これは、テーブル属性がクラスプロパティの名前と異なる DynamoDB テーブルに、任意データをマッピングするためのカスタムコンバーターを記述する場合に役立ちます。

DynamoDBRangeKey

テーブルのプライマリキーのソートキーにクラスプロパティをマッピングします。テーブルに複合プライマリキー (パーティションキーおよびソートキー) がある場合は、クラスマッピングで、DynamoDBHashKey と DynamoDBRangeKey の両方の属性を指定する必要があります。

たとえば、サンプルテーブルの Reply には、Id パーティションキーと Replenishment ソートキーで構成されたプライマリキーがあります。次の C# サンプルコードでは、Reply クラスを Reply テーブルにマッピングします。クラス定義では、プロパティのうち 2 つがプライマリキーにマッピングされることも示しています。

サンプルテーブルの詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

```
[DynamoDBTable("Reply")]
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }
```

```
    // Additional properties go here.
}
```

DynamoDBTable

クラスがマッピングされる、DynamoDB 内のターゲットテーブルを識別します。例えば次の C# サンプルコードでは、Developer クラスを DynamoDB の People テーブルにマッピングしています。

```
[DynamoDBTable("People")]
public class Developer { ...}
```

この属性は、継承またはオーバーライドすることができます。

- DynamoDBTable 属性は継承できます。前述の例では、Lead クラスから継承された新しいクラス Developer を追加すると、People テーブルにもマッピングされます。Developer テーブルに Lead と People の両方のオブジェクトが格納されます。
- DynamoDBTable 属性もオーバーライドできます。次の C# サンプルコードでは、Manager クラスは、Developer クラスから継承されます。ただし、DynamoDBTable 属性を明示的に追加すると、クラスは別のテーブル (Managers) にマップされます。

  ```
  [DynamoDBTable("Managers")]
  public class Manager : Developer { ...}
  ```

次の C# の例に示すように、オプションのパラメータ LowerCamelCaseProperties を追加すると、オブジェクトをテーブルに格納する際にプロパティ名の先頭文字を小文字にするよう、DynamoDB にリクエストを送ることができます。

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
    ...
}
```

Developer クラスのインスタンスを保存する場合、DynamoDBContext では DeveloperName プロパティが developerName として保存されます。

DynamoDBVersion

項目のバージョン番号を格納するクラスプロパティを識別します。バージョニングの詳細については、「DynamoDB で AWS SDK for .NET のオブジェクト永続性モデルを使用した、バージョン番号による楽観的ロック」を参照してください。

DynamoDBContext クラス

`DynamoDBContext` クラスは Amazon DynamoDB データベースのエントリポイントです。このクラスから DynamoDB に接続して、各種のテーブル内のデータにアクセスし、さまざまな CRUD オペレーションとクエリを実行することができます。`DynamoDBContext` クラスでは次のメソッドを使用できます。

CreateMultiTableBatchGet

複数の個々の `MultiTableBatchGet` オブジェクトで構成される `BatchGet` オブジェクトを作成します。これらの各 `BatchGet` オブジェクトは、単一の DynamoDB テーブルから項目を取り出す場合に使用します。

1 つ以上のテーブルから項目を取得するには、`ExecuteBatchGet` メソッドを使用し、`MultiTableBatchGet` オブジェクトをパラメータとして渡します。

CreateMultiTableBatchWrite

複数の個々の `MultiTableBatchWrite` オブジェクトで構成される `BatchWrite` オブジェクトを作成します。これらの各 `BatchWrite` オブジェクトは、単一の DynamoDB テーブルに項目を書き込んだり、それを削除したりするために使用します。

テーブルに書き込むには、`ExecuteBatchWrite` メソッドを使用し、`MultiTableBatchWrite` オブジェクトをパラメータとして渡します。

CreateBatchGet

テーブルから複数の項目を取り出すために使用できる `BatchGet` オブジェクトを作成します。詳細については、「バッチ取得: 複数の項目の取得 」を参照してください。

createBatchWrite

テーブルに複数の項目を入力する、またはテーブルから複数の項目を削除するために使用できる `BatchWrite` オブジェクトを作成します。詳細については、「バッチ書き込み: 複数の項目の置換および削除 」を参照してください。

Delete

テーブルから項目を削除します。このメソッドでは、削除する項目のプライマリキーが必要になります。プライマリキーの値、またはこのメソッドのパラメータとしてプライマリキーの値を使用するクライアント側オブジェクトを入力できます。

- クライアント側オブジェクトをパラメータとして指定し、オプティミスティックロックを有効にすると、クライアント側とサーバー側のオブジェクトのバージョンが一致する場合のみ、削除が成功します。
- プライマリキーの値だけをパラメータとして指定すると、オプティミスティックロックを有効にしているかどうかにかかわらず、削除が成功します。

> **Note**
>
> このオペレーションをバックグラウンドで実行するには、`DeleteAsync` メソッドを使用します。

Dispose

すべてのマネージドリソースとアンマネージドリソースを破棄します。

ExecuteBatchGet

1 つ以上のテーブルからデータを読み込みます。`BatchGet` 内のすべての `MultiTableBatchGet` オブジェクトを処理します。

> **Note**
>
> このオペレーションをバックグラウンドで実行するには、`ExecuteBatchGetAsync` メソッドを使用します。

ExecuteBatchWrite

1 つ以上のテーブルにデータを書き込むまたは削除します。`BatchWrite` 内のすべての `MultiTableBatchWrite` オブジェクトを処理します。

> Note
>
> このオペレーションをバックグラウンドで実行するには、ExecuteBatchWriteAsync メソッドを使用します。

FromDocument

Document のインスタンスと仮定すると、FromDocument メソッドは、クライアント側のクラスのインスタンスを返します。

これは、オブジェクト永続性モデルと合わせてドキュメントモデルクラスを使用してデータオペレーションを行う場合に役立ちます。AWS SDK for .NET で使用されるドキュメントモデルクラスの詳細については、「.NET ドキュメントモデル」を参照してください。

Document という名前の doc オブジェクトがあり、Forum 項目の表現を含んでいるとします。(このオブジェクトの構成方法については、下の ToDocument メソッドの説明を参照してください)。次の C# サンプルコードに示すように、FromDocument を使用して Forum から Document 項目を取得することができます。

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> Note
>
> Document オブジェクトで IEnumerable インターフェイスを実装している場合、FromDocuments メソッドを使用できます。これにより、Document のすべてのクラスインスタンスを反復的に処理できます。

FromQuery

QueryOperationConfig オブジェクトに定義されたクエリパラメータを使用して、Query オペレーションを実行します。

> Note
>
> このオペレーションをバックグラウンドで実行するには、`FromQueryAsync` メソッドを使用します。

FromScan

`ScanOperationConfig` オブジェクトに定義されたスキャンパラメータを使用して、`Scan` オペレーションを実行します。

> Note
>
> このオペレーションをバックグラウンドで実行するには、`FromScanAsync` メソッドを使用します。

GetTargetTable

指定した型のターゲットテーブルを取り出します。これは、任意データを DynamoDB テーブルにマッピングするためのカスタムコンバーターを記述していて、カスタムデータ型に関連付けられているテーブルを特定する必要がある場合に役立ちます。

Load

テーブルから項目を取り出します。このメソッドでは、取り出す項目のプライマリキーだけが必要になります。

DynamoDB のデフォルトでは、結果整合性のある値が含まれる項目が返されます。結果整合性モデルの詳細については、「読み込み整合性」を参照してください。

> Note
>
> このオペレーションをバックグラウンドで実行するには、`LoadAsync` メソッドを使用します。

Query

指定したクエリパラメータに基づいてテーブルのクエリが実行されます。

複合プライマリキー (パーティションキーおよびソートキー) が存在する場合にのみ、テーブルにクエリを実行できます。クエリを実行する場合は、パーティションキーと、ソートキーに適用される条件を指定する必要があります。

ここで、クライアント側の Reply クラスが、DynamoDB の Reply テーブルにマッピングされている場合を考えてみます。次の C# サンプルコードでは、Reply テーブルのクエリを実行し、過去 15 日間に投稿されたフォーラムスレッドの返信を検索しています。Reply テーブルには、Id パーティションキーと ReplyDateTime ソートキーを持つプライマリキーがあります。Reply テーブルの詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

Example

```
DynamoDBContext context = new DynamoDBContext(client);

string replyId = "DynamoDB#DynamoDB Thread 1"; //Partition key
DateTime twoWeeksAgoDate = DateTime.UtcNow.Subtract(new TimeSpan(14, 0, 0, 0)); // Date
 to compare.
IEnumerable<Reply> latestReplies = context.Query<Reply>(replyId,
 QueryOperator.GreaterThan, twoWeeksAgoDate);
```

これにより、Reply オブジェクトのコレクションが返されます。

Query メソッドでは、「遅延ロード」された IEnumerable コレクションが返ります。最初に結果が 1 ページのみ返され、必要に応じて、さらに次ページを要求するサービス呼び出しが行われます。一致する項目をすべて取得するには、IEnumerable を反復的に処理します。

テーブルにシンプルなプライマリキー (パーティションキー) がある場合は、Query メソッドを使用できません。代わりに Load メソッドを使用して、パーティションキーを入力して項目を取り出すことができます。

Note

このオペレーションをバックグラウンドで実行するには、QueryAsync メソッドを使用します。

保存

指定したオブジェクトがテーブルに保存されます。入力オブジェクトで指定されたプライマリキーがテーブル内に存在しない場合は、このメソッドによって新しい項目がテーブルに追加されます。プライマリキーが存在する場合は、このメソッドによって既存の項目が更新されます。

オプティミスティックロックを設定している場合には、クライアント側とサーバー側で項目のバージョンが一致する場合のみ、更新が正常に実行されます。詳細については、「DynamoDB で AWS SDK for .NET のオブジェクト永続性モデルを使用した、バージョン番号による楽観的ロック」を参照してください。

> **Note**
>
> このオペレーションをバックグラウンドで実行するには、SaveAsync メソッドを使用します。

Scan

テーブル全体のスキャンを実行します。

スキャン結果をフィルタリングするには、スキャン条件を指定します。この条件は、テーブル内の任意の属性に適用することができます。例えば、クライアント側の Book クラスが、DynamoDB の ProductCatalog テーブルにマッピングされているとします。次の C# コード例では、テーブルがスキャンされ、価格が 0 未満の書籍項目だけが返されています。

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

Scan メソッドでは、「遅延ロード」された IEnumerable コレクションが返ります。最初に結果が 1 ページのみ返され、必要に応じて、さらに次ページを要求するサービス呼び出しが行われます。一致するすべての項目は、IEnumerable を反復的に処理するだけで取得できます。

パフォーマンス上の理由から、テーブルについてはスキャンを避け、クエリを行うようにしてください。

> Note
>
> このオペレーションをバックグラウンドで実行するには、ScanAsync メソッドを使用します。

ToDocument

クラスインスタンスから、Document ドキュメントモデルクラスのインスタンスが返されます。

これは、オブジェクト永続性モデルと合わせてドキュメントモデルクラスを使用してデータオペレーションを行う場合に役立ちます。AWS SDK for .NET で使用されるドキュメントモデルクラスの詳細については、「.NET ドキュメントモデル」を参照してください。

クライアント側のクラスが Forum サンプルテーブルにマッピングされているとします。その場合は次の C# サンプルコードに示すように、DynamoDBContext を使用して、Document テーブルから項目を Forum オブジェクトとして取得することができます。

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

DynamoDBContext でのオプションパラメータの指定

オブジェクト永続性モデルを使用する場合は、DynamoDBContext に次のオプションパラメータを指定できます。

- **ConsistentRead**-Load、Query、または Scan オペレーションを使用してデータを取得する場合、このオプションパラメータを追加して、データの最新の値をリクエストすることができます。
- **IgnoreNullValues**-このパラメータにより、Save オペレーション時に属性の null 値を無視するように DynamoDBContext に指示できます。このパラメータが false の場合 (または設定されていない場合)、null 値は、特定の属性を削除するディレクティブと見なされます。
- **SkipVersionCheck**–このパラメータは、項目の保存または削除を実行する際、バージョンの比較を行わないように DynamoDBContext に指示します。バージョニングの詳細については、「DynamoDB で AWS SDK for .NET のオブジェクト永続性モデルを使用した、バージョン番号による楽観的ロック」を参照してください。

- **TableNamePrefix**-すべてのテーブル名に特定の文字列をプレフィックスします。このパラメータが null の場合（または設定されていない場合）、プレフィックスは使用されません。

次の C# サンプルコードでは、前述のオプションパラメータのうち 2 つを指定して、新しい DynamoDBContext を作成します。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
       new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

DynamoDBContext では、このコンテキストを使用して送信した各リクエストに、これらのオプションパラメータが含められます。

次の C# サンプルコードに示すように、これらのパラメータを DynamoDBContext レベルで設定する代わりに、DynamoDBContext を使用して実行するオペレーションに対して個別に指定することもできます。この例では特定の書籍項目がロードされています。Load の DynamoDBContext メソッドでは、前述のオプションパラメータを指定します。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

この場合 DynamoDBContext には、Get リクエストを送信する場合のみ、これらのパラメータが含まれます。

サポートされているデータ型

オブジェクト永続性モデルでは、プリミティブな .NET データ型、コレクション、および任意のデータ型のセットがサポートされています。このモデルでは、次のプリミティブデータ型がサポートされています。

- bool

- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

オブジェクト永続性モデルでは .NET コレクション型もサポートされています。`DynamoDBContext` は具体的なコレクション型、およびシンプルな Plain Old CLR Objects (POCOs) を変換できます。

次の表に、前述の .NET 型が DynamoDB の型にどのようにマッピングされるかを示します。

.NET プリミティブ型	DynamoDB 型
すべての数値型	N（数値型）
すべての文字列型	S (文字列型)
MemoryStream、byte[]	B（バイナリ型）
ブール	N (数値型)。0 は false、1 は true を表します。
コレクション型	BS（バイナリセット）型、SS（文字列セット）型、NS（数値セット）型
DateTime	S (文字列型)。`DateTime` の値は、ISO-8601 形式の文字列として格納されます。

オブジェクト永続性モデルでは、任意のデータ型もサポートされています。ただし、複合型を DynamoDB 型にマッピングする場合には、変換用のコードを用意する必要があります。

> Note
>
> - 空のバイナリ値がサポートされています。
> - 空の文字列値の読み取りがサポートされています。空の文字列属性値は、DynamoDB への書き込み中に、文字列セット型の属性値内でサポートされます。文字列型の空の文字列属性値と、リスト型またはマップ型に含まれる空の文字列値が書き込みリクエストから削除されます。

DynamoDB で AWS SDK for .NET のオブジェクト永続性モデルを使用した、バージョン番号による楽観的ロック

オブジェクト永続性モデルではオプティミスティックロックがサポートされており、項目を更新または削除する前に、アプリケーションの項目バージョンとサーバー側の項目バージョンが同じになります。更新する項目を取得するとします。しかし、更新を返送する前に、他のアプリケーションが同じ項目を更新しました。この場合、アプリケーションに項目の古いコピーが残ることになります。オプティミスティックロックがない場合に更新を行うと、他のアプリケーションで行われた更新が上書きされます。

オブジェクト永続性モデルのオプティミスティックロック機能では、オプティミスティックロックを有効にするために `DynamoDBVersion` タグを使用できます。この機能を使用するには、バージョン番号を格納するためのプロパティをクラスに追加します。`DynamoDBVersion` 属性をプロパティに追加します。最初にオブジェクトを保存すると、`DynamoDBContext` によってバージョン番号が割り当てられ、項目を更新するたびにその値が増えていきます。

更新または削除リクエストは、クライアント側のオブジェクトのバージョンが、サーバー側の対応する項目のバージョン番号に一致する場合のみ成功します。アプリケーションに古いコピーがある場合に項目を更新または削除するには、その前にサーバーから最新バージョンを取得する必要があります。

次の C# サンプルコードでは、オブジェクト永続性属性と合わせて `Book` クラスを定義し、`ProductCatalog` テーブルにマッピングします。`VersionNumber` 属性が指定されたクラスの `DynamoDBVersion` プロパティには、バージョン番号が格納されます。

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]   //Partition key
    public int Id { get; set; }
    [DynamoDBProperty]
    public string Title { get; set; }
    [DynamoDBProperty]
    public string ISBN { get; set; }
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }
    [DynamoDBVersion]
    public int? VersionNumber { get; set; }
  }
```

> Note
>
> DynamoDBVersion 属性は、null が許容された数値プリミティブ型 (int? など) に対してのみ適用できます。

オプティミスティックロックは、DynamoDBContext オペレーションに対して次のような影響があります。

- DynamoDBContext は、新しい項目に対して初期バージョン番号 0 を割り当てます。既存の項目を取得し、1 つ以上のプロパティを更新して変更を保存しようとすると、保存オペレーションは、クライアント側とサーバー側のバージョン番号が一致する場合にのみ成功します。DynamoDBContext によってバージョン番号が増加します。バージョン番号を設定する必要はありません。
- 次の C# サンプルコードに示すように、Delete メソッドでは、プライマリキーの値またはオブジェクトのいずれかをパラメータとして指定できるオーバーロードを使用できます。

  Example

  ```
  DynamoDBContext context = new DynamoDBContext(client);
  ...
  // Load a book.
  Book book = context.Load<ProductCatalog>(111);
  ```

```
// Do other operations.
// Delete 1 - Pass in the book object.
context.Delete<ProductCatalog>(book);

// Delete 2 - Pass in the Id (primary key)
context.Delete<ProductCatalog>(222);
```

パラメータとしてオブジェクトを指定した場合、オブジェクトのバージョンがサーバー側の対応する項目のバージョンと一致する場合のみ、削除が成功します。ただし、パラメータとしてプライマリキーの値を入力した場合には、DynamoDBContext はバージョン番号を認識せず、バージョンチェックを行わずに項目を削除します。

オブジェクト永続性モデルのコードで、オプティミスティックロックを内部実装する場合は、DynamoDB の条件付き更新と条件付き削除用の API アクションが使用されることに注意してください。

楽観的ロックの無効化

オプティミスティックロックを無効にするには、SkipVersionCheck 設定プロパティを使用します。このプロパティは、DynamoDBContext の作成時に設定できます。この場合、このコンテキストを使用して作成したすべてのリクエストについて、オプティミスティックロックが無効になります。詳細については、「DynamoDBContext でのオプションパラメータの指定 」を参照してください。

コンテキストレベルでプロパティを設定する代わりに、次の C# サンプルコードに示すように、特定のオペレーションに対するオプティミスティックロックを無効にすることができます。この例では、このコンテキストを使用して書籍項目を削除しています。Delete メソッドはオプションの SkipVersionCheck プロパティを true に設定し、バージョンチェックを無効にします。

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Load a book.
Book book = context.Load<ProductCatalog>(111);
...
// Delete the book.
context.Delete<Book>(book, new DynamoDBContextConfig { SkipVersionCheck = true });
```

AWS SDK for .NET のオブジェクト永続性モデルを使用した、DynamoDB での任意データのマッピング

サポート済みの .NET 型 (「サポートされているデータ型」を参照) に加えて、アプリケーション内にあり Amazon DynamoDB 型に直接マッピングされていない型を使用できます。オブジェクト永続性モデルでは、任意型から DynamoDB 型に (またはその反対に) データを変換できるコンバーターを用意することで、任意型のデータを格納できます。コンバーターコードによって、オブジェクトの保存およびロード中にデータが変換されます。

クライアント側で任意の型を作成できます。ただし、テーブルには、いずれかの DynamoDB 型を使用してデータが格納されます。また、クエリおよびスキャンの際のデータ比較は、DynamoDB に格納されているデータに対して行われます。

次の C# コード例では、Book、Id、Title、および ISBN プロパティを使用して Dimension クラスを定義しています。Dimension プロパティは、DimensionType、Height、および Width プロパティを記述する Thickness に含まれています。このコード例では、コンバーターメソッド ToEntry および FromEntry によって、DimensionType と DynamoDB の文字列型との間でデータが変換されます。たとえば、Book インスタンスを保存すると、コンバータによって、「8.5 x 11 x 05」などの書籍 Dimension 文字列が作成されます。書籍を取得すると、その文字列は DimensionType インスタンスに変換されます。

この例では、Book 型を ProductCatalog テーブルにマッピングしています。サンプルの Book インスタンスを保存し、取得し、寸法を更新し、更新された Book を再度保存しています。

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelMappingArbitraryData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);

                // 1. Create a book.
                DimensionType myBookDimensions = new DimensionType()
                {
                    Length = 8M,
                    Height = 11M,
                    Thickness = 0.5M
                };

                Book myBook = new Book
                {
                    Id = 501,
                    Title = "AWS SDK for .NET Object Persistence Model Handling 
 Arbitrary Data",
                    ISBN = "999-9999999999",
                    BookAuthors = new List<string> { "Author 1", "Author 2" },
                    Dimensions = myBookDimensions
                };

                context.Save(myBook);

                // 2. Retrieve the book.
                Book bookRetrieved = context.Load<Book>(501);

                // 3. Update property (book dimensions).
                bookRetrieved.Dimensions.Height += 1;
                bookRetrieved.Dimensions.Length += 1;
                bookRetrieved.Dimensions.Thickness += 0.2M;
                // Update the book.
                context.Save(bookRetrieved);

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }
```

```
    }
    [DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey] //Partition key
        public int Id
        {
            get; set;
        }
        [DynamoDBProperty]
        public string Title
        {
            get; set;
        }
        [DynamoDBProperty]
        public string ISBN
        {
            get; set;
        }
        // Multi-valued (set type) attribute.
        [DynamoDBProperty("Authors")]
        public List<string> BookAuthors
        {
            get; set;
        }
        // Arbitrary type, with a converter to map it to DynamoDB type.
        [DynamoDBProperty(typeof(DimensionTypeConverter))]
        public DimensionType Dimensions
        {
            get; set;
        }
    }

    public class DimensionType
    {
        public decimal Length
        {
            get; set;
        }
        public decimal Height
        {
            get; set;
        }
        public decimal Thickness
        {
            get; set;
        }
    }

    // Converts the complex type DimensionType to string and vice-versa.
    public class DimensionTypeConverter : IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            DimensionType bookDimensions = value as DimensionType;
            if (bookDimensions == null) throw new ArgumentOutOfRangeException();

            string data = string.Format("{1}{0}{2}{0}{3}", " x ",
                            bookDimensions.Length, bookDimensions.Height,
bookDimensions.Thickness);

            DynamoDBEntry entry = new Primitive
            {
                Value = data
            };
            return entry;
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            Primitive primitive = entry as Primitive;
            if (primitive == null || !(primitive.Value is String) ||
string.IsNullOrEmpty((string)primitive.Value))
                throw new ArgumentOutOfRangeException();

            string[] data = ((string)(primitive.Value)).Split(new string[] { " x " },
StringSplitOptions.None);
            if (data.Length != 3) throw new ArgumentOutOfRangeException();

            DimensionType complexData = new DimensionType
            {
                Length = Convert.ToDecimal(data[0]),
                Height = Convert.ToDecimal(data[1]),
                Thickness = Convert.ToDecimal(data[2])
            };
            return complexData;
        }
    }
```

```
}
```

AWS SDK for .NET オブジェクト永続性モデルを使用したバッチオペレーション

バッチ書き込み: 複数の項目の置換および削除

単一のリクエストでテーブルに対して複数のオブジェクトを入力または削除するには、次の手順を実行します。

- DynamoDBContext の createBatchWrite メソッドを実行して、BatchWrite クラスのインスタンスを作成します。
- 入力または削除する項目を指定します。
  - 1 つ以上の項目を入力するには、AddPutItem または AddPutItems メソッドを使用します。
  - 1 つ以上の項目を削除するには、項目のプライマリキー、または削除する項目にマッピングするクライアント側オブジェクトを指定します。削除する項目のリストを指定するには、AddDeleteItem、AddDeleteItems、および AddDeleteKey メソッドを使用します。
- 指定したすべての項目をテーブルに入力またはテーブルから削除するには、BatchWrite.Execute メソッドを呼び出します。

> **Note**
>
> オブジェクト永続性モデルを使用する場合には、バッチ内で任意の数のオペレーションを指定できます。ただし Amazon DynamoDB では、1 つのバッチ内のオペレーションの数と、1 つのバッチオペレーションでのバッチの合計サイズが制限されています。具体的な制限事項については、「BatchWriteItem」を参照してください。許容されている書き込みリクエスト数、または許容されている HTTP ペイロードの最大サイズをバッチ書き込みリクエストが超えたことを API が検出すると、バッチが複数の小さなバッチに分割されます。さらに、バッチ書き込みに対する応答で、未処理の項目が返された場合には、API がそれら未処理の項目を使用して、別のバッチリクエストを自動的に送信します。

DynamoDB の ProductCatalog テーブルへのマッピングを行う、Book というクラスを C# クラスとして定義した場合を考えます。次の C# サンプルコードでは、BatchWrite オブジェクトを使用して、ProductCatalog テーブルに 2 つの項目をアップロードし、1 つの項目を削除しています。

Example

```
DynamoDBContext context = new DynamoDBContext(client);
var bookBatch = context.CreateBatchWrite<Book>();

// 1. Specify two books to add.
Book book1 = new Book
{
  Id = 902,
  ISBN = "902-11-11-1111",
  ProductCategory = "Book",
  Title = "My book3 in batch write"
};
Book book2 = new Book
{
  Id = 903,
  ISBN = "903-11-11-1111",
  ProductCategory = "Book",
  Title = "My book4 in batch write"
};


bookBatch.AddPutItems(new List<Book> { book1, book2 });

// 2. Specify one book to delete.
bookBatch.AddDeleteKey(111);

bookBatch.Execute();
```

複数のテーブルを対象にオブジェクトを入力または削除するには、次の手順を実行します。

- それぞれの型について BatchWrite クラスの 1 つのインスタンスを作成し、前述のセクションで説明したように、入力または削除する項目を指定します。
- 次のいずれかの方法を使用して、MultiTableBatchWrite のインスタンスを作成します。
  - 前述のステップで作成したいずれかの BatchWrite オブジェクトで、Combine メソッドを実行します。
  - MultiTableBatchWrite オブジェクトのリストを入力して、BatchWrite 型のインスタンスを作成します。
  - DynamoDBContext の CreateMultiTableBatchWrite メソッドを実行して、BatchWrite オブジェクトのリストを渡します。

- 指定された入力および削除オペレーションを各種のテーブルで実行する、Execute の MultiTableBatchWrite メソッドを呼び出します。

DynamoDB の Forum と Thread テーブルにマッピングする C# クラス Forum と Thread を定義したとします。さらに、Thread クラスでバージョニングが有効になっているとします。バッチオペレーションではバージョニングがサポートされていないため、次の C# サンプルコードに示すように、バージョニングを明示的に無効にする必要があります。この例では、MultiTableBatchWrite オブジェクトを使用して複数のテーブルを更新しています。

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Create BatchWrite objects for each of the Forum and Thread classes.
var forumBatch = context.CreateBatchWrite<Forum>();

DynamoDBOperationConfig config = new DynamoDBOperationConfig();
config.SkipVersionCheck = true;
var threadBatch = context.CreateBatchWrite<Thread>(config);

// 1. New Forum item.
Forum newForum = new Forum
{
  Name = "Test BatchWrite Forum",
  Threads = 0
};
forumBatch.AddPutItem(newForum);

// 2. Specify a forum to delete by specifying its primary key.
forumBatch.AddDeleteKey("Some forum");

// 3. New Thread item.
Thread newThread = new Thread
{
  ForumName = "Amazon S3 forum",
  Subject = "My sample question",
  KeywordTags = new List<string> { "Amazon S3", "Bucket" },
   Message = "Message text"
};

threadBatch.AddPutItem(newThread);

// Now run multi-table batch write.
```

```
var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);
superBatch.Execute();
```

実例については、「例: AWS SDK for .NET オブジェクト永続性モデルを使用したバッチ書き込みオペレーション」を参照してください。

> Note
>
> DynamoDB バッチ API では、バッチ内の書き込み数と、バッチのサイズが制限されています。詳細については、「BatchWriteItem」を参照してください。.NET オブジェクト永続性モデル API を使用する場合は、任意の数のオペレーションを指定できます。ただし、バッチ内のオペレーションの数またはそのサイズが制限を超えた場合には、.NET API はバッチ書き込みリクエストを複数の小さなバッチに分割して、それらのバッチ書き込みリクエストを DynamoDB に送信します。

バッチ取得: 複数の項目の取得

単一のリクエストでテーブルから複数の項目を取り出すには、次の手順を実行します。

- CreateBatchGet クラスのインスタンスを作成します。
- 取り出すプライマリキーのリストを指定します。
- Execute メソッドを呼び出します。応答では、項目が Results プロパティによって返されます。

次の C# のサンプルコードでは、ProductCatalog から 3 つの項目を取得します。結果内の項目の順序は、必ずしもプライマリキーを指定した順序と同じではありません。

Example

```
DynamoDBContext context = new DynamoDBContext(client);
var bookBatch = context.CreateBatchGet<ProductCatalog>();
bookBatch.AddKey(101);
bookBatch.AddKey(102);
bookBatch.AddKey(103);
bookBatch.Execute();
// Process result.
Console.WriteLine(bookBatch.Results.Count);
Book book1 = bookBatch.Results[0];
Book book2 = bookBatch.Results[1];
Book book3 = bookBatch.Results[2];
```

複数のテーブルからオブジェクトを取り出すには、次の手順を実行します。

- それぞれの型について `CreateBatchGet` 型のインスタンスを作成し、各テーブルから取り出すプライマリキーの値を指定します。
- 次のいずれかの方法を使用して、`MultiTableBatchGet` クラスのインスタンスを作成します。
  - 前述のステップで作成したいずれかの `Combine` オブジェクトで、`BatchGet` メソッドを実行します。
  - `MultiBatchGet` オブジェクトのリストを入力して、`BatchGet` 型のインスタンスを作成します。
  - `DynamoDBContext` の `CreateMultiTableBatchGet` メソッドを実行して、`BatchGet` オブジェクトのリストを渡します。
- `Execute` の `MultiTableBatchGet` メソッドを呼び出すと、個々の `BatchGet` オブジェクトに結果が返ります。

次の C# のサンプルコードでは、`Order` メソッドを使用して、`OrderDetail` および `CreateBatchGet` テーブルから複数の項目を取得します。

Example

```
var orderBatch = context.CreateBatchGet<Order>();
orderBatch.AddKey(101);
orderBatch.AddKey(102);

var orderDetailBatch = context.CreateBatchGet<OrderDetail>();
orderDetailBatch.AddKey(101, "P1");
orderDetailBatch.AddKey(101, "P2");
orderDetailBatch.AddKey(102, "P3");
orderDetailBatch.AddKey(102, "P1");

var orderAndDetailSuperBatch = orderBatch.Combine(orderDetailBatch);
orderAndDetailSuperBatch.Execute();

Console.WriteLine(orderBatch.Results.Count);
Console.WriteLine(orderDetailBatch.Results.Count);

Order order1 = orderBatch.Results[0];
Order order2 = orderBatch.Results[1];
```

```
OrderDetail orderDetail1 = orderDetailBatch.Results[0];
```

例: AWS SDK for .NET オブジェクト永続性モデルを使用した CRUD オペレーション

次の C# サンプルコードでは、Book、Id、Title、および ISBN プロパティを使用して Authors クラスを宣言しています。この例では、オブジェクト永続性属性を使用して、これらのプロパティを Amazon DynamoDB の ProductCatalog テーブルにマップしています。次に、この例では DynamoDBContext を使用して、一般的な作成、読み取り、更新、および削除 (CRUD) オペレーションを示します。この例では、サンプルの Book インスタンスを作成して ProductCatalog テーブルに保存しています。さらに書籍項目を取り出し、その ISBN および Authors プロパティを更新しています。更新によって、既存の作成者リストが置き換えられていることに注意してください。最終的に、この例では書籍項目が削除されています。

この例で使用されている ProductCatalog テーブルの詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。次の例をテストするための詳しい手順については、「.NET コード例」を参照してください。

> Note
>
> 以下の例は、同期メソッドをサポートしていないため、.NET Core コアには適用されません。詳細については、「.NET 用 AWS 非同期 API」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class HighLevelItemCRUD
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);
                TestCRUDOperations(context);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void TestCRUDOperations(DynamoDBContext context)
        {
            int bookID = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookID,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                ISBN = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book.
            context.Save(myBook);
            // Retrieve the book.
            Book bookRetrieved = context.Load<Book>(bookID);

            // Update few properties.
            bookRetrieved.ISBN = "222-2222221001";
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
 x" }; // Replace existing authors list with this.
            context.Save(bookRetrieved);

            // Retrieve the updated book. This time add the optional ConsistentRead
 parameter using DynamoDBContextConfig object.
            Book updatedBook = context.Load<Book>(bookID, new DynamoDBContextConfig
            {
                ConsistentRead = true
            });

            // Delete the book.
            context.Delete<Book>(bookID);
            // Try to retrieve deleted book. It should return null.
            Book deletedBook = context.Load<Book>(bookID, new DynamoDBContextConfig
            {
                ConsistentRead = true
            });
            if (deletedBook == null)
                Console.WriteLine("Book is deleted");
        }
    }

    [DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey] //Partition key
        public int Id
        {
            get; set;
        }
        [DynamoDBProperty]
        public string Title
        {
            get; set;
        }
        [DynamoDBProperty]
        public string ISBN
        {
            get; set;
        }
        [DynamoDBProperty("Authors")] //String Set datatype
        public List<string> BookAuthors
        {
            get; set;
        }
    }
}
```

例: AWS SDK for .NET オブジェクト永続性モデルを使用したバッチ書き込みオペレーション

次の C# サンプルコードでは、宣言した `Book`、`Forum`、`Thread`、および `Reply` クラスを Amazon DynamoDB テーブルに対し、オブジェクト永続性モデル属性を使用しながらマッピングしています。

この例では、さらに `DynamoDBContext` を使用して、次のバッチ書き込みオペレーションを示しています。

- BatchWrite オブジェクト。ProductCatalog テーブルに対して書籍項目を入力または削除します。
- MultiTableBatchWrite オブジェクト。Forum および Thread テーブルに対して書籍項目を入力または削除します。

この例で使用されているテーブルの詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。次の例をテストするための詳しい手順については、「.NET コード例」を参照してください。

Note

以下の例は、同期メソッドをサポートしていないため、.NET Core コアには適用されません。詳細については、「.NET 用 AWS 非同期 API」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelBatchWriteItem
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);
                SingleTableBatchWrite(context);
                MultiTableBatchWrite(context);
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void SingleTableBatchWrite(DynamoDBContext context)
        {
            Book book1 = new Book
            {
                Id = 902,
                InPublication = true,
                ISBN = "902-11-11-1111",
                PageCount = "100",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book3 in batch write"
            };
            Book book2 = new Book
            {
                Id = 903,
                InPublication = true,
                ISBN = "903-11-11-1111",
                PageCount = "200",
                Price = 10,
                ProductCategory = "Book",
                Title = "My book4 in batch write"
            };

            var bookBatch = context.CreateBatchWrite<Book>();
            bookBatch.AddPutItems(new List<Book> { book1, book2 });

            Console.WriteLine("Performing batch write in SingleTableBatchWrite().");
            bookBatch.Execute();
        }

        private static void MultiTableBatchWrite(DynamoDBContext context)
        {
            // 1. New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
            forumBatch.AddPutItem(newForum);
```

```
           // 2. New Thread item.
           Thread newThread = new Thread
           {
               ForumName = "S3 forum",
               Subject = "My sample question",
               KeywordTags = new List<string> { "S3", "Bucket" },
               Message = "Message text"
           };

           DynamoDBOperationConfig config = new DynamoDBOperationConfig();
           config.SkipVersionCheck = true;
           var threadBatch = context.CreateBatchWrite<Thread>(config);
           threadBatch.AddPutItem(newThread);
           threadBatch.AddDeleteKey("some partition key value", "some sort key 
value");

           var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);
           Console.WriteLine("Performing batch write in MultiTableBatchWrite().");
           superBatch.Execute();
       }
   }

   [DynamoDBTable("Reply")]
   public class Reply
   {
       [DynamoDBHashKey] //Partition key
       public string Id
       {
           get; set;
       }

       [DynamoDBRangeKey] //Sort key
       public DateTime ReplyDateTime
       {
           get; set;
       }

       // Properties included implicitly.
       public string Message
       {
           get; set;
       }
       // Explicit property mapping with object persistence model attributes.
```

```
        [DynamoDBProperty("LastPostedBy")]
        public string PostedBy
        {
            get; set;
        }
        // Property to store version number for optimistic locking.
        [DynamoDBVersion]
        public int? Version
        {
            get; set;
        }
    }

    [DynamoDBTable("Thread")]
    public class Thread
    {
        // PK mapping.
        [DynamoDBHashKey]      //Partition key
        public string ForumName
        {
            get; set;
        }
        [DynamoDBRangeKey]      //Sort key
        public String Subject
        {
            get; set;
        }
        // Implicit mapping.
        public string Message
        {
            get; set;
        }
        public string LastPostedBy
        {
            get; set;
        }
        public int Views
        {
            get; set;
        }
        public int Replies
        {
            get; set;
        }
        public bool Answered
        {
            get; set;
        }
        public DateTime LastPostedDateTime
        {
            get; set;
        }
        // Explicit mapping (property and table attribute names are different.
        [DynamoDBProperty("Tags")]
        public List<string> KeywordTags
        {
            get; set;
        }
        // Property to store version number for optimistic locking.
        [DynamoDBVersion]
        public int? Version
        {
            get; set;
        }
    }

    [DynamoDBTable("Forum")]
    public class Forum
    {
        [DynamoDBHashKey]      //Partition key
        public string Name
        {
            get; set;
        }
        // All the following properties are explicitly mapped,
        // only to show how to provide mapping.
        [DynamoDBProperty]
        public int Threads
        {
            get; set;
        }
        [DynamoDBProperty]
        public int Views
        {
            get; set;
        }
        [DynamoDBProperty]
        public string LastPostBy
        {
            get; set;
        }
        [DynamoDBProperty]
        public DateTime LastPostDateTime
        {
            get; set;
        }
        [DynamoDBProperty]
        public int Messages
        {
            get; set;
        }
    }

    [DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey] //Partition key
        public int Id
        {
            get; set;
        }
        public string Title
        {
            get; set;
        }
        public string ISBN
        {
            get; set;
        }
        public int Price
        {
            get; set;
        }
        public string PageCount
        {
            get; set;
        }
        public string ProductCategory
        {
            get; set;
        }
        public bool InPublication
```

```
            {
                get; set;
            }
        }
}
```

例: AWS SDK for .NET のオブジェクト永続性モデルを使用した、DynamoDB でのクエリおよびスキャン

このセクションの C# コード例では、次のクラスを定義して、DynamoDB 内のテーブルにマッピングしています。この例で使用されているテーブルの作成については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

- Book クラスは、ProductCatalog テーブルにマッピングされます。
- Forum、Thread、および Reply クラスは、同じ名前のテーブルにマッピングされます。

この例では DynamoDBContext を使用して、さらに次のクエリとスキャンオペレーションを実行しています。

- Id によって書籍を取得します。

  ProductCatalog テーブルには、プライマリキーとして Id があります。プライマリキーの一部にソートキーは含まれていません。したがって、テーブルのクエリを行うことはできません。項目は Id 値を使用して取得できます。
- Reply テーブルに対して次のクエリを実行します。(Reply テーブルのプライマリキーには、Id および ReplyDateTime 属性が含まれます。ReplyDateTime はソートキーで、これによりテーブルのクエリが実行されます。)
  - 過去 15 日間に投稿されたフォーラムスレッドに対する返信を検索します。
  - 特定の日付範囲の間に投稿されたフォーラムスレッドに対する返信を検索します。
- ProductCatalog テーブルをスキャンし、価格が 0 以下の書籍を検索します。

  パフォーマンス上の理由から、スキャンオペレーションではなくクエリオペレーションを使用するようにしてください。ただし、場合によってはテーブルをスキャンする必要があります。データ入力エラーがあり、書籍の価格の 1 つが 0 未満に設定されたとします。この例では、ProductCategory テーブルをスキャンして、価格が 0 未満である書籍項目 (ProductCategory は書籍) を検索しています。

実例を作成する方法については、「.NET コード例」を参照してください。

Note

以下の例は、同期メソッドをサポートしていないため、.NET Core コアには適用されません。詳細については、「.NET 用 AWS 非同期 API」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using System.Configuration;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class HighLevelQueryAndScan
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);
                // Get an item.
                GetBook(context, 101);

                // Sample forum and thread to test queries.
                string forumName = "Amazon DynamoDB";
                string threadSubject = "DynamoDB Thread 1";
                // Sample queries.
                FindRepliesInLast15Days(context, forumName, threadSubject);
                FindRepliesPostedWithinTimePeriod(context, forumName, threadSubject);

                // Scan table.
                FindProductsPricedLessThanZero(context);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void GetBook(DynamoDBContext context, int productId)
        {
            Book bookItem = context.Load<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine("Title: {0} \n No.Of threads:{1} \n No. of messages:
 {2}",
                      bookItem.Title, bookItem.ISBN, bookItem.PageCount);
        }

        private static void FindRepliesInLast15Days(DynamoDBContext context,
                                string forumName,
                                string threadSubject)
        {
            string replyId = forumName + "#" + threadSubject;
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            IEnumerable<Reply> latestReplies =
                context.Query<Reply>(replyId, QueryOperator.GreaterThan,
 twoWeeksAgoDate);
            Console.WriteLine("\nFindRepliesInLast15Days: Printing result.....");
            foreach (Reply r in latestReplies)
                Console.WriteLine("{0}\t{1}\t{2}\t{3}", r.Id, r.PostedBy, r.Message,
 r.ReplyDateTime);
        }

        private static void FindRepliesPostedWithinTimePeriod(DynamoDBContext context,
                                      string forumName,
                                      string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: Printing
 result.....");

            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

            IEnumerable<Reply> repliesInAPeriod = context.Query<Reply>(forumId,
                                           QueryOperator.Between, startDate, endDate);
            foreach (Reply r in repliesInAPeriod)
                Console.WriteLine("{0}\t{1}\t{2}\t{3}", r.Id, r.PostedBy, r.Message,
 r.ReplyDateTime);
        }

        private static void FindProductsPricedLessThanZero(DynamoDBContext context)
        {
            int price = 0;
            IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                new ScanCondition("Price", ScanOperator.LessThan, price),
                new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
                );
            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
 result.....");
            foreach (Book r in itemsWithWrongPrice)
                Console.WriteLine("{0}\t{1}\t{2}\t{3}", r.Id, r.Title, r.Price,
 r.ISBN);
        }
    }

    [DynamoDBTable("Reply")]
    public class Reply
    {
        [DynamoDBHashKey] //Partition key
        public string Id
        {
            get; set;
        }

        [DynamoDBRangeKey] //Sort key
        public DateTime ReplyDateTime
        {
            get; set;
        }

        // Properties included implicitly.
        public string Message
        {
            get; set;
        }
        // Explicit property mapping with object persistence model attributes.
        [DynamoDBProperty("LastPostedBy")]
        public string PostedBy
        {
            get; set;
        }
        // Property to store version number for optimistic locking.
        [DynamoDBVersion]
        public int? Version
        {
            get; set;
        }
    }

    [DynamoDBTable("Thread")]
    public class Thread
    {
        // Partition key mapping.
        [DynamoDBHashKey] //Partition key
        public string ForumName
        {
            get; set;
        }
        [DynamoDBRangeKey] //Sort key
        public DateTime Subject
        {
            get; set;
        }
        // Implicit mapping.
        public string Message
        {
            get; set;
        }
        public string LastPostedBy
        {
            get; set;
        }
        public int Views
        {
            get; set;
        }
        public int Replies
        {
            get; set;
        }
        public bool Answered
        {
            get; set;
        }
        public DateTime LastPostedDateTime
        {
            get; set;
        }
        // Explicit mapping (property and table attribute names are different).
        [DynamoDBProperty("Tags")]
        public List<string> KeywordTags
        {
            get; set;
        }
        // Property to store version number for optimistic locking.
        [DynamoDBVersion]
        public int? Version
        {
            get; set;
        }
    }

    [DynamoDBTable("Forum")]
    public class Forum
    {
        [DynamoDBHashKey]
        public string Name
        {
            get; set;
        }
        // All the following properties are explicitly mapped
        // to show how to provide mapping.
        [DynamoDBProperty]
        public int Threads
        {
            get; set;
        }
        [DynamoDBProperty]
        public int Views
        {
            get; set;
        }
        [DynamoDBProperty]
        public string LastPostBy
        {
            get; set;
        }
        [DynamoDBProperty]
        public DateTime LastPostDateTime
        {
            get; set;
        }
        [DynamoDBProperty]
        public int Messages
        {
            get; set;
        }
    }

    [DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey] //Partition key
        public int Id
        {
            get; set;
        }
        public string Title
        {
            get; set;
        }
        public string ISBN
        {
            get; set;
        }
        public int Price
        {
            get; set;
        }
        public string PageCount
        {
            get; set;
        }
        public string ProductCategory
        {
            get; set;
        }
        public bool InPublication
        {
            get; set;
        }
    }
}
```

このデベロッパーガイドのコード例の実行

AWS SDK では、次の言語で Amazon DynamoDB を幅広くサポートしています。

- Java
- ブラウザの JavaScript
- .NET
- Node.js
- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

このような言語を使用してすばやく開始する方法については、「DynamoDB および AWS SDK の使用開始」を参照してください。

この開発者ガイドのコード例は、以下のプログラミング言語を使用して DynamoDB オペレーションの詳細を示しています。

- Java コードの例
- .NET コード例

この演習を開始する前に、AWS アカウントを作成し、アクセスキーとシークレットキーを取得して、コンピュータで AWS Command Line Interface (AWS CLI) をセットアップします。詳細については、「」を参照してください DynamoDB (ウェブサービス) の設定

Note

DynamoDB のダウンロード可能バージョンを使用する場合は、AWS CLI を使用してテーブルとサンプルデータを作成する必要があります。また、各 `--endpoint-url` コマンドで AWS CLI パラメータも指定する必要があります。詳細については、「ローカルエンドポイントの設定」を参照してください。

DynamoDB でのコード例用のテーブルの作成とデータのロード

DynamoDB でのテーブルの作成、サンプルデータセットのロード、データに対するクエリの実行、およびデータの更新に関する基本については、以下を参照してください。

- ステップ 1: テーブルを作成します
- ステップ 2: コンソールまたは AWS CLI を使用して、テーブルにデータを書き込みます
- ステップ 3: テーブルからデータを読み込みます
- ステップ 4: テーブルのデータを更新します

Java コードの例

トピック

- Java: AWS 認証情報の設定
- Java: AWS リージョンとエンドポイントの設定

この開発者ガイドには、Java コードスニペットとすぐに使用できるプログラムが含まれています。これらのコード例は、次のセクションで確認することができます。

- 項目と属性の操作
- DynamoDB でのテーブルとデータの操作
- DynamoDB のクエリオペレーション
- DynamoDB でのスキャンの使用
- セカンダリインデックスを使用したデータアクセス性の向上
- Java 1.x: DynamoDBMapper
- DynamoDB Streams の変更データキャプチャ

Eclipse と AWS Toolkit for Eclipse を使用して、すぐに開始できます。フル機能の IDE に加えて、自動更新で AWS SDK for Java および AWS アプリケーションの構築用に事前設定されたテンプレートも取得できます。

Java サンプルコードを実行するには (Eclipse を使用)

1. Eclipse IDE をダウンロードし、インストールします。
2. のダウンロードおよびインストール。。AWS Toolkit for Eclipse
3. Eclipse を起動し、Eclipse メニューから、[File] (ファイル)、[New] (新規)、[Other] (その他) の順に選択します。
4. [Select a wizard] (ウィザードの選択) で、AWS、[AWS Java Project] (Java プロジェクト) の順に選択してから、[Next] (次へ) をクリックします。
5. [AWS Java の作成] で、次の操作を行います。

   a. [Project name] (プロジェクト名) にプロジェクトの名前を入力します。
   b. [Select Account] (アカウントの選択) リストから認証情報プロファイルを選択します。

      AWS Toolkit for Eclipse を初めて使用する場合、[AWS アカウントの設定] を選択して、AWS 認証情報を設定します。
6. [Finish] (完了) を選択してプロジェクトを作成します。
7. Eclipse メニューから [File] (ファイル)、[New] (新規)、[Class] (クラス) の順に選択します。
8. [Java Class] (Java クラス) の [Name] (名前) にクラスの名前を入力し (実行するコード例と同じ名前を使用)、[Finish] (完了) を選択してクラスを作成します。
9. ドキュメントページから Eclipse エディタにサンプルコードをコピーします。
10. コードを実行するには、Eclipse メニューの [Run] (実行) を選択します。

SDK for Java には、DynamoDB を操作するためにスレッドセーフなクライアントが用意されています。ベストプラクティスとして、ご利用のアプリケーションでクライアントを 1 つ作成し、そのクライアントをスレッド間で再利用します。

詳細については、AWS SDK for Java を参照してください。

> Note
>
> このガイドのサンプルコードは、最新バージョンの AWS SDK for Java で使用するためのものです。

AWS Toolkit for Eclipse を使用している場合、SDK for Java の自動更新を設定することができます。これを Eclipse で行うには、[Preferences] (基本設定) に移動し、AWS Toolkit、AWS SDK for Java、[Download new SDKs automatically] (新しい SDK を自動的にダウンロード) の順に選択します。

Java: AWS 認証情報の設定

SDK for Java では、ランタイムにアプリケーションに AWS 認証情報を指定する必要があります。このガイドのコード例では、AWS 認証情報ファイルを使用していることを前提としています。詳細については、「AWS SDK for Java デベロッパーガイド」の「開発用の AWS 認証情報のセットアップ」を参照してください。

`~/.aws/credentials` という名前の AWS 認証情報ファイルの例を次に示します。ここで、チルダ文字 (~) はホームディレクトリを表します。

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: AWS リージョンとエンドポイントの設定

デフォルトでは、コード例は、米国西部 (オレゴン) リージョンの DynamoDB にアクセスします。このリージョンを変更するには、`AmazonDynamoDB` プロパティを変更します。

次のサンプルコードは、新しい `AmazonDynamoDB` をインスタンス化します。

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

`withRegion` メソッドを使用して、利用可能な任意のリージョンで、DynamoDB を対象としてコードを実行できます。リージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS リージョンとエンドポイント」を参照してください。

ローカルコンピュータで DynamoDB を使用してサンプルコードを実行する場合は、エンドポイントを設定する必要があります。

AWS SDKV1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

AWS SDK V2

```
DynamoDbClient client = DynamoDbClient.builder()
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
 validation
    .region(Region.US_EAST_1)
    .credentialsProvider(StaticCredentialsProvider.create(
    AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

.NET コード例

トピック

- .NET: AWS 認証情報の設定
- .NET: AWS リージョンとエンドポイントの設定

このガイドには、.NET コードスニペットとすぐに使用できるプログラムが含まれています。これらのコード例は、次のセクションで確認することができます。

- 項目と属性の操作
- DynamoDB でのテーブルとデータの操作
- DynamoDB のクエリオペレーション
- DynamoDB でのスキャンの使用
- セカンダリインデックスを使用したデータアクセス性の向上
- .NET ドキュメントモデル
- .NET: オブジェクト永続性モデル

- DynamoDB Streams の変更データキャプチャ

AWS SDK for .NET と Toolkit for Visual Studio を使用すると、すぐに始めることができます。

.NET サンプルコードを実行するには (Visual Studio を使用)

1. Microsoft Visual Studio をダウンロードし、インストールします。
2. Toolkit for Visual Studio をダウンロードしてインストールします。
3. Visual Studio を起動します。[File] (ファイル)、[New] (新規)、[Project] (プロジェクト) の順に選択します。
4. [新しいプロジェクト] で、[空の AWS プロジェクト]、[OK] の順に選択します。
5. [AWS アクセス認証情報] で、[既存のプロファイルを使用] を選択し、リストから自分の認証情報の内容を選択して、[OK] を選択します。

   初めて Toolkit for Visual Studio を使用する場合は、[Use a new profile] (新しいプロファイルを使用) を選択し、AWS の認証情報を設定します。
6. Visual Studio プロジェクトで、プログラムソースコード (`Program.cs`) のタブを選択します。サンプルコードをドキュメントページから Visual Studio にコピーして、エディタに表示されている他のコードを置き換えます。
7. 「The type or namespace name...could not be found(名前空間のタイプが検出されません)」のような形式のエラーメッセージが表示された場合は、次のように DynamoDB の AWS SDK アセンブリをインストールする必要があります。

   a. ソリューションエクスプローラーで、プロジェクトのコンテキスト (右クリック) メニューを開いて、[Manage NuGet Packages] (NuGet パッケージの管理) を選択します。
   b. NuGet パッケージマネージャーで、[Browse] (参照) を選択します。
   c. 検索ボックスに「**`AWSSDK.DynamoDBv2`**」と入力し、完了するまで待ちます。
   d. AWSSDK.DynamoDBv2、[Install] (インストール) の順に選択します。
   e. インストールが完了したら、Program.cs タブを選択し、プログラムに戻ります。
8. コードを実行するには、Visual Studio ツールバーの [Start] (開始) を選択します。

AWS SDK for .NET には、DynamoDB を操作するためにスレッドセーフなクライアントが用意されています。ベストプラクティスとして、ご利用のアプリケーションでクライアントを 1 つ作成し、そのクライアントをスレッド間で再利用します。

詳細については、「AWS SDK for .NET」を参照してください。

Note

このガイドのサンプルコードは、最新バージョンの AWS SDK for .NET で使用するためのものです。

.NET: AWS 認証情報の設定

AWS SDK for .NET では、ランタイムにアプリケーションに AWS 認証情報を指定する必要があります。このガイドのコード例は、SDK ストアを使用して AWS 認証情報ファイルを管理することを前提としています。詳細については、「AWS SDK for .NET デベロッパーガイド」の「SDK ストアの使用」を参照してください。

Toolkit for Visual Studio では、任意の数のアカウントの複数セットの認証情報がサポートされています。各セットはプロファイルと呼ばれています。Visual Studio では、プロジェクトの App.config ファイルにエントリを追加するため、アプリケーションはランタイム時に AWS 認証情報を見つけることができます。

次の例は、Toolkit for Visual Studio を使用した新しいプロジェクトの作成時に生成されたデフォルトの App.config ファイルを示しています。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

ランタイム時に、このプログラムは AWSProfileName エントリで指定されているように、AWS 認証情報の default セットを使用します。AWS 認証情報自体は、暗号化されたフォームで SDK ストアに保持されます。Toolkit for Visual Studio は、すべて Visual Studio 内からの認証情報を管理するためのグラフィカルユーザーインターフェイスを使用します。詳細については、「AWS Toolkit for Visual Studio ユーザーガイド」の「認証情報の指定」を参照してください。

Note

デフォルトでは、コード例は、米国西部 (オレゴン) リージョンの DynamoDB にアクセスします。リージョンを変更するには、App.config ファイルで AWSRegion エントリを変更します。AWSRegion を DynamoDB が利用可能なすべてのリージョンに設定します。リージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS リージョンとエンドポイント」を参照してください。

.NET: AWS リージョンとエンドポイントの設定

デフォルトでは、コード例は、米国西部 (オレゴン) リージョンの DynamoDB にアクセスします。リージョンを変更するには、AWSRegion ファイルで App.config エントリを変更します。または、AmazonDynamoDBClient プロパティを変更してリージョンを変更することもできます。

次のサンプルコードは、新しい AmazonDynamoDBClient をインスタンス化します。クライアントは、別のリージョンでコードが DynamoDB に対して実行するように変更されます。

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

リージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「AWS リージョンとエンドポイント」を参照してください。

ローカルコンピュータで DynamoDB を使用してサンプルコードを実行する場合は、エンドポイントを設定する必要があります。

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Python と Boto3 による Amazon DynamoDB のプログラミング

このガイドは、Python で Amazon DynamoDB を使用したいと考えているプログラマーを対象としています。さまざまな抽象化レイヤー、設定管理、エラー処理、再試行ポリシーの制御、キープアライブの管理などについて説明します。

トピック

Boto について

Python から DynamoDB にアクセスするには、Python 用の公式 AWS SDK (一般に Boto3 と呼ばれる) を使用します。Boto (ボトと発音) という名前は、アマゾン川に自生する淡水イルカに由来しています。Boto3 ライブラリは, ライブラリの 3 番目のメジャーバージョンであり、2015 年に初めてリリースされました。Boto3 ライブラリは、DynamoDB だけでなくすべての AWS サービスをサポートするため、非常に大きいです。このオリエンテーションは、DynamoDB に関連する Boto3 の部分のみを対象としています。

Boto は、GitHub でホストされるオープンソースプロジェクトとして AWS によって管理および公開されています。Botocore と Boto3 の 2 つのパッケージに分かれています。

- Botocore は低レベルの機能を備えています。Botocore には、クライアント、セッション、認証情報、設定、および例外クラスがあります。
- Boto3 は Botocore の上に構築されています。よりハイレベルで、より Python 的なインターフェイスを提供します。具体的には、DynamoDB テーブルをリソースとして公開し、低レベルのサービス指向のクライアントインターフェイスよりもシンプルで洗練されたインターフェイスを提供します。

これらのプロジェクトは GitHub でホストされているため、ソースコードを表示したり、未解決の問題を追跡したり、独自の問題を送信したりできます。

Boto ドキュメントの使用

以下のリソースを使って Boto ドキュメントを使い始めましょう。

- まず、パッケージインストールの確実な出発点となる「クイックスタート」セクションから始めてください。まだ Boto3 をインストールしていない場合は、そちらを参照してください (Boto3 は多くの場合、AWS Lambda などの AWS サービス内で自動的に利用可能になります)。
- その後、ドキュメントの DynamoDB ガイドに注目してください。テーブルの作成と削除、項目の操作、バッチ操作の実行、クエリの実行、スキャンの実行など、基本的な DynamoDB アクティビティを実行する方法を示しています。例ではリソースインターフェイスを使用しています。`boto3.resource('dynamodb')` が表示されれば、上位レベルのリソースインターフェイスを使用していることがわかります。
- ガイドを読み終えたら、DynamoDB リファレンスを確認できます。このランディングページには、使用できるクラスとメソッドがすべて記載されています。上部には `DynamoDB.Client` クラスが表示されます。これにより、コントロールプレーンとデータプレーンのすべての操作に低レベルでアクセスできます。一番下にある `DynamoDB.ServiceResource` クラスを見てください。これは上位レベルの Python インターフェイスです。これを使用すると、テーブルを作成したり、複数のテーブルにわたってバッチ操作を実行したり、テーブル固有のアクションのために `DynamoDB.ServiceResource.Table` インスタンスを取得したりできます。

クライアントとリソースの抽象化レイヤーを理解する

使用する 2 つのインターフェイスは、クライアントインターフェイスとリソースインターフェイスです。

- 低レベルのクライアントインターフェイスは、基盤となるサービス API に 1 対 1 で対応します。DynamoDB が提供するすべての API は、クライアントを通じて利用できます。つまり、クライアントインターフェイスは完全な機能を提供できますが、多くの場合、より冗長で使い方が複雑です。
- 上位レベルのリソースインターフェイスは、基盤となるサービス API と 1 対 1 で対応するわけではありません。ただし、`batch_writer` などのサービスへのアクセスが便利になります。

クライアントインターフェイスを使用して項目を挿入する例を次に示します。すべての値が、型を示すキー (文字列は 'S'、数値は 'N') と文字列としての値のマップとして渡されることに注目してください。これは DynamoDB JSON 形式と呼ばれます。

```
import boto3

dynamodb = boto3.client('dynamodb')

dynamodb.put_item(
    TableName='YourTableName',
    Item={
        'pk': {'S': 'id#1'},
        'sk': {'S': 'cart#123'},
        'name': {'S': 'SomeName'},
        'inventory': {'N': '500'},
        # ... more attributes ...
    }
)
```

リソースインターフェイスを使用した同じ `PutItem` 操作を次に示します。データ型の指定は暗黙的です。

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

table.put_item(
    Item={
        'pk': 'id#1',
        'sk': 'cart#123',
        'name': 'SomeName',
        'inventory': 500,
        # ... more attributes ...
    }
)
```

必要に応じて、boto3 に用意されている `TypeSerializer` および `TypeDeserializer` クラスを使用して、通常の JSON と DynamoDB JSON の間で変換できます。

```
def dynamo_to_python(dynamo_object: dict) -> dict:
    deserializer = TypeDeserializer()
    return {
        k: deserializer.deserialize(v)
        for k, v in dynamo_object.items()
    }

def python_to_dynamo(python_object: dict) -> dict:
    serializer = TypeSerializer()
    return {
        k: serializer.serialize(v)
        for k, v in python_object.items()
    }
```

クライアントインターフェイスを使用してクエリを実行する方法は次のとおりです。クエリは JSON コンストラクトとして表現されます。キーワードの競合が発生する可能性がある場合は、変数置換が必要な `KeyConditionExpression` 文字列を使用します。

```
import boto3

client = boto3.client('dynamodb')

# Construct the query
response = client.query(
    TableName='YourTableName',
    KeyConditionExpression='pk = :pk_val AND begins_with(sk, :sk_val)',
    FilterExpression='#name = :name_val',
    ExpressionAttributeValues={
        ':pk_val': {'S': 'id#1'},
        ':sk_val': {'S': 'cart#'},
        ':name_val': {'S': 'SomeName'},
    },
    ExpressionAttributeNames={
        '#name': 'name',
    }
)
```

リソースインターフェイスを使った同じクエリ操作を短くして簡略化できます。

```
import boto3
from boto3.dynamodb.conditions import Key, Attr
```

```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

response = table.query(
    KeyConditionExpression=Key('pk').eq('id#1') & Key('sk').begins_with('cart#'),
    FilterExpression=Attr('name').eq('SomeName')
)
```

最後の例として、テーブルのおおよそのサイズ (テーブルに保持され、約 6 時間ごとに更新されるメタデータ) を取得したいとします。クライアントインターフェイスで、describe_table() 操作を行い、返された JSON 構造から回答を引き出します。

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

リソースインターフェイスでは、テーブルは describe 操作を暗黙的に実行し、データを属性として直接表示します。

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

Note

クライアントインターフェイスとリソースインターフェイスのどちらを使用して開発するかを検討する際、リソースドキュメントではリソースインターフェイスに新しい機能が追加されないことに注意してください。「AWS Python SDK チームは boto3 のリソースインターフェイスに新しい機能を追加する予定はありません。既存のインターフェイスは boto3 のライフサイクル中も引き続き動作します。顧客はクライアントインターフェイスを通じて新しいサービス機能にアクセスできます。」

テーブルリソース batch_writer の使用

上位レベルのテーブルリソースでのみ利用できる便利な機能の 1 つは、batch_writer です。DynamoDB はバッチ書き込み操作をサポートしており、1 つのネットワークリクエストで最大 25 件の入力または削除操作が可能です。このようなバッチ処理は、ネットワークラウンドトリップを最小限に抑えることで効率性を高めます。

低レベルのクライアントライブラリでは、client.batch_write_item() 操作を使用してバッチを実行します。作業は手動で 25 個のバッチに分割する必要があります。各操作の後に、未処理の項目のリストを受け取るように要求する必要もあります (書き込み操作の中には成功するものもあれば、失敗するものもあります)。その後、それらの未処理項目を後の batch_write_item() 操作に再度渡す必要があります。大量のボイラープレートコードがあります。

Table.batch_writer メソッドは、オブジェクトをバッチで書き込むためのコンテキストマネージャーを作成します。一度に 1 つずつ項目を書き込んでいるように見えても、内部的には項目をバッファリングしてバッチで送信するインターフェイスを提供します。また、未処理の項目の再試行も暗黙的に処理されます。

```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

movies = # long list of movies in {'pk': 'val', 'sk': 'val', etc} format
with table.batch_writer() as writer:
    for movie in movies:
        writer.put_item(Item=movie)
```

クライアントレイヤーとリソースレイヤーを調べるその他のコード例

次のコードサンプルリポジトリを参照して、クライアントとリソースの両方を使用してさまざまな機能の使用法を調べることもできます。

- AWS 公式シングルアクションのコード例。
- AWS 公式のシナリオ指向コード例。
- コミュニティが管理するシングルアクションのコード例。

Client オブジェクトと Resource オブジェクトがセッションやスレッドとどのように相互作用するかを理解する

Resource オブジェクトはスレッドセーフではないため、スレッドやプロセス間で共有しないでください。詳細については、Resource に関するガイドを参照してください。

これとは対照的に、Client オブジェクトは、特定の高度な機能を除いて、通常はスレッドセーフです。詳細については、Clients に関するガイドを参照してください。

Session オブジェクトはスレッドセーフではありません。そのため、マルチスレッド環境で Client または Resource を作成するたびに、まず、新しい Session を作成してから、その Session から Client または Resource を作成する必要があります。詳細については、Sessions に関するガイドを参照してください。

boto3.resource() を呼び出すとき、暗黙的にデフォルトの Session を使用することになります。これはシングルスレッドのコードを書くのに便利です。マルチスレッドのコードを書くときには、まず、スレッドごとに新しい Session を構築し、その Session からリソースを取得します。

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Config オブジェクトのカスタマイズ

Client または Resource オブジェクトを作成するときに、オプションの名前付きパラメータを渡して動作をカスタマイズできます。config という名前のパラメータは、さまざまな機能をアンロックします。これは botocore.client.Config のインスタンスであり、Config のリファレンスドキュメントには、ユーザーが制御できるすべての情報が記載されています。設定ガイドには、概要がわかりやすく書かれています。

> ⓘ Note
>
> これらの動作設定の多くは、Session レベル、AWS 構成ファイル内、または環境変数として変更できます。

タイムアウトの設定

カスタム設定の用途の 1 つは、ネットワークの動作を調整することです。

- connect_timeout (float または int) — 接続を試みたときにタイムアウト例外がスローされるまでの時間 (秒単位)。デフォルト値は 60 秒です。
- read_timeout (float または int) — 接続からの読み取りを試みたときにタイムアウト例外がスローされるまでの時間 (秒単位)。デフォルト値は 60 秒です。

DynamoDB では 60 秒のタイムアウトは過剰です。つまり、一時的なネットワークの不具合により、クライアントが再試行できるようになるまでに 1 分間の遅延が生じます。次のコードはタイムアウトを 1 秒に短縮します。

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

タイムアウトの詳細については、「レイテンシーを考慮した DynamoDB アプリケーションのための AWS Java SDK HTTP リクエスト設定のチューニング」を参照してください。Java SDK には Python よりも多くのタイムアウト設定があることに注意してください。

キープアライブの設定

botocore 1.27.84 以降を使用している場合は、TCP キープアライブを制御することもできます。

- tcp_keepalive (bool) - `True` に設定された場合、新しい接続の作成時に使用される TCP キープアライブソケットオプションを有効にします (デフォルトは `False`)。これは botocore 1.27.84 以降でのみ使用可能です。

TCP キープアライブを `True` に設定すると、平均レイテンシーを減らすことができます。以下は、適切な botocore バージョンを使用している場合に TCP キープアライブを条件付きで true に設定するサンプルコードです。

```
import botocore
import boto3
from botocore.config import Config
from distutils.version import LooseVersion
```

```
required_version = "1.27.84"
current_version = botocore.__version__

my_config = Config(
   connect_timeout = 0.5,
   read_timeout = 0.5
)
if LooseVersion(current_version) > LooseVersion(required_version):
    my_config = my_config.merge(Config(tcp_keepalive = True))

dynamodb = boto3.resource('dynamodb', config=my_config)
```

> ⓘ Note
>
> TCP キープアライブは HTTP キープアライブとは異なります。TCP キープアライブでは、接続を維持してドロップを即座に検出するために、基盤となるオペレーティングシステムからソケット接続を介して小さなパケットが送信されます。HTTP キープアライブでは、基盤となるソケット上に構築された Web 接続が再利用されます。HTTP キープアライブは boto3 では常に有効になっています。

アイドル状態の接続を維持できる時間には制限があります。接続がアイドル状態で、次のリクエストではすでに確立されている接続を使いたい場合は、定期的に (たとえば 1 分ごとに) リクエストを送信することを検討してください。

再試行の設定

この設定では、希望する再試行動作を指定できる retries という辞書も使用できます。SDK がエラーを受信し、そのエラーが一時的なタイプの場合、SDK 内で再試行が行われます。エラーが内部で再試行された場合（そして再試行によって最終的に成功の応答が得られた場合）、呼び出し側のコードから見ると、エラーは発生せず、レイテンシーがわずかに増加するだけです。指定できる値は次のとおりです。

- max_attempts — 1 回のリクエストで行われる最大再試行回数を表す整数。たとえば、この値を 2 に設定すると、最初のリクエストから最大 2 回リクエストが再試行されます。この値を 0 に設定すると、最初のリクエスト以降は再試行されません。
- total_max_attempts — 1 回のリクエストで行われる最大再試行合計回数を表す整数。これには最初のリクエストが含まれるため、値が 1 の場合はリクエストが再試行されないことを示します。`total_max_attempts` と `max_attempts` の両方が指定された場

合、total_max_attempts が優先されます。total_max_attempts は、AWS_MAX_ATTEMPTS 環境変数と max_attempts 設定ファイルの値にマップされるので、max_attempts より優先されます。

- mode — botocore が使用すべき再試行モードの種類を表す文字列。有効な値は次のとおりです。
  - legacy — デフォルトモード。最初の再試行は 50 ms 待機し、基本係数 2 でエクスポネンシャルバックオフを使用します。DynamoDB の場合、最大試行回数は合計 10 回です (上記でオーバーライドされない限り)。

    > Note
    >
    > エクスポネンシャルバックオフでは、最後の試行ではほぼ 13 秒待機します。

  - standard — 他の AWS SDK との整合性が高いことから、standard と名付けられています。最初の再試行時には、0 ミリ秒から 1,000 ミリ秒までのランダムな時間だけ待ちます。再試行が必要な場合は、0 ミリ秒から 1,000 ミリ秒までの別のランダムな時間を選択し、その値に 2 を掛けます。さらに再試行が必要な場合は、同じようにランダムな時間を選択し、これに 4 を掛け、試行を続けます。各待機時間の上限は 20 秒です。このモードは、検出された障害条件の数が legacy モードよりも多い場合に再試行を実行します。DynamoDB の場合、最大試行回数は合計 3 回です (上記でオーバーライドされない限り)。
  - adaptive – standard モードのすべての機能を含みながら、クライアント側の自動スロットリングを含む実験的な再試行モード。適応型レート制限を使用すると、SDK は AWS のサービスの容量をより適切に処理するために、リクエストの送信速度を遅くすることができます。これは暫定モードであり、動作が変わる可能性があります。

これらの再試行モードの拡張定義は、再試行ガイドと SDK リファレンスの「再試行動作」トピックに記載されています。

次の例は、legacy 再試行ポリシーを明示的に使用して、合計リクエスト 3 回 (再試行 2 回) までにした例です。

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0,
   retries = {
     'mode': 'legacy',
        'total_max_attempts': 3
    }
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

DynamoDB は可用性が高く、レイテンシーが低いシステムであるため、組み込みの再試行ポリシーで許可されているよりも積極的に再試行の速度を上げる必要がある場合があります。boto3 に頼って暗黙的な再試行を行うのではなく、最大試行回数を 0 に設定し、自分で例外をキャッチし、必要に応じて独自のコードから再試行することで、独自の再試行ポリシーを実装できます。

独自の再試行ポリシーを管理する場合は、スロットルとエラーを区別しておきましょう。

- スロットル (`ProvisionedThroughputExceededException` または `ThrottlingException` で示される) は、DynamoDB テーブルまたはパーティションの読み取り容量または書き込み容量を超過したことを通知する正常なサービスを示します。1 ミリ秒が経過するごとに、読み取りまたは書き込み容量が少しずつ増えるため、すぐに (50 ミリ秒ごとなど) 再試行して、新しく解放された容量へのアクセスを試みることができます。スロットルについては、エクスポネンシャルバックオフは特に必要ありません。軽量で DynamoDB が返しやすく、リクエストごとの課金も発生しないためです。エクスポネンシャルバックオフでは、すでに最長の待機時間が過ぎたクライアントスレッドの遅延をさらに長くしていくため、統計的に p50 と p99 が外側に広がります。
- エラー (`InternalServerError` または `ServiceUnavailable` などによって示される) は、サービスに一時的な問題があることを示します。これはテーブル全体の場合もあれば、読み取り元または書き込み先のパーティションだけに関する場合もあります。エラーが発生した場合は、再試行の前に少し長く一時停止し (250 ミリ秒や 500 ミリ秒など)、ジッターを使って再試行をずらすことができます。

最大プール接続数の設定

最後に、この構成では接続プールのサイズを制御できます。

- max_pool_connections (int) — 接続プールに保持できる最大接続数。この値が設定されていない場合、デフォルト値の 10 が使用されます。

このオプションは、プールして再利用できるように保持する HTTP 接続の最大数を制御します。Session ごとに異なるプールが保持されます。10 個を超えるスレッドが、同じ Session から構築されたクライアントやリソースに対して実行されることが予想される場合は、プールされた接続を使用する他のスレッドに対してスレッドが待機する必要がないように、この値を増やすことを検討すべきです。

```
import boto3
from botocore.config import Config

my_config = Config(
   max_pool_connections = 20
)

# Setup a single session holding up to 20 pooled connections
session = boto3.Session(my_config)

# Create up to 20 resources against that session for handing to threads
# Notice the single-threaded access to the Session and each Resource
resource1 = session.resource('dynamodb')
resource2 = session.resource('dynamodb')
# etc
```

エラー処理

Boto3 では AWS サービスの例外がすべて静的に定義されているわけではありません。これは、AWS サービスのエラーや例外は大きく異なり、変更される可能性があるためです。Boto3 はすべてのサービス例外を `ClientError` としてラップし、詳細を構造化された JSON として公開します。例えば、エラーレスポンスは以下のような構造になっているかもしれません。

```
{
    'Error': {
        'Code': 'SomeServiceException',
        'Message': 'Details/context around the exception or error'
    },
    'ResponseMetadata': {
        'RequestId': '1234567890ABCDEF',
        'HostId': 'host ID data will appear here as a hash',
        'HTTPStatusCode': 400,
        'HTTPHeaders': {'header metadata key/values will appear here'},
        'RetryAttempts': 0
    }
}
```

次のコードは、すべての `ClientError` 例外をキャッチし、`Error` 内の `Code` の文字列値を調べて実行するアクションを決定します。

```
import botocore
import boto3

dynamodb = boto3.client('dynamodb')

try:
    response = dynamodb.put_item(...)

except botocore.exceptions.ClientError as err:
    print('Error Code: {}'.format(err.response['Error']['Code']))
    print('Error Message: {}'.format(err.response['Error']['Message']))
    print('Http Code: {}'.format(err.response['ResponseMetadata']['HTTPStatusCode']))
    print('Request ID: {}'.format(err.response['ResponseMetadata']['RequestId']))

    if err.response['Error']['Code'] in ('ProvisionedThroughputExceededException',
 'ThrottlingException'):
        print("Received a throttle")
    elif err.response['Error']['Code'] == 'InternalServerError':
        print("Received a server error")
    else:
        raise err
```

一部の (すべてではない) 例外コードは最上位クラスとして実装されています。これらを直接処理することもできます。Client インターフェイスを使用する場合、これらの例外はクライアントに動的に入力され、以下のようにクライアントインスタンスを使用してこれらの例外をキャッチします。

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Resource インターフェイスを使用しているときには、以下のように、`.meta.client` を使用してリソースから基盤となる Client までトラバースして例外にアクセスする必要があります。

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

マテリアライズド例外タイプのリストを確認するには、リストを動的に生成できます。

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

条件式を使用して書き込み操作を行う場合、式が失敗した場合にその項目の値をエラーレスポンスで返すようにリクエストできます。

```
try:
    response = table.put_item(
        Item=item,
        ConditionExpression='attribute_not_exists(pk)',
        ReturnValuesOnConditionCheckFailure='ALL_OLD'
    )
except table.meta.client.exceptions.ConditionalCheckFailedException as e:
    print('Item already exists:', e.response['Item'])
```

エラー処理と例外について詳しくは、以下を参照してください。

- エラー処理技術の詳細については、エラー処理に関する boto3 ガイドを参照してください。
- DynamoDB 開発者ガイドのプログラミングエラーに関するセクションには、発生する可能性のあるエラーが記載されています。
- API リファレンスの「一般的なエラー」セクション。
- 各 API 操作のドキュメントには、呼び出しによって発生する可能性のあるエラー (例: BatchWriteItem) が記載されています。

ログ記録

boto3 ライブラリは Python の組み込みロギングモジュールと統合されており、セッション中に何が起こるかを追跡できます。ロギングレベルを制御するには、ロギングモジュールを設定します。

```
import logging

logging.basicConfig(level=logging.INFO)
```

これにより、INFO 以上のレベルのメッセージを記録するようにルートロガーが設定されます。レベルよりも重大度の低いロギングメッセージは無視されます。ロギングレベルには、DEBUG、INFO、WARNING、ERROR、および CRITICAL があります。デフォルトは WARNING です。

boto3 のロガーは階層構造になっています。ライブラリはいくつかの異なるロガーを使用しており、それぞれがライブラリの異なる部分に対応しています。それぞれの動作を個別に制御できます。

- boto3: boto3 モジュールのメインロガー。
- botocore: botocore パッケージのメインロガー。
- botocore.auth: リクエストの AWS 署名作成を記録するために使用されます。

- botocore.credentials: 認証情報の取得と更新のプロセスを記録するために使用されます。
- botocore.endpoint: ネットワーク経由で送信される前に、リクエストの作成を記録するために使用されます。
- botocore.hooks: ライブラリでトリガーされたイベントの記録に使用されます。
- botocore.loaders: AWS サービスモデルの一部がロードされたときの記録に使用されます。
- botocore.parsers: AWS サービスレスポンスを解析する前に記録するために使用されます。
- botocore.retryhandler: AWS サービスリクエストの再試行の処理を記録するために使用されます (レガシーモード)。
- botocore.retries.standard: AWS サービスリクエストの再試行の処理を記録するために使用されます (standard または adaptive モード)。
- botocore.utils: ライブラリ内のさまざまなアクティビティを記録するために使用されます。
- botocore.waiter: 特定の状態になるまで AWS サービスにポーリングするウェイターの機能をログに記録するために使用されます。

他のライブラリもログに記録されます。内部的には、boto3 は HTTP 接続処理にサードパーティの urllib3 を使用しています。レイテンシーが重要な場合は、ログを監視して、urllib3 が新しい接続を確立したり、アイドル状態の接続を閉じたりするタイミングを確認することで、プールが十分に利用されているかどうかを確認できます。

- urllib3.connectionpool: 接続プールの処理イベントを記録するために使用されます。

以下のコードスニペットは、ほとんどのロギングを INFO に設定し、エンドポイントと接続プールのアクティビティの記録には DEBUG に設定しています。

```
import logging

logging.getLogger('boto3').setLevel(logging.INFO)
logging.getLogger('botocore').setLevel(logging.INFO)
logging.getLogger('botocore.endpoint').setLevel(logging.DEBUG)
logging.getLogger('urllib3.connectionpool').setLevel(logging.DEBUG)
```

イベントフック

Botocore は実行中のさまざまな段階でイベントを発生させます。これらのイベントのハンドラーを登録すると、イベントが発生するたびにハンドラーが呼び出されます。これにより、内部を変更しなくても botocore の動作を拡張できます。

例えば、アプリケーションの DynamoDB テーブルで PutItem 操作が呼び出されるたびに記録しておきたいとしましょう。'provide-client-params.dynamodb.PutItem' イベントに登録すると、関連する Session で PutItem 操作が呼び出されるたびにキャッチしてログに記録できます。例を示します。

```
import boto3
import botocore
import logging

def log_put_params(params, **kwargs):
    if 'TableName' in params and 'Item' in params:
        logging.info(f"PutItem on table {params['TableName']}: {params['Item']}")

logging.basicConfig(level=logging.INFO)

session = boto3.Session()
event_system = session.events

# Register our interest in hooking in when the parameters are provided to PutItem
event_system.register('provide-client-params.dynamodb.PutItem', log_put_params)

# Now, every time you use this session to put an item in DynamoDB,
# it will log the table name and item data.
dynamodb = session.resource('dynamodb')
table = dynamodb.Table('YourTableName')
table.put_item(
    Item={
        'pk': '123',
        'sk': 'cart#123',
        'item_data': 'YourItemData',
        # ... more attributes ...
    }
)
```

ハンドラー内で、パラメータをプログラムで操作して動作を変更することもできます。

```
params['TableName'] = "NewTableName"
```

イベントについて詳しくは、イベントに関する botocore ドキュメントと、イベントに関する boto3 ドキュメントを参照してください。

ページネーションとページネーター

Query や Scan などの一部のリクエストでは、1 回のリクエストで返されるデータのサイズが制限され、後続のページを取得するにはリクエストを繰り返し行う必要があります。

各ページで読み取られる項目の最大数は、`limit` パラメータを使用して制御できます。例えば、最後の 10 項目だけが必要な場合、`limit` を使用して最後の 10 項目だけを取得できます。制限は、フィルタリングが適用される前にテーブルから読み取られる量であることに注意してください。フィルタリング後に正確に 10 を指定する方法はありません。事前にフィルター処理される回数を制御し、実際に 10 個を取得した時点でクライアント側で確認することしかできません。制限にかかわらず、すべての応答の最大サイズは常に 1 MB です。

応答に `LastEvaluatedKey` が含まれている場合、それは件数またはサイズの制限に達したために応答が終了したことを示します。キーはレスポンスで評価された最後のキーです。この `LastEvaluatedKey` を取得して，フォローアップコールに `ExclusiveStartKey` として渡すと、その開始点から次のチャンクを読み取ることができます。`LastEvaluatedKey` が返されないということは、Query または Scan に一致するアイテムがもうないということです。

以下は、1 ページあたり最大 100 個の項目を読み込み、すべての項目が読み込まれるまでループする単純な例です (Resource インターフェイスを使用していますが、Client インターフェイスでも同じパターンです)。

```
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

query_params = {
    'KeyConditionExpression': Key('pk').eq('123') & Key('sk').gt(1000),
    'Limit': 100
}

while True:
    response = table.query(**query_params)

    # Process the items however you like
    for item in response['Items']:
        print(item)

    # No LastEvaluatedKey means no more items to retrieve
    if 'LastEvaluatedKey' not in response:
        break

    # If there are possibly more items, update the start key for the next page
    query_params['ExclusiveStartKey'] = response['LastEvaluatedKey']
```

便宜上、boto3 はページネーターを使ってこれを行うことができます。ただし、Client インターフェイスでのみ機能します。以下はページネーターを使うように書き直されたコードです。

```
import boto3

dynamodb = boto3.client('dynamodb')

paginator = dynamodb.get_paginator('query')

query_params = {
    'TableName': 'YourTableName',
    'KeyConditionExpression': 'pk = :pk_val AND sk > :sk_val',
    'ExpressionAttributeValues': {
        ':pk_val': {'S': '123'},
        ':sk_val': {'N': '1000'},
    },
    'Limit': 100
}

page_iterator = paginator.paginate(**query_params)

for page in page_iterator:
    # Process the items however you like
    for item in page['Items']:
        print(item)
```

詳細については、「ページネーターに関するガイド」と「DynamoDB.Paginator.Query の API リファレンス」を参照してください。

> Note
>
> ページネーターには、`MaxItems`、`StartingToken`、および `PageSize` という名前の独自の構成設定もあります。DynamoDB でページ分割を行う場合は、これらの設定は無視してください。

ウェイター

ウェイターは、処理が完了するのを待ってから次の処理を進めることができます。現在のところ、サポートしているのはテーブルが作成または削除されるのを待つことだけです。バックグラウンドでは、ウェイター操作は 20 秒ごとに最大 25 回までチェックを行います。これは自分で行うこともできますが、オートメーションを書くときはウェイターを使うのが便利です。

次のコードは、特定のテーブルが作成されるまで待機する方法を示しています。

```
# Create a table, wait until it exists, and print its ARN
response = client.create_table(...)
waiter = client.get_waiter('table_exists')
waiter.wait(TableName='YourTableName')
print('Table created:', response['TableDescription']['TableArn']
```

詳細については、「ウェイターガイド」と「ウェイターに関するリファレンス」を参照してください。

JavaScript による Amazon DynamoDB のプログラミング

このガイドは、JavaScript で Amazon DynamoDB を使用したいと考えているプログラマーを対象としています。AWS SDK for JavaScript、利用可能な抽象化レイヤー、接続の設定、エラー処理、再試行ポリシーの定義、キープアライブの管理などについて説明します。

トピック

- エラー処理
- ログ記録
- 考慮事項

AWS SDK for JavaScript について

AWS SDK for JavaScript を使用すると、ブラウザスクリプトまたは Node.js のいずれかで AWS の サービス にアクセスできます。このドキュメントでは、最新バージョンの SDK (V3) を主に取り上げます。AWS SDK for JavaScript V3 は、AWS が管理するオープンソースのプロジェクトであり、GitHub でホストされています。Issue や Feature のリクエストは公開されており、GitHub リポジトリの [Issues] ページからアクセスできます。

JavaScript V2 は V3 と似ていますが、構文が違います。V3 の方がモジュール性が高いため、依存関係をより小さくして配布することが容易で、TypeScript のサポートも充実しています。最新バージョンの SDK を使用することをお勧めします。

AWS SDK for JavaScript V3 を使用する

Node Package Manager を使用して SDK を Node.js アプリケーションに追加できます。以下の例は、DynamoDB の操作に一般的に使われる SDK パッケージの追加方法を示しています。

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`
- `npm install @aws-sdk/util-dynamodb`

パッケージをインストールすると、package.json プロジェクトファイルの依存関係セクションに参照が追加されます。新しい ECMAScript モジュール構文を使用することもできます。これら 2 つのアプローチの詳細については、「考慮事項」セクションを参照してください。

JavaScript のドキュメントを参照する

まずは、JavaScript のドキュメントを確認しましょう。以下のリソースを参照してください。

- JavaScript の主要ドキュメントについては、「デベロッパーガイド」を参照してください。インストール手順は「Setting up」セクションに記載されています。
- API リファレンスドキュメントを参照し、使用可能なすべてのクラスとメソッドを確認してください。

- SDK for JavaScript は DynamoDB 以外にも数多くの AWS のサービス をサポートしています。以下の手順にそって、DynamoDB の特定の API カバレッジを調べてください。

  1. [Services] から [DynamoDB and Libraries] を選択します。これは、低レベルクライアントをドキュメント化したものです。
  2. lib-dynamodb を選択します。これは、高レベルクライアントをドキュメント化したものです。これら 2 つのクライアントは、2 つの異なる抽象化レイヤーを表しています。任意で選んで使用できます。抽象レイヤーの詳細については、以下のセクションを参照してください。

抽象化レイヤー

SDK for JavaScript V3 には、低レベルのクライアント (`DynamoDBClient`) と高レベルのクライアント (`DynamoDBDocumentClient`) があります。

トピック

- 低レベルクライアント (DynamoDBClient)
- 高レベルクライアント (DynamoDBDocumentClient)

低レベルクライアント (**`DynamoDBClient`**)

低レベルクライアントでは、基盤のワイヤプロトコルが別段抽象化されません。通信をあらゆる側面から完全に制御できますが、抽象化されていないため、項目定義の提供などの操作は、DynamoDB JSON 形式を使って行う必要があります。

以下の例に示すように、この形式ではデータ型を明示的に指定する必要があります。S は文字列値を示し、N は数値を示します。ネットワーク上の数値は、精度が損なわれないように、必ず数値型としてタグ付けされた文字列として送信されます。低レベルの API コールには、`PutItemCommand` や `GetItemCommand` などの命名規則があります。

次の例では、低レベルクライアントを使用し、DynamoDB JSON で `Item` を定義しています。

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      "id": { S: "Product01" },
      "description": { S: "Hiking Boots" },
      "category": { S: "footwear" },
      "sku": { S: "hiking-sku-01" },
      "size": { N: "9" }
    }
  };

  try {
    const data = await client.send(new PutItemCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

高レベルクライアント (`DynamoDBDocumentClient`)

高レベルの DynamoDB ドキュメントクライアントには、データを手動でマーシャリングする必要がない、標準の JavaScript オブジェクトを使用して直接読み書きできるなど、便利な機能が組み込まれています。lib-dynamodb のドキュメントには、利点が一覧で紹介されています。

`DynamoDBDocumentClient` をインスタンス化するには、まず、低レベルの `DynamoDBClient` を構築し、それを `DynamoDBDocumentClient` でラップします。関数の命名規則は 2 つのパッケージ間で若干異なります。例えば、低レベルでは `PutItemCommand` を使用し、高レベルでは `PutCommand` を使用します。名前が違えば、両方の関数セットが同じコンテキストで共存可能です。つまり、同一スクリプト内で両方を組み合わせることができます。

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    },
  };

  try {
    const data = await docClient.send(new PutCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

addProduct();
```

`GetItem`、`Query`、`Scan` などの API オペレーションを使用して項目を読み取る場合の使用パターンは同じです。

marshall ユーティリティ関数を使用する

低レベルクライアントを使用して、データ型を自分でマーシャリングまたはアンマーシャリングできます。ユーティリティパッケージ util-dynamodb には、JSON を受け入れて DynamoDB JSON を生成する `marshall()` ユーティリティ関数と、その逆を行う `unmarshall()` 関数があります。次の例では、低レベルクライアントを使用し、`marshall()` を呼び出してデータマーシャリングを処理しています。

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
const { marshall } = require("@aws-sdk/util-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: marshall({
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    }),
  };

  try {
    const data = await client.send(new PutItemCommand(params));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

項目の読み込み

DynamoDB から項目を 1 つ読み込むには、`GetItem` API オペレーションを使用します。`PutItem` コマンドと同様に、低レベルのクライアントまたは高レベルのドキュメントクライアントのいずれかを選んで使用できます。以下の例は、高レベルのドキュメントクライアントを使用して項目を取得する方法を示しています。

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, GetCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function getProduct() {
  const params = {
    TableName: "products",
    Key: {
      id: "Product01",
    },
  };

  try {
    const data = await docClient.send(new GetCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

getProduct();
```

Query API オペレーションを使用して、複数の項目を読み込みます。低レベルのクライアントまたはドキュメントクライアントを使用できます。以下の例では、高レベルのドキュメントクライアントを使用しています。

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  QueryCommand,
} = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function productSearch() {
  const params = {
    TableName: "products",
    IndexName: "GSI1",
    KeyConditionExpression: "#category = :category and begins_with(#sku, :sku)",
    ExpressionAttributeNames: {
      "#category": "category",
      "#sku": "sku",
    },
    ExpressionAttributeValues: {
      ":category": "footwear",
      ":sku": "hiking",
    },
  };

  try {
    const data = await docClient.send(new QueryCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

productSearch();
```

条件付きの書き込み

DynamoDB の書き込みオペレーションでは、論理条件式を指定できます。条件式が true と評価されないと、書き込みは続行されません。条件の評価が true にならなかった場合は、例外が生成されます。条件式では、該当する項目が既に存在するかどうかや、その属性が特定の制約に一致するかどうかを確認できます。

```
ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"
```

条件式が失敗した場合は、ReturnValuesOnConditionCheckFailure を使用して、条件を満たさなかった項目をエラーレスポンスに含めるようにリクエストできるため、問題の原因究明に役立ちます。詳細については、「Handle conditional write errors in high concurrency scenarios with Amazon DynamoDB」を参照してください。

```
try {
      const response = await client.send(new PutCommand({
          TableName: "YourTableName",
          Item: item,
          ConditionExpression: "attribute_not_exists(pk)",
          ReturnValuesOnConditionCheckFailure: "ALL_OLD"
      }));
  } catch (e) {
      if (e.name === 'ConditionalCheckFailedException') {
          console.log('Item already exists:', e.Item);
      } else {
          throw e;
      }
  }
```

JavaScript SDK V3 のその他の使用法を示す追加のコード例は、JavaScript SDK V3 ドキュメントと DynamoDB-SDK-Examples GitHub リポジトリでご覧ください。

ページ分割

トピック

- 便利な paginateScan メソッドを使用する

Scan や Query などの読み取りリクエストでは、データセット内の複数の項目が返される場合があります。Limit パラメータを指定して Scan や Query を実行した場合は、システムで多数の項目が

全部読み取られた後、一部のレスポンスが送信されます。追加の項目を取得するには、ページ分割が必要です。

システムは、1 回のリクエストにつき最大 1 MB のデータのみを読み取ります。Filter 式を含めた場合も、システムは最大 1 MB のデータをディスクから読み取りますが、その 1 MB の中から、指定したフィルターに一致した項目だけを返します。フィルターオペレーションでは 1 ページあたりに返される項目数が 0 個になる場合もありますが、それでも、さらにページ分割をしないと、最後まで検索できません。

データの取得を継続するには、レスポンスで LastEvaluatedKey を探し、それを後続のリクエストの ExclusiveStartKey パラメータに指定する必要があります。これが、以下の例に示すように、ブックマークの役割を果たします。

Note

この例では、1 回目の反復で null lastEvaluatedKey が ExclusiveStartKey として渡されています。これは許容されています。

LastEvaluatedKey を使用する例:

```
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScan() {
  let lastEvaluatedKey;
  let pageCount = 0;

  do {
    const params = {
      TableName: "products",
      ExclusiveStartKey: lastEvaluatedKey,
    };

    const response = await client.send(new ScanCommand(params));
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, response.Items);
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);
}
```

```
paginatedScan().catch((err) => {
  console.error(err);
});
```

便利な **paginateScan** メソッドを使用する

SDK には、paginateScan と paginateQuery という便利なメソッドがあります。これらは、ページ分割を自動で行い、繰り返しのリクエストをバックグラウンドで処理してくれます。標準の Limit パラメータを使用して、1 回のリクエストで読み取る項目の最大数を指定します。

```
const { DynamoDBClient, paginateScan } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScanUsingPaginator() {
  const params = {
    TableName: "products",
    Limit: 100
  };

  const paginator = paginateScan({client}, params);

  let pageCount = 0;

  for await (const page of paginator) {
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, page.Items);
  }
}

paginatedScanUsingPaginator().catch((err) => {
  console.error(err);
});
```

> Note
>
> テーブルが小さくない限り、テーブル全体のスキャンを定期的に実行することは、アクセスパターンとして推奨されません。

構成を指定する

トピック

- タイムアウトの設定
- キープアライブの設定
- 再試行の設定

DynamoDBClient を設定する際に、構成オブジェクトをコンストラクタに渡すことで、さまざまな構成オーバーライドを指定できます。例えば、接続先のリージョン (呼び出し側のコンテキストが把握していない場合) や、使用するエンドポイント URL を指定できます。開発目的で DynamoDB Local インスタンスをターゲットにする場合に便利です。

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

タイムアウトの設定

DynamoDB では、クライアント/サーバー通信に HTTPS を使用します。HTTP レイヤーの一部の側面は、NodeHttpHandler オブジェクトを指定することで制御できます。例えば、主要なタイムアウト値である connectionTimeout や requestTimeout を調整できます。connectionTimeout は、クライアントが接続を試みる際の最長待機時間 (ミリ秒単位) です。この時間内に確立できない場合は、接続を断念します。

requestTimeout では、リクエストが送信されてからクライアントがレスポンスを待機する時間 (ミリ秒単位) を定義します。いずれもデフォルト値は 0 です。その場合、タイムアウトは無効になり、レスポンスが届かなければクライアントは制限なく待ち続けることになります。ネットワークに問題が発生した場合にリクエストがエラーになり、新しいリクエストを開始できるように、妥当なタイムアウト値を設定しておいた方が賢明です。例:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";

const requestHandler = new NodeHttpHandler({
  connectionTimeout: 2000,
  requestTimeout: 2000,
});
```

```
const client = new DynamoDBClient({
  requestHandler
});
```

> Note
>
> この例では Smithy をインポートしています。Smithy はサービスと SDK を定義するための言語で、AWS が管理し、オープンソースで公開しています。

タイムアウト値を設定するほかに、最大ソケット数を設定し、オリジンあたりの同時接続数を増やすことができます。デベロッパーガイドでは、maxSockets パラメータの設定について詳しく解説しています。

キープアライブの設定

HTTPS を使用する場合、最初のリクエストでは、安全な接続を確立するために何往復かの通信が必要になります。HTTP キープアライブを使用すると、すでに確立された接続を後続のリクエストで再利用できるため、リクエストの効率が上がり、レイテンシーが短縮されます。JavaScript V3 では、HTTP キープアライブがデフォルトで有効になっています。

アイドル状態の接続を維持できる時間には制限があります。接続がアイドル状態になるが、すでに確立されている接続を次のリクエストで利用したい場合は、定期的に (たとえば 1 分ごとに) リクエストを送信することを検討してください。

> Note
>
> SDK の旧バージョン V2 では、キープアライブはデフォルトで無効になっているため、各接続は使用後すぐに切断されます。V2 を使用している場合は、この設定をオーバーライドできます。

再試行の設定

SDK がエラーレスポンスを受信し、そのエラーを SDK が再開可能であると判断した場合 (スロットリング例外や一時的なサービス例外など)、再試行されます。呼び出し側にはわからない形で行われますが、リクエスト成功までの時間が長引くという症状は現れます。

SDK for JavaScript V3 は、デフォルトではリクエストを合計 3 回行います。それで成功しなければそれ以上は再試行せず、呼び出し側のコンテキストにエラーを渡します。こうした再試行の回数と頻度を調整できます。

DynamoDBClient コンストラクタには、試行回数を制限する maxAttempts を設定できます。以下の例では、その値がデフォルトの 3 から合計 5 に引き上げられています。0 または 1 に設定した場合、自動再試行は不要であり、再開可能なエラーはキャッチブロック内で手動で処理するという意味になります。

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

また、再試行のタイミングも、カスタムの再試行戦略で制御できます。その場合は、util-retry ユーティリティパッケージをインポートし、現在の再試行が何回目かに応じて再試行の間隔を計算するカスタムのバックオフ関数を作成します。

以下の例では、1 回目の試行が失敗した場合は、15、30、90、360 ミリ秒と徐々に遅延を増やしながら最大 5 回試行するように指定しています。カスタムのバックオフ関数 calculateRetryBackoff は、再試行回数 (初回の再試行が 1 で、以降増分) に応じて遅延を計算し、該当するリクエストで待機するミリ秒数を返します。

```
const { ConfiguredRetryStrategy } = require("@aws-sdk/util-retry");

const calculateRetryBackoff = (attempt) => {
  const backoffTimes = [15, 30, 90, 360];
  return backoffTimes[attempt - 1] || 0;
};

const client = new DynamoDBClient({
  retryStrategy: new ConfiguredRetryStrategy(
    5, // max attempts.
    calculateRetryBackoff // backoff function.
  ),
});
```

ウェーター

DynamoDB クライアントには、便利なウェーター関数が 2 つあります。テーブルの作成、変更、削除時に、テーブルの変更が完了するまでコードの処理を待機させたい場合に、これらの関数を使用で

きます。例えば、テーブルをデプロイして waitUntilTableExists 関数を呼び出すと、テーブルが ACTIVE になるまでコードがブロックされます。ウェーターは内部的に 20 秒ごとに describe-table を実行し、DynamoDB サービスをポーリングします。

```
import {waitUntilTableExists, waitUntilTableNotExists} from "@aws-sdk/client-dynamodb";

… <create table details>

const results = await waitUntilTableExists({client: client, maxWaitTime: 180},
 {TableName: "products"});
if (results.state == 'SUCCESS') {
  return results.reason.Table
}
console.error(`${results.state} ${results.reason}`);
```

waitUntilTableExists 関数は、describe-table コマンドを実行でき、テーブルのステータスが ACTIVE と表示された場合にのみ制御を返します。そのため、waitUntilTableExists を使用して、テーブルの作成や、GSI インデックスの追加などの変更の完了を待つことができます。こうした変更には時間がかかり、終わってからテーブルが ACTIVE ステータスに戻ります。

エラー処理

ここで最初に紹介した数例では、すべてのエラーを網羅してキャッチしています。しかし、実際のアプリケーションでは、さまざまなエラータイプを見分け、より緻密なエラー処理を実装することが重要です。

DynamoDB エラーレスポンスには、エラーの名前を含むメタデータが含まれています。エラーをキャッチし、エラー条件の文字列名候補と照合して、処理方法を決定できます。サーバー側のエラーについては、@aws-sdk/client-dynamodb パッケージによってエクスポートされたエラータイプで instanceof 演算子を利用して、エラー処理を効率的に管理できます。

こうしたエラーは、再試行回数をすべて使い切って初めて露呈する点に注意が重要です。エラーが再試行され、最終的に呼び出しが成功した場合、コードの観点ではエラーは発生せず、レイテンシーが若干長引いただけになります。再試行は、スロットルリクエストやエラーリクエストなど、失敗したリクエストとして Amazon CloudWatch のグラフに表示されます。クライアントは再試行回数が最大回数に達すると、例外を生成します。つまり、クライアントはそれ以上再試行しないということがわかります。

以下のスニペットは、エラーをキャッチし、返されたエラーのタイプに基づいてアクションを実行します。

```
import {
  ResourceNotFoundException
  ProvisionedThroughputExceededException,
  DynamoDBServiceException,
} from "@aws-sdk/client-dynamodb";

try {
  await client.send(someCommand);
} catch (e) {
    if (e instanceof ResourceNotFoundException) {
      // Handle ResourceNotFoundException
    } else if (e instanceof ProvisionedThroughputExceededException) {
      // Handle ProvisionedThroughputExceededException
    } else if (e instanceof DynamoDBServiceException) {
      // Handle DynamoDBServiceException
    } else {
      // Other errors such as those from the SDK
      if (e.name === "TimeoutError") {
        // Handle SDK TimeoutError.
      } else {
        // Handle other errors.
      }
    }
}
```

一般的なエラー文字列については、「DynamoDB デベロッパーガイド」の「the section called “エラー処理”」を参照してください。特定の API コールで発生する可能性のある具体的なエラーについては、Query API のドキュメントなど、該当する API コールのドキュメントに記載されています。

エラーのメタデータには、エラーによって異なりますが、追加のプロパティが含まれます。TimeoutError の場合、以下に示すように、試行回数と totalRetryDelay がメタデータに含まれます。

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

独自の再試行ポリシーを管理する場合は、スロットルとエラーを区別しておきましょう。

- スロットル (ProvisionedThroughputExceededException または ThrottlingException で示される) は、DynamoDB テーブルまたはパーティションの読み取り容量または書き込み容量を超過したことを通知する正常なサービスを示します。1 ミリ秒が経過するごとに、読み取りまたは書き込みの容量が少しずつ増えるため、すぐに (50 ミリ秒ごとなど) 再試行して、新しく解放された容量へのアクセスを試みることができます。

  スロットルについては、エクスポネンシャルバックオフは特に必要ありません。軽量で DynamoDB が返しやすく、リクエストごとの課金も発生しないためです。エクスポネンシャルバックオフでは、すでに最長の待機時間が過ぎたクライアントスレッドの遅延をさらに長くしていくため、統計的に p50 と p99 が外側に広がります。
- エラー (InternalServerError やServiceUnavailable などで示される) が発生した場合は、サービス (テーブル全体、または読み取り/書き込み先のパーティションなど) に一時的な問題があります。エラーについては、再試行する前に比較的長く (250 ミリ秒や 500 ミリ秒など) 一時停止したり、ジッターを追加して再試行をずらしたりできます。

ログ記録

ログ記録を有効にすると、SDK による処理内容を詳しく把握できます。以下の例に示すように、DynamoDBClient にパラメータを設定できます。ステータスコードや消費容量などのメタデータを含む詳細なログ情報がコンソールに表示されます。コードをターミナルウィンドウでローカルに実行した場合は、そのウィンドウにログが表示されます。AWS Lambda でコードを実行した場合、Amazon CloudWatch Logs が設定されていれば、コンソール出力がそのログに書き込まれます。

```
const client = new DynamoDBClient({
  logger: console
});
```

また、内部 SDK アクティビティにフックして、特定のイベントが発生したときにカスタムのログ記録を実行することもできます。以下の例では、クライアントの middlewareStack を使用して、各リクエストを SDK から送信された時点でインターセプトし、発生時にログに記録します。

```
const client = new DynamoDBClient({});

client.middlewareStack.add(
  (next) => async (args) => {
    console.log("Sending request from AWS SDK", { request: args.request });
    return next(args);
  },
  {
    step: "build",
    name: "log-ddb-calls",
  }
);
```

MiddlewareStack は、SDK の動作を観察して制御するための強力なフックを提供します。詳細については、ブログ「Introducing Middleware Stack in Modular AWS SDK for JavaScript」を参照してください。

考慮事項

プロジェクトに AWS SDK for JavaScript を実装する際には、さらに以下の点を考慮する必要があります。

モジュールシステム

> SDK は CommonJS と ES (ECMAScript) の 2 つのモジュールシステムに対応しています。CommonJS は require 関数を使用し、ES は import キーワードを使用します。
>
> 1. CommonJS – const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
> 2. ES (ECMAScript – import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";
>
> プロジェクトタイプは package.json ファイルの type セクションで指定され、使用するモジュールシステムを決定します。デフォルトは CommonJS です。"type": "module" を使用して、ES プロジェクトを指定します。CommonJS パッケージ形式を使用する既存の Node.JS プロジェクトがある場合でも、関数ファイルに .mjs 拡張子の付いた名前を指定することで、より新しい SDK V3 Import 構文を使用して関数を追加できます。そうすることで、コードファイルを ES (ECMAScript) として扱うことができます。

非同期オペレーション

> コールバックとプロミスを使用して DynamoDB オペレーションの結果を処理するサンプルコードが多数あります。最近の JavaScript では、そのように複雑な処理は必要なく、デベロッパーはもっと簡潔で読みやすい async/await 構文を活用して非同期オペレーションを実行できます。

ウェブブラウザランタイム

React または React Native を使用してビルドしているウェブ開発者やモバイル開発者は、各自のプロジェクトで SDK for JavaScript を使用できます。SDK の旧バージョン V2 では、ウェブ開発者は https://sdk.amazonaws.com/js/ でホストされている SDK イメージを参照して、SDK 全体をブラウザにロードする必要がありました。

V3 では、必要な V3 クライアントモジュールと必要なすべての JavaScript 関数を Webpack を使用して単一の JavaScript ファイルにバンドルし、HTML ページの <head> のスクリプトタグに追加できます。詳しくは、SDK ドキュメントの「Getting started in a browser script」セクションで説明しています。

DAX データプレーンオペレーション

SDK for JavaScript V3 は、現時点では Amazon DynamoDB Streams Accelerator (DAX) データプレーンオペレーションには対応していません。DAX サポートが必要な場合は、DAX データプレーン操作に対応している SDK for JavaScript V2 の使用を検討してください。

AWS SDK for Java 2.x を使用した Amazon DynamoDB のプログラミング

このガイドは、Java を使って Amazon DynamoDB を使用することを検討しているプログラマーを対象としています。このガイドでは、抽象化レイヤー、設定管理、エラー処理、再試行ポリシーの制御、キープアライブの管理など、さまざまな概念について説明します。

トピック

- ログ記録
- ページ分割
- データクラス注釈

AWS SDK for Java 2.x について

DynamoDB には、公式の AWS SDK for Java を使用して Java からアクセスできます。SDK for Java には、1.x と 2.x の 2 つのバージョンがあります。1.x については、2024 年 1 月 12 日にサポート終了が発表されました。1.x は、2024 年 7 月 31 日にメンテナンスモードとなり、サポートは 2025 年 12 月 31 日に終了する予定です。新規開発には、2018 年に最初にリリースされた 2.x を使用することを強くお勧めします。このガイドは 2.x のみを対象としており、SDK の DynamoDB との関連性がある部分のみを重点的に取り扱っています。

SDK のメンテナンスとサポートの詳細については、「AWS SDK とツールのリファレンスガイド」の「AWS SDK とツールのメンテナンスポリシー」と「AWS SDK とツールのバージョンのサポートマトリックス」のトピックを参照してください。

AWS SDK for Java 2.x では、Java 8 で導入されたノンブロッキング I/O などの最新の Java 機能をサポートするために 1.x のコードベースが大幅に書き直されています。SDK for Java 2.x では、ネットワーク接続の柔軟性と構成可能性を高めるために、プラガブル HTTP クライアント実装のサポートも追加され、ネットワーク接続の柔軟性と設定しやすさが向上しています。

SDK for Java 1.x から SDK for Java 2.x への主な変更点は、パッケージ名の変更です。Java 1.x SDK は、`com.amazonaws` のパッケージ名を使用していますが、Java 2.x SDK は `software.amazon.awssdk` を使用します。同様に、Java 1.x SDK の Maven アーティファクトは groupId `com.amazonaws` を使用していますが、Java 2.x SDK のアーティファクトは groupId `software.amazon.awssdk` を使用します。

> **Important**
>
> AWS SDK for Java 1.x には、`com.amazonaws.dynamodbv2` という名前の DynamoDB パッケージがあります。このパッケージ名の v2 は Java v2 であることを示すものではありません。v2 は、パッケージが低レベル API の元のバージョンではなく、DynamoDB 低レベル API の 2 番目のバージョンをサポートしていることを示します。

Java バージョンのサポート

AWS SDK for Java 2.x は、Java リリースの長期サポート (LTS) を完全にサポートします。

AWS SDK for Java 2.x の開始方法

次のチュートリアルでは、SDK for Java 2.x の依存関係を定義するために Apache Maven を使用する方法を説明しています。このチュートリアルでは、DynamoDB に接続して使用可能な DynamoDB テーブルを一覧表示するコードの記述方法も説明します。このチュートリアルは「AWS SDK for Java 2.x の使用を開始する」に基づいています。このチュートリアルは、Amazon S3 の代わりに DynamoDB を呼び出すように編集されています。

このチュートリアルを完了するには、次のステップを実行します。

- ステップ 1: このチュートリアルのために設定する
- ステップ 2: プロジェクトを作成する
- ステップ 3: コードを記述する
- ステップ 4: アプリケーションを構築して実行する

ステップ 1: このチュートリアルのために設定する

このチュートリアルを開始する前に、以下を実行する必要があります

- Amazon DynamoDB へのアクセス許可
- AWS IAM Identity Center へのシングルサインオンを使用して AWS のサービス へアクセスするように設定された Java 開発環境

「設定の概要」の手順に従って、このチュートリアルの設定を行います。Java SDK の「シングルサインオンアクセスを使用して開発環境を設定」し、「アクティブなAWSアクセスポータルセッション」ができた後、このチュートリアルの「ステップ 2」に進みます。

ステップ 2: プロジェクトを作成する

このチュートリアル用のプロジェクトを作成するには、プロジェクトの設定方法に関する入力を求める Maven コマンドを実行します。すべての入力を終えて確定すると、Maven は `pom.xml` ファイルとスタブ Java ファイルを作成して、プロジェクトの構築を完了します。

1. ターミナルまたはコマンドプロンプトウィンドウを開き、`Desktop` や `Home` フォルダなど、任意のディレクトリに移動します。

2. ターミナルに以下のコマンドを入力して、`Enter` を押します。

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. 各プロンプトの 2 列目にリストされている値を入力します。

プロンプト	入力する値
`Define value for property 'service':`	`dynamodb`
`Define value for property 'httpClient'` ：	`apache-client`
`Define value for property 'nativeImage'` ：	`false`
`Define value for property 'credentialProvider'`	`identity-center`
`Define value for property 'groupId':`	`org.example`
`Define value for property 'artifactId':`	`getstarted`
`Define value for property 'version' 1.0-SNAPSHOT:`	*`<Enter>`*
`Define value for property 'package' org.example:`	*`<Enter>`*

4. 最後の値を入力すると、Maven は選択した内容を一覧表示します。*Y* を入力して確認するか、*N* を入力して値を再入力します。

Maven は、入力した getstarted 値に基づいて artifactId という名前が付けられたプロジェクトフォルダーを作成します。getstarted フォルダー内で、レビューできる README.md ファイル、pom.xml ファイル、および src ディレクトリを検索します。

Maven は以下のディレクトリーツリーを構築します。

```
getstarted
 ### README.md
 ### pom.xml
 ### src
     ### main
     #   ### java
     #   #   ### org
     #   #       ### example
     #   #           ### App.java
     #   #           ### DependencyFactory.java
     #   #           ### Handler.java
     #   ### resources
     #       ### simplelogger.properties
     ### test
         ### java
             ### org
                 ### example
                     ### HandlerTest.java

 10 directories, 7 files
```

以下は、pom.xml プロジェクトファイルの内容を示しています。

pom.xml

dependencyManagement セクションには AWS SDK for Java 2.x への依存関係が含まれており、dependencies セクションには Amazon DynamoDB に対する依存関係があります。このような依存関係を指定すると、Maven は関連する jar ファイルを Java クラスパスに強制的に含めます。デフォルトでは、AWS SDK にはすべての AWS のサービス クラスは含まれません。DynamoDB の場合、低レベルのインターフェイスを使用するなら dynamodb アーティファクトに依存する必要があり、高レベルのインターフェイスを使用するなら dynamodb-enhanced に依存する必要があります。上記の関連する依存関係が含まれていないと、コードはコンパイルされません。プロジェクトでは Java 1.8 が使用されています。これは、maven.compiler.source、maven.compiler.target およびプロパティに 1.8 値があるためです。

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>org.example</groupId>
     <artifactId>getstarted</artifactId>
     <version>1.0-SNAPSHOT</version>
     <packaging>jar</packaging>
     <properties>
         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
         <maven.compiler.source>1.8</maven.compiler.source>
         <maven.compiler.target>1.8</maven.compiler.target>
         <maven.shade.plugin.version>3.2.1</maven.shade.plugin.version>
         <maven.compiler.plugin.version>3.6.1</maven.compiler.plugin.version>
         <exec-maven-plugin.version>1.6.0</exec-maven-plugin.version>
         <aws.java.sdk.version>2.22.0</aws.java.sdk.version> <--------- SDK version
 picked up from archetype version.
         <slf4j.version>1.7.28</slf4j.version>
         <junit5.version>5.8.1</junit5.version>
     </properties>

     <dependencyManagement>
         <dependencies>
             <dependency>
                 <groupId>software.amazon.awssdk</groupId>
                 <artifactId>bom</artifactId>
                 <version>${aws.java.sdk.version}</version>
                 <type>pom</type>
                 <scope>import</scope>
             </dependency>
         </dependencies>
     </dependencyManagement>

     <dependencies>
         <dependency>
             <groupId>software.amazon.awssdk</groupId>
             <artifactId>dynamodb</artifactId>  <--------- DynamoDB dependency
             <exclusions>
                 <exclusion>
                     <groupId>software.amazon.awssdk</groupId>
                     <artifactId>netty-nio-client</artifactId>
```

```
                </exclusion>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>apache-client</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sso</artifactId> <--------- Required for identity center 
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>ssooidc</artifactId> <--------- Required for identity center 
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>apache-client</artifactId> <--------- HTTP client specified.
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
        <!-- Needed to adapt Apache Commons Logging used by Apache HTTP Client to
 Slf4j to avoid
        ClassNotFoundException: org.apache.commons.logging.impl.LogFactoryImpl during
 runtime -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Test Dependencies -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>${junit5.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven.compiler.plugin.version}</version>
            </plugin>
        </plugins>
    </build>

 </project>
```

ステップ 3: コードを記述する

次のコードは Maven によって作成された App クラスを示しています。main メソッドはアプリケーションへのエントリポイントであり、Handler クラスのインスタンスを作成してその sendRequest メソッドを呼び出します。

App クラス

```
package org.example;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 public class App {
     private static final Logger logger = LoggerFactory.getLogger(App.class);

     public static void main(String... args) {
         logger.info("Application starts");

         Handler handler = new Handler();
         handler.sendRequest();

         logger.info("Application ends");
     }
 }
```

Maven が作成する `DependencyFactory` クラスには、[`DynamoDbClient`](#) インスタンスを構築して返す `dynamoDbClient` ファクトリメソッドが含まれています。`DynamoDbClient` インスタンスは Apache ベースの HTTP クライアントのインスタンスを使用します。これは、どの HTTP クライアントを使用するかを Maven が求めたときに `apache-client` が指定されたためです。

`DependencyFactory` は次のコードに示されています。

DependencyFactory クラス

```
package org.example;

 import software.amazon.awssdk.http.apache.ApacheHttpClient;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

 /**
  * The module containing all dependencies required by the {@link Handler}.
  */
 public class DependencyFactory {

     private DependencyFactory() {}

     /**
      * @return an instance of DynamoDbClient
      */
     public static DynamoDbClient dynamoDbClient() {
         return DynamoDbClient.builder()
                        .httpClientBuilder(ApacheHttpClient.builder())
                        .build();
     }
 }
```

`Handler` クラスにはプログラムのメインロジックが含まれています。`Handler` のインスタンスが `App` クラス内で作成されると、`DependencyFactory` は `DynamoDbClient` サービスクライアントを配置します。コードでは、`DynamoDbClient` インスタンスを使用して DynamoDB サービスを呼び出します。

Maven は *TODO* コメント付きの次の `Handler` クラスを生成します。チュートリアルの次のステップでは、*TODO* をコードに置き換えます。

Maven によって生成された **`Handler`** クラス

```
package org.example;

 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;


 public class Handler {
     private final DynamoDbClient dynamoDbClient;

     public Handler() {
         dynamoDbClient = DependencyFactory.dynamoDbClient();
     }

     public void sendRequest() {
         // TODO: invoking the API calls using dynamoDbClient.
     }
 }
```

ロジックを設定するには、`Handler` クラスのすべてのコンテンツを次のコードに置き換えます。`sendRequest` メソッドが入力され、必要なインポートが追加されます。

Handler クラス、実装済み

次のコードでは、[`DynamoDbClient`]インスタンスを使用して既存のテーブルのリストを取得します。特定のアカウントとリージョンにテーブルが配置されている場合、コードはこの `Logger` インスタンスを使用してこのようなテーブルの名前をログ記録します。

```
package org.example;

 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
 import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
```

```
public class Handler {
    private final DynamoDbClient dynamoDbClient;

    public Handler() {
        dynamoDbClient = DependencyFactory.dynamoDbClient();
    }

    public void sendRequest() {
        Logger logger = LoggerFactory.getLogger(Handler.class);

        logger.info("calling the DynamoDB API to get a list of existing tables");
        ListTablesResponse response = dynamoDbClient.listTables();

        if (!response.hasTableNames()) {
            logger.info("No existing tables found for the configured account &
region");
        } else {
            response.tableNames().forEach(tableName -> logger.info("Table: " +
tableName));
        }
    }
}
```

ステップ 4: アプリケーションを構築して実行する

プロジェクトが作成され、完全な Handler クラスを含めたら、アプリケーションを構築して実行します。

1. IAM Identity Center セッションがアクティブであることを確認してください。そのためには、AWS Command Line Interface コマンド aws sts get-caller-identity を実行してレスポンスを確認します。アクティブなセッションがない場合は、「AWS CLI を使用してサインインする」の手順を参照してください。
2. ターミナルまたはコマンドプロンプトウィンドウを開いて、プロジェクトディレクトリ getstarted に移動します。
3. プロジェクトを構築するには、以下のコマンドを使用します。

```
mvn clean package
```

4. 次のコマンドを使用して、アプリケーションを実行します。

```
mvn exec:java -Dexec.mainClass="org.example.App"
```

ファイルを表示した後、オブジェクトを削除し、その後にバケットを削除します。

成功

Maven プロジェクトがエラーなしで構築および実行された場合は、正常に完了しています! Java 2.x 用 SDK を使用して最初の Java アプリケーションを正常に構築しました。

クリーンアップ

このチュートリアルで作成されたリソースをクリーンアップするには、以下を行います。

- プロジェクトフォルダ getstarted を削除します。

AWS SDK for Java 2.x ドキュメントの使用

このAWS SDK for Java 2.xドキュメントには、すべての AWS のサービス にわたる SDK の側面すべてをカバーしています。次のトピックを出発点として使用してください。

- バージョン 1.x から 2.x への移行 — 1.x と 2.x の違いについての詳細な説明が記載されています。このトピックでは、両方のメジャーバージョンを並行して使用する方法も説明されています。
- Java 2.x SDK 向け DynamoDB ガイド — テーブルの作成、項目の操作、項目の取得など、基本的な DynamoDB のオペレーションを実行する方法を説明しています。上記の例では、低レベルインターフェイスを使用しています。「サポートされているインターフェイス」セクションで説明するとおり、Java には複数のインターフェイスがあります。

> Tip
>
> AWS SDK for Java 2.x ドキュメントのこのようなトピックを確認したら、Javadoc ドキュメントをブックマークすることをお勧めします。このドキュメントは、すべての AWS のサービス をカバーしており、API の主要リファレンスとして利用できます。

サポートされているインターフェイス

AWS SDK for Java 2.x は、必要とされる抽象化レベルに応じて、次のインターフェイスをサポートします。

このセクションのトピック

低レベルインターフェイス

低レベルインターフェイスは、基盤となるサービス API に 1 対 1 でマッピングされています。すべての DynamoDB API は、このインターフェイスを介して利用できます。つまり、低レベルのインターフェイスでは完全な機能を提供するとはいえ、多くの場合、使い方はより冗長で複雑になります。例えば、文字列を保持するには .s() 関数を使用し、数値を保持するには n() 関数を使用する必要があります。次の PutItem の例では、低レベルインターフェイスを使用して項目を挿入しています。

```
import org.slf4j.*;
import software.amazon.awssdk.http.crt.AwsCrtHttpClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class PutItem {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT = DynamoDbClient.create();
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemResponse response = DYNAMODB_CLIENT.putItem(PutItemRequest.builder()
                .item(Map.of(
                        "pk", AttributeValue.builder().s("123").build(),
                        "sk", AttributeValue.builder().s("cart#123").build(),
                        "item_data",
 AttributeValue.builder().s("YourItemData").build(),
                        "inventory", AttributeValue.builder().n("500").build()
                        // ... more attributes ...
                ))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .tableName("YourTableName")
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

高レベルインターフェイス

AWS SDK for Java 2.x の高レベルインターフェイスは DynamoDB 拡張クライアントと呼ばれます。このインターフェイスでは、より慣用的なコード作成エクスペリエンスが提供されます。

拡張クライアントでは、クライアント側のデータクラスと、そのデータを保存するように設計された複数の DynamoDB テーブル間でマッピングできます。テーブルおよび対応するモデルクラスの間の関係をコードで定義します。これにより、SDK を利用してデータ型の操作を管理できます。拡張クライアントの詳細については、「AWS SDK for Java 2.x」ドキュメントの「DynamoDB 拡張クライアントの API」を参照してください。

次の PutItem の例では、高レベルインターフェイスを使用してしています。この例では、YourItem という名前の DynamoDbBean が、TableSchema を作成して、putItem() コールの入力として直接使用できるようにしています。

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName", TableSchema.fromBean(YourItem.class));
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourItem.class)
                .item(new YourItem("123", "cart#123", "YourItemData", 500))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String itemData, int inventory) {
            this.pk = pk;
            this.sk = sk;
            this.itemData = itemData;
            this.inventory = inventory;
        }

        private String pk;
        private String sk;
        private String itemData;

        private int inventory;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }
```

```
        public void setItemData(String itemData) {
            this.itemData = itemData;
        }

        public String getItemData() {
            return itemData;
        }

        public void setInventory(int inventory) {
            this.inventory = inventory;
        }

        public int getInventory() {
            return inventory;
        }
    }
}
```

AWS SDK for Java 1.x には独自の高レベルインターフェイスがあり、メインクラス DynamoDBMapper で頻繁に参照されます。AWS SDK for Java 2.x は software.amazon.awssdk.enhanced.dynamodb という名前の別のパッケージ (と Maven アーティファクト) で公開されています。Java 2.x SDK は、メインクラス DynamoDbEnhancedClient で頻繁に参照されます。

イミュータブルデータクラスを使用する高レベルインターフェイス

DynamoDB 拡張クライアント API のマッピング機能は、イミュータブルデータクラスで動作します。イミュータブルクラスには getter しかないため、SDK がクラスのインスタンスを作成するために使用する Builder クラスが必要です。Java のイミュータビリティは、デベロッパーが副作用のないクラスを作成するためによく利用されるスタイルです。これにより、複雑なマルチスレッドアプリケーションでの動作の予測が容易になります。「High-level interface example」で説明したとおり、@DynamoDbBean アノテーションを使用する代わりに、イミュータブルクラスは Builder クラスを入力として受け取る @DynamoDbImmutable アノテーションを使用します。

次の例では、DynamoDbEnhancedClientImmutablePutItem Builder クラスを入力としてテーブルスキーマを作成します。この例では次に、スキーマを PutItem API コールの入力として使用します。

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutablePutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutablePutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableItem.class)
                .item(YourImmutableItem.builder()
                                        .pk("123")
                                        .sk("cart#123")
                                        .itemData("YourItemData")
                                        .inventory(500)
                                        .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

次の例は、イミュータブルデータクラスを示しています。

```
@DynamoDbImmutable(builder = YourImmutableItem.YourImmutableItemBuilder.class)
class YourImmutableItem {
    private final String pk;
    private final String sk;
    private final String itemData;
    private final int inventory;
    public YourImmutableItem(YourImmutableItemBuilder builder) {
        this.pk = builder.pk;
        this.sk = builder.sk;
        this.itemData = builder.itemData;
        this.inventory = builder.inventory;
    }
    public static YourImmutableItemBuilder builder() { return new
 YourImmutableItemBuilder(); }

    @DynamoDbPartitionKey
    public String getPk() {
        return pk;
    }

    @DynamoDbSortKey
    public String getSk() {
        return sk;
    }

    public String getItemData() {
        return itemData;
    }

    public int getInventory() {
        return inventory;
    }

    static final class YourImmutableItemBuilder {
        private String pk;
        private String sk;
        private String itemData;
        private int inventory;

        private YourImmutableItemBuilder() {}

        public YourImmutableItemBuilder pk(String pk) { this.pk = pk; return this; }
        public YourImmutableItemBuilder sk(String sk) { this.sk = sk; return this; }
        public YourImmutableItemBuilder itemData(String itemData) { this.itemData =
 itemData; return this; }
        public YourImmutableItemBuilder inventory(int inventory) { this.inventory =
 inventory; return this; }

        public YourImmutableItem build() { return new YourImmutableItem(this); }
    }
}
```

イミュータブルデータクラスとサードパーティーのボイラープレート生成ライブラリを使用する高レベルインターフェイス

前のセクションで説明したイミュータブルデータクラスの例には、ボイラープレートコードが必要です。例えば、Builder クラス以外にも、データクラスには getter ロジックと setter ロジックがあります。Project Lombok などのサードパーティーライブラリは、このようなタイプのボイラープレートコードの生成に役立ちます。

AWS SDK for Java 2.x ドキュメントの「Lombok などのサードパーティーライブラリを使用する」セクションでは、Project Lombok ライブラリを活用する概念を紹介しています。ボイラープレートコードの大部分を低減すると、イミュータブルデータクラスと AWS SDK の使用に必要なコードの量を制限できます。これにより、コードの生産性と可読性がさらに向上します。

次の例は、Project Lombok が DynamoDB 拡張クライアント API を使用するために必要なコードをどのように簡略化できるかを示しています。

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutableLombokPutItem {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableLombokItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableLombokItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutableLombokPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableLombokItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableLombokItem.clas
                .item(YourImmutableLombokItem.builder()
                        .pk("123")
                        .sk("cart#123")
                        .itemData("YourItemData")
                        .inventory(500)
                        .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

次の例は、イミュータブルデータクラスのイミュータブルデータオブジェクトを示しています。

```
import lombok.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;

@Builder
@DynamoDbImmutable(builder =
 YourImmutableLombokItem.YourImmutableLombokItemBuilder.class)
@Value
public class YourImmutableLombokItem {

    @Getter(onMethod_=@DynamoDbPartitionKey)
    String pk;
    @Getter(onMethod_=@DynamoDbSortKey)
    String sk;
    String itemData;
    int inventory;
}
```

YourImmutableLombokItem クラスは Project Lombok と AWS SDK が提供する以下のアノテーションを使用します。

- @Builder — Project Lombok が提供するデータクラス用の複雑な Builder API を生成します。
- @DynamoDbImmutable — DynamoDbImmutable クラスを、AWS SDK によって提供される DynamoDB マッピング可能なエンティティのアノテーションとして識別します。
- @Value — @Data のイミュータブル Variant。すべてのフィールドはデフォルトでプライベートかつ最終的なものになり、setters は生成されません。このアノテーションは Project Lombok が提供します。

Document インターフェイス

Document インターフェイスでは、データ型記述子を指定する必要がありません。データ型は、データ自体のセマンティクスによって暗示されています。AWS SDK for Java 1.x の Document インターフェイスを使い慣れている場合は、AWS SDK for Java 2.x の Document インターフェイスも同様の、ただし再設計されたインターフェイスを提供します。

次の Document interface example は、この Document インターフェイスを使用した PutItem コールの表現を示しています。この例では EnhancedDocument も使用しています。Enhanced Document API を使用して DynamoDB テーブルに対してコマンドを実行するには、まずテーブルを Document テーブルスキーマに関連付けて、DynamoDBTable リソースオブジェクトを作成する必要があります。Document テーブルスキーマの Builder には、プライマリインデックスキーと属性コンバーターのプロバイダーが必要です。

デフォルトタイプの Document 属性の変換には AttributeConverterProvider.defaultProvider() を使用できます。全体的なデフォルト動作は、カスタム AttributeConverterProvider 実装で変更できます。また、1 つの属性のコンバーターを変更することもできます。「AWS SDK ドキュメント」には、カスタムコンバーターの使用方法の詳細と例が提供されています。コンバーターは主に、デフォルトコンバーターがないドメインクラスの属性に使用します。カスタムコンバータを使用すると、DynamoDB への書き込みまたは読み取りに必要な情報を SDK に提供できます。

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedDocumentClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()

 .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk", AttributeValueType.S)
                            .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                            .build());

    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<EnhancedDocument> response =
 DYNAMODB_TABLE.putItemWithResponse(
                        PutItemEnhancedRequest.builder(EnhancedDocument.class)
```

```
                                 .item(
                                     EnhancedDocument.builder()

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                                             .putString("pk", "123")
                                             .putString("sk", "cart#123")
                                             .putString("item_data", "YourItemData")
                                             .putNumber("inventory", 500)
                                             .build())
                                 .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                                 .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

}
```

次のユーティリティメソッドを使用して、JSON ドキュメントとネイティブの Amazon DynamoDB のデータ型との間の変換ができます。

- `EnhancedDocument.fromJson(String json)`— JSON 文字列から新しい EnhancedDocument インスタンスを作成します。
- `EnhancedDocument.toJson()`— Document の JSON 文字列表現を作成して、その他の JSON オブジェクトと同様にアプリケーションで使用できるようにします。

Query 例を使用したインターフェイスの比較

このセクションでは、さまざまなインターフェイスを使って、同じ Query コールの表現を説明します。以下のクエリでは、いくつかの属性を使用して次のとおりクエリ結果を微調整します。

- DynamoDB では、単一の特定のパーティションキー値をターゲットにするため、パーティションキーは完全に指定する必要があります。
- ソートキーには `begins_with` を使用したキー条件式があるため、カートの商品のみがこのクエリの対象となります。
- `limit()` を使用して、クエリが最大 100 の商品を返すように制限します。
- `scanIndexForward` は false に設定します。結果は UTF-8 バイトの順序で返されます。つまり通常、番号が最も小さいカートの商品が最初に返されます。`scanIndexForward` を false に設定すると、この順序が逆になり、番号が最も大きいカートの商品が最初に返されます。

- フィルターを適用して、条件に一致しない結果をすべて削除します。商品がフィルターに一致するかどうかにかかわらず、フィルターが適用されるデータは読み取りキャパシティを消費します。

Example 低レベルインターフェイスを使用するクエリ

次の例では、クエリが特定のプレフィックス値で始まる特定のパーティションキー値とソートキー値に制限する `keyConditionExpression` を使用して `YourTableName` という名前のテーブルをクエリします。このようなキーの条件は、DynamoDB から読み取られるデータ量を制限します。このクエリは最後に、`filterExpression` を使用して DynamoDB から取得したデータにフィルターを適用します。

```
import org.slf4j.*;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class Query {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT = 
 DynamoDbClient.builder().build();
    private static final Logger LOGGER = LoggerFactory.getLogger(Query.class);

    private static void query() {
        QueryResponse response = DYNAMODB_CLIENT.query(QueryRequest.builder()
                .expressionAttributeNames(Map.of("#name", "name"))
                .expressionAttributeValues(Map.of(
                    ":pk_val", AttributeValue.fromS("id#1"),
                    ":sk_val", AttributeValue.fromS("cart#"),
                    ":name_val", AttributeValue.fromS("SomeName")))
                .filterExpression("#name = :name_val")
                .keyConditionExpression("pk = :pk_val AND begins_with(sk, :sk_val)")
                .limit(100)
                .scanIndexForward(false)
                .tableName("YourTableName")
                .build());

        LOGGER.info("nr of items: " + response.count());
        LOGGER.info("First item pk: " + response.items().get(0).get("pk"));
        LOGGER.info("First item sk: " + response.items().get(0).get("sk"));
    }
}
```

Example Document インターフェイスを使用するクエリ

次の例では、Document インターフェイスを使用して、`YourTableName` という名前のテーブルをクエリします。

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;

import java.util.Map;

public class DynamoDbEnhancedDocumentClientQuery {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()
                    .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk",
 AttributeValueType.S)
                    .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                    .build());
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientQuery.class);

    private void query() {
        PageIterable<EnhancedDocument> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
```

```
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " +
 response.items().iterator().next().getString("pk"));
        LOGGER.info("First item sk: " +
 response.items().iterator().next().getString("sk"));

    }
}
```

Example 高レベルインターフェイスを使用するクエリ

次の例では、DynamoDB Enhanced Client API を使用して、`YourTableName` という名前のテーブルをクエリします。

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import java.util.Map;

public class DynamoDbEnhancedClientQuery {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromBean(DynamoDbEnhancedClientQuery.YourItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientQuery.class);

    private void query() {
        PageIterable<YourItem> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " + response.items().iterator().next().getPk());
        LOGGER.info("First item sk: " + response.items().iterator().next().getSk());
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String name) {
            this.pk = pk;
            this.sk = sk;
            this.name = name;
        }

        private String pk;
        private String sk;
        private String name;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }
```

```
        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setName(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }
    }
}
```

イミュータブルデータクラスを使用する高レベルインターフェイス

高レベルのイミュータブルデータクラスを使用して Query を実行する場合、`YourItem` または `YourImmutableItem` エンティティクラスの構築を除き、コードは高レベルインターフェイスの例と同じです。詳細については、「PutItem」の例を参照してください。

イミュータブルデータクラスとサードパーティーのボイラープレート生成ライブラリを使用する高レベルインターフェイス

高レベルのイミュータブルデータクラスを使用して Query を実行する場合、`YourItem` または `YourImmutableLombokItem` エンティティクラスの構築を除き、コードは高レベルインターフェイスの例と同じです。詳細については、「PutItem」の例を参照してください。

その他のコード例

SDK for Java 2.x で DynamoDB を使用する場合のその他の例が記載されている以下のコードサンプルリポジトリを参照することもできます。

- AWS 公式シングルアクションのコード例
- コミュニティが管理するシングルアクションのコード例。
- AWS 公式のシナリオ指向コード例

同期プログラミングと非同期プログラミング

AWS SDK for Java 2.x は、DynamoDB などの AWS のサービス に同期クライアントと非同期クライアントの両方を提供します。

`DynamoDbClient` クラスと `DynamoDbEnhancedClient` クラスは、クライアントがサービスから応答を受信するまでスレッドの実行をブロックする同期メソッドを提供します。非同期のオペレーションが必要ない場合、このクライアントは DynamoDB を操作する最も簡単な方法です。

`DynamoDbAsyncClient` クラスと `DynamoDbEnhancedAsyncClient` クラスは、直ちに呼び出し元に戻り、応答を待たずに呼び出し元のスレッドに制御を返す非同期メソッドを提供します。ノンブロッキングクライアントの場合、少数のスレッド間で優れた同時実行が可能で、最小限のコンピューティングリソースで I/O リクエストを効率的に処理できるという利点があります。これにより、スループットと即応性が向上します。

AWS SDK for Java 2.x はノンブロッキング I/O のネイティブサポートを使用しています。AWS SDK for Java 1.x ではノンブロッキング I/O をシミュレートする必要がありました。

非同期メソッドは応答が利用できるようになる前に呼び出し元に戻るため、応答の準備ができたら応答を取得する手段を講じる必要があります。AWS SDK for Java の非同期メソッドは、その後の非同期オペレーションの結果を含む `CompletableFuture` オブジェクトを返します。このような `CompletableFuture` オブジェクトに対して `get()` または `join()` を呼び出すと、結果が利用できるようになるまでコードはブロックされます。リクエストと同時にこの呼び出しを行う場合、動作は通常の同期呼び出しと同様になります。

この非同期スタイルを活用する方法の詳細については、「非同期プログラミング」に関する AWS SDK ドキュメントに記載されています。

HTTP クライアント

すべてのクライアントをサポートするための AWS のサービス との通信を処理する HTTP クライアントがあります。アプリケーションに最も適切な特性を持つ HTTP クライアントを選択して、代替 HTTP クライアントをプラグインできます。比較的軽量なクライアントも、設定オプションが豊富にあるクライアントもあります。

HTTP クライアントによっては、同期使用のみをサポートするものもあれば、非同期使用のみをサポートするものもあります。AWS SDK ドキュメントには、ワークロードに最適な HTTP クライアントを選択するのに役立つフローチャートが記載されています。

利用できる HTTP クライアントのいくつかを次の一覧にまとめています。

トピック

- Apache ベースの HTTP クライアント
- URLConnection ベースの HTTP クライアント
- Netty ベースの HTTP クライアント
- AWS CRT ベースの HTTP クライアント

Apache ベースの HTTP クライアント

ApacheHttpClient は、同期サービスクライアントをサポートしています。これは同期的な使用の場合のデフォルトの HTTP クライアントです。`ApacheHttpClient` の設定については、AWS SDK for Java 2.x ドキュメントの「Apache ベースの HTTP クライアントを設定する」を参照してください。

URLConnection ベースの HTTP クライアント

URLConnectionHttpClient は同期クライアント向けのもう 1 つのオプションです。Apache ベースの HTTP クライアントよりもロード時間がかからないとはいえ、機能は限られています。UrlConnectionHttpClient の設定については、「URLConnection ベースの HTTP クライアントを設定する」を参照してください。

Netty ベースの HTTP クライアント

`NettyNioAsyncHttpClient` は非同期クライアントをサポートしており、非同期を使用する場合のデフォルトのオプションです。`NettyNioAsyncHttpClient` の設定については、「Netty ベースの HTTP クライアントを設定する」を参照してください。

AWS CRT ベースの HTTP クライアント

同期クライアントと非同期クライアントの両方をサポートするもう 1 つのオプションには、AWS Common Runtime (CRT) ライブラリに新たに追加された `AwsCrtHttpClient` と `AwsCrtAsyncHttpClient` があります。他の HTTP クライアントと比較して、AWS CRT は以下を提供します。

- SDK 起動時間の短縮
- より小さなメモリフットプリント
- レイテンシータイムの短縮
- 接続のヘルス管理

- DNS ロードバランサー

AwsCrtHttpClient と AwsCrtAsyncHttpClient の設定については、「AWS CRT ベースの HTTP クライアントを設定する」を参照してください。

AWS CRT ベースの HTTP クライアントは、既存のアプリケーションとの下位互換性が失われるため、デフォルトではありません。ただし、DynamoDB の場合は、同期と非同期の両方に AWS CRT ベースの HTTP クライアントを使用することをお勧めします。

AWS CRT ベースの HTTP クライアントの概要については、「AWS SDK for Java 2.x の AWS CRT HTTP クライアントの一般提供を開始」を参照してください。

HTTP クライアントの設定

クライアントを設定する際には、次のとおりのさまざまな設定オプションを指定できます。

- API コールのさまざまな側面でのタイムアウトの設定
- TCP キープアライブを有効にするかどうかの制御
- エラー発生時の再試行ポリシーの制御
- 実行インターセプターインスタンスが変更できる実行属性の指定。実行インターセプターは、API リクエストと応答の実行をインターセプトするコードを作成できます。これにより、メトリクスの公開やリクエストの変更などのタスクを進行中に実行できます。
- HTTP ヘッダーの追加や操作
- クライアント側のパフォーマンスメトリクスの追跡を有効にできます。この機能を使用すると、アプリケーション内のサービスクライアントに関するメトリクスを収集し、Amazon CloudWatch で出力を分析できます。
- 非同期の再試行やタイムアウトタスクなどのタスクのスケジュール設定に使用する代替エグゼキュータサービスを指定できます。

設定を制御するには、ClientOverrideConfiguration オブジェクトをサービスクライアントの Builder クラスに提供します。これについては、次のセクションのコードサンプルのいくつかで説明します。

ClientOverrideConfiguration では、標準的な設定のオプションが提供されます。プラガブル HTTP クライアントによっては、実装に応じた設定もできます。このようなクライアントについてはそれぞれ独自のドキュメントも提供されています。

- Apache ベースの HTTP クライアント
- URLConnection ベースの HTTP クライアント
- Netty ベースの HTTP クライアント
- AWS CRT ベースの HTTP クライアント

このセクションのトピック

タイムアウト設定

クライアントの設定を調整して、サービスの呼びだしに関連するさまざまなタイムアウトを制御できます。DynamoDB は、その他の AWS のサービス サービスと比較してレイテンシーが低減します。このため、ネットワークに問題が発生した場合のフェイルファストを実現するように、プロパティを調整してタイムアウト値を低く指定することをお勧めします。

DynamoDB クライアントで `ClientOverrideConfiguration` を使用するか、基盤となる HTTP クライアント実装で詳細な設定オプションを変更することによって、レイテンシーに関する動作をカスタマイズできます。

以下の影響を左右するプロパティを設定するには、`ClientOverrideConfiguration` を使用できます。

- `apiCallAttemptTimeout` — HTTP リクエストの 1 回の試行が完了するまで、諦めてタイムアウトになるまで待機する時間。
- `apiCallTimeout` — クライアントが API コールの実行を完了するのに許可される時間。これには、再試行を含むすべての HTTP リクエストで設定されるリクエストハンドラーの実行が含まれます。

接続タイムアウトやソケットタイムアウトなど、タイムアウトオプションによっては、AWS SDK for Java 2.x は、デフォルト値を提供します。SDK は、API コールのタイムアウトや個別の API コールの試行タイムアウトのデフォルト値は提供していません。このようなタイムアウトが `ClientOverrideConfiguration` で設定されていない場合、SDK は全体的な API コールタイムア

ウトの代わりにソケットのタイムアウト値を使用します。ソケットのタイムアウトのデフォルト値は 30 秒です。

RetryMode

タイムアウト設定に関して考慮すべきもう 1 つの設定は、`RetryMode` 設定オブジェクトです。この設定オブジェクトには、再試行動作のコレクションが含まれています。

SDK for Java 2.x は、次の再試行モードをサポートしています。

- `legacy` — 明示的に変更しない場合のデフォルトの再試行モード。この再試行モードは Java SDK 固有のもので、次の特徴があります。
  - 最大 3 回まで再試行できます。DynamoDB などのサービスの場合は 8 回まで再試行できます。
- `standard` — 他の AWS SDK との整合性が高いことから、standard と名付けられています。このモードでは、最初の再試行で 0 ミリ秒から 1,000 ミリ秒の範囲のランダムな期間だけ待機します。もう一度再試行が必要な場合は、0 ミリ秒から 1,000 ミリ秒までの別のランダムな期間を選択して、これを 2 倍します。さらに再試行が必要な場合は、同じようにランダムな時間を選択し、これに 4 を掛け、試行を続けます。各待機時間の上限は 20 秒です。このモードは、検出された障害条件の数が `legacy` モードよりも多い場合に再試行を実行します。DynamoDB の場合、numRetries でオーバーライドしない限り、最大合計 3 回の試行が実行されます。
- `adaptive` — `standard` モードに基づいて構築されています。成功率を最大化するために AWS リクエスト率を動的に制限します。これにより、リクエストのレイテンシーが増大する場合があります。予測可能なレイテンシーが重要な場合は、アダプティブ再試行モードはお勧めしません。

このような再試行モードの詳細な定義については、「AWS SDK とツールのリファレンスガイド」の「再試行動作」トピックを参照してください。

再試行ポリシー

すべての RetryMode 設定には、単一または複数の RetryCondition 設定に基づいて構築される RetryPolicy があります。TokenBucketRetryCondition は、DynamoDB SDK クライアント実装の再試行動作には、特に重要です。この条件は、トークンバケットアルゴリズムを使用して SDK が行う再試行の数を制限します。選択した再試行モードに応じて、スロットリング例外によって `TokenBucket` からトークンが減算される場合と減算されない場合があります。

クライアントでスロットリング例外や一時的なサーバーエラーなどの再試行できるエラーが発生すると、SDK は自動的にリクエストを再試行します。このような再試行の回数と頻度は制御できます。

クライアントを設定する際、次のパラメータをサポートする RetryPolicy を指定できます。

- numRetries—リクエストが失敗したとみなされるまでに適用すべき最大再試行回数。使用する再試行モードを問わず、デフォルト値は 8 です。

> ⚠ Warning
>
> このデフォルト値は、慎重に考慮した上で変更してください。

- backoffStrategy— 再試行に適用する BackoffStrategy。デフォルトの戦略は、FullJitterBackoffStrategy です。この戦略は、現在の再試行回数、ベースとなる遅延、最大バックオフ時間に基づいて、追加の再試行の間に指数関数的な遅延を実行します。次に、ジッターを追加して、多少のランダム性を提供します。指数関数的遅延で使用される基本遅延は、再試行モードを問わず 25 ミリ秒です。
- retryCondition— RetryCondition は、リクエストを再試行すべきかどうかを決定します。デフォルトでは、再試行可能と思われる特定の HTTP ステータスコードと例外のセットを再試行します。ほとんどの場合、このデフォルトの設定で十分です。

次のコードは代替再試行ポリシーを提供します。合計で 5 回の再試行 (合計 6 回のリクエスト) を許可するように指定しています。最初の再試行は約 100 ミリ秒の遅延の後に実行され、追加の再試行ごとにその時間が指数関数的に倍増され、最大 1 秒の遅延が発生します。

```
DynamoDbClient client = DynamoDbClient.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .retryPolicy(RetryPolicy.builder()
            .backoffStrategy(FullJitterBackoffStrategy.builder()
                .baseDelay(Duration.ofMillis(100))
                .maxBackoffTime(Duration.ofSeconds(1))
                .build())
            .numRetries(5)
            .build())
        .build())
    .build();
```

DefaultsMode

ClientOverrideConfiguration と RetryMode が管理しないタイムアウトプロパティは、通常、明示的に設定されません。この代わりに、DefaultsMode を指定することによって暗黙的に設定されます。

DefaultsMode のサポートは AWS SDK for Java 2.x (バージョン 2.17.102 以降) で導入され、HTTP 通信設定、再試行動作、サービスのリージョンエンドポイント設定などの一般的な設定可能な設定のデフォルト値のセットが提供されています。この機能を使用すると、一般的な使用シナリオに合わせて新しい設定のデフォルトを指定できます。

デフォルトモードはすべての AWS SDK で標準化されています。SDK for Java 2.x は、次のデフォルトモードをサポートしています。

- legacy — SDK によって異なり、DefaultsMode が確立される前に存在していたデフォルト設定を使用します。
- standard — ほとんどのシナリオで、最適化されていないデフォルト設定を提供します。
- in-region — 標準モードに基づいて構築されており、同じ AWS リージョン 内から AWS のサービス を呼び出すアプリケーションにカスタマイズされた最適化が含まれています。
- cross-region — 標準モードに基づいて構築されており、異なる AWS リージョン の AWS のサービス を呼び出すアプリケーション向けにタイムアウトを長く指定した設定が含まれています。
- mobile — 標準モードに基づいて構築されており、レイテンシーの長いモバイルアプリケーション向けにカスタマイズした、タイムアウト時間を長く指定した設定が含まれています。
- auto — 標準モードに基づいて構築されており、実験的な機能が含まれています。SDK はランタイム環境を検出して適切な設定の自動決定を試行します。自動検出はヒューリスティックベースで、100% の精度が得られるとは限りません。ランタイム環境が特定できない場合は、標準モードを使用します。自動検出を行うと、インスタンスのメタデータとユーザデータがクエリされる可能性があり、これによりレイテンシーが発生する場合があります。起動時のレイテンシーがアプリケーションにとって最重要な場合は、代わりに明示的な DefaultsMode を選択することをおすすめします。

デフォルトモードは次の方法で設定できます。

- AwsClientBuilder.Builder#defaultsMode(DefaultsMode) を介して直接クライアントで
- defaults_mode プロファイルファイルプロパティを使用して設定プロファイルで
- aws.defaultsMode システムプロパティを使用してグローバルに
- AWS_DEFAULTS_MODE 環境変数を使用してグローバルに

> Note
>
> legacy 以外のモードでは、ベストプラクティスの進化に応じて、提供されるデフォルト値が変更される可能性があります。このため、legacy 以外のモードを使用している場合は、SDK をアップグレードする際にテストを実施することをお勧めします。

「AWS SDK とツールのリファレンスガイド」の「スマート設定のデフォルト」には、さまざまなデフォルトモードでの設定プロパティとデフォルト値の一覧が記載されています。

デフォルトのモード値は、アプリケーションの特性と、アプリケーションが操作する AWS のサービス に基づいて選択します。

デフォルト値は、幅広い AWS のサービス が選択されることを考慮して設定されています。DynamoDB テーブルとアプリケーションの両方が単一のリージョンにデプロイされる典型的な DynamoDB デプロイの場合、standard デフォルトモードの中で最も関連性があるのは in-region のデフォルトモードです。

Example 低レイテンシーの呼び出し向けに調整した DynamoDB SDK クライアントの設定

次の例では、低レイテンシーが期待される DynamoDB 呼び出し向けにタイムアウトをより低い値に調整します。

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

個別の HTTP クライアント実装により、タイムアウトと接続使用の動作はさらに詳細に制御できます。例えば、AWS CRT ベースのクライアントの場合、ConnectionHealthConfiguration を有効にすると、AWS CRT ベースのクライアントが使用される接続のヘルスをアクティブにモニタリングできます。詳細については、AWS SDK for Java 2.x の ドキュメントを参照してください。

キープアライブの設定

キープアライブを有効にすると、接続を再利用することでレイテンシーを短縮できます。キープアライブには、HTTP キープアライブと TCP キープアライブの 2 種類があります。

- HTTP キープアライブは、その後のリクエストで接続を再利用できるように、クライアントとサーバー間の HTTPS 接続の維持を試行します。これにより、重い HTTPS 認証がその後のリクエストでスキップされます。HTTP キープアライブはすべてのクライアントでデフォルトで有効になっています。
- TCP キープアライブは、基盤となるオペレーティングシステムに対して、ソケット接続を介して小型のパケットを送信するようにリクエストします。これにより、ソケットが維持されることがさらに保証され、ドロップを即座に検出できます。これにより、ドロップしたソケットの使用にその後のリクエストが時間を費やす必要がなくなります。TCP キープアライブは、デフォルトではすべてのクライアントで無効になっています。これは、有効にできます。次のコードサンプルは、各 HTTP クライアントでこれを実行する方法を示しています。CRT ベース以外のすべての HTTP クライアントの場合、キープアライブを有効にすると、実際のキープアライブメカニズムはオペレーティングシステムに依存します。このため、タイムアウトやパケット数など、追加の TCP キープアライブ値をオペレーティングシステムで設定する必要があります。これを実行するには、Linux または Mac マシンでは `sysctl` を使用し、Windows マシンではレジストリ値を使用します。

Example Apache ベースの HTTP クライアントで TCP キープアライブを有効にするには

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

URLConnection ベースの HTTP クライアント

URLConnection ベースの HTTP クライアント `HttpURLConnection` を使用する同期クライアントには、キープアライブを有効にするメカニズムはありません。

Example Netty ベースの HTTP クライアントで TCP キープアライブを有効にするには

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example AWS CRT ベースの HTTP クライアントで TCP キープアライブを有効にするには

AWS CRT ベースの HTTP クライアントを使用すると、TCP キープアライブを有効にして、実行時間を制御できます。

```
DynamoDbClient client = DynamoDbClient.builder()
```

```
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

非同期の DynamoDB クライアントを使用する場合は、次のコードのとおり TCP キープアライブを有効にすることができます。

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

エラー処理

例外処理については、AWS SDK for Java 2.x ではランタイム (非チェック) 例外を使用します。

すべての SDK 例外をカバーする基本となる例外は、Java 非チェック `RuntimeException` の拡張である `SdkServiceException` です。これを使用してキャッチすると、SDK からスローされるすべての例外をキャッチできます。

`SdkServiceException` には、`AwsServiceException` というサブクラスがあります。このサブクラスは、AWS のサービス との間の通信に問題があることを示します。これには、DynamoDB との通信の問題を示す `DynamoDbException` というサブクラスがあります。これを使用してキャッチすると、DynamoDB に関連するすべての例外はキャッチされますが、他の SDK 例外はキャッチされません。

`DynamoDbException` には、さらに具体的な例外タイプもあります。このような例外タイプの中には、`TableAlreadyExistsException` などのコントロールプレーンの操作に適用されるものもあります。データプレーン操作に適用されるものもあります。一般的なデータプレーン例外の例は、以下のとおりです。

- `ConditionalCheckFailedException`—リクエストに false と評価される条件を指定しました。たとえば、項目に条件付き更新を実行しようとしたかもしれませんが、属性の実際の値は、条件の予期される値と一致しませんでした。この方法で失敗したリクエストは再試行されません。

その他の状況については、特定の例外は定義されていません。例えば、リクエストがスロットリングすると、特定の ProvisionedThroughputExceededException がスローされ、その他の場合にはより一般的な DynamoDbException がスローされる可能性があります。いずれの場合も、isThrottlingException() が true を返したかを確認することで、例外がスロットリングによって引き起こされたかどうかを判断できます。

アプリケーションのニーズに応じて、すべての AwsServiceException インスタンスまたは DynamoDbException インスタンスをキャッチできます。ただし多くの場合、状況に応じて異なる動作が必要になります。条件チェックの失敗に対処するロジックは、スロットリングに対処する場合とは異なります。どの例外パスに対処するかを定義して、代替パスは必ずテストしてください。テストの実施は、関連するすべてのシナリオに確実に対処できるようにするうえで役立ちます。

「DynamoDB でのエラー処理」には、発生する可能性のある一般的なエラーがいくつか記載されています。一般的なエラーのリストについては、「一般的なエラー」でも参照できます。特定の API コールについて、発生する可能性のある正確なエラーについては、「Query」API などのドキュメントにも記載されています。例外処理の詳細については、「AWS SDK for Java 2.x の例外処理」を参照してください。

AWS リクエスト ID

各リクエストにはリクエスト ID が含まれており、AWS Support と連携して問題を診断する場合に取得すると便利です。SdkServiceException から派生した各例外には、リクエスト ID を取得できる requestId() メソッドがあります。

ログ記録

SDK が提供するログ記録を使用すると、クライアントライブラリから重要なメッセージをキャッチする場合や、より詳細なデバッグを行う場合の両方で役に立ちます。ロガーは階層化されており、SDK はルートロガーとして software.amazon.awssdk を使用します。レベルは、TRACE、DEBUG、INFO、WARN、ERROR、ALL、または OFF のいずれかを使用して設定できます。設定したレベルは、そのロガーとロガー階層の下位レベルに適用されます。

AWS SDK for Java 2.x は、ログ記録に Simple Logging Façade for Java (SLF4J) を利用しており、これはその他のロガーの周囲の抽象化レイヤーとして機能します。これにより、選択したロガーをプラグインできます。ロガーの接続方法については、「SLF4J ユーザーマニュアル」を参照してください。

各ロガーには特定の動作があります。Log4j 2.x ロガーは、デフォルトで、ログイベントを System.out に追加する ConsoleAppender を作成し、デフォルトで ERROR ログレベルを設定します。

SLF4J に含まれる SimpleLogger ロガーは、デフォルトで System.err を出力し、ログレベルはデフォルトで INFO になります。

出力量を制限しながら、SDK のクライアントライブラリから重要なメッセージをキャッチできるように、本番環境のデプロイでは software.amazon.awssdk のレベルを WARN に設定することをお勧めします。

SLF4J がクラスパス上でサポートされているロガーを見つけられない場合 (SLF4J バインディングがない場合)、SLF4J はデフォルトで no operation implementation になります。この実装により、SLF4J がクラスパス上でロガー実装を見つけられなかったことを説明するメッセージが System.err に記録されます。このような状況を防ぐには、ロガー実装を追加する必要があります。これを実行するには、Apache Maven の pom.xml に org.slf4j.slf4j-simple や org.apache.logging.log4j.log4j-slf4j2-imp などのアーティファクトに対する依存関係を追加します。

AWS SDK for Java 2.x のドキュメントでは、アプリケーション設定へのログの依存関係の追加など、SDK でのログの設定方法について説明しています。詳細については、「SDK for Java 2.x でのログ記録」を参照してください。

Log4j2.xml ファイルの次の設定では、Apache Log4j 2 ロガーを使用している場合にログ記録の動作を調整する方法を示しています。この設定では、ルートロガーレベルは WARN に設定しています。このログレベルは、software.amazon.awssdk ロガーなど、階層内のすべてのロガーに継承されます。

デフォルトでは、出力は System.out に送信されます。次の例では、デフォルトの出力 Log4j アペンダを上書きして、カスタマイズした Log4j PatternLayout を適用します。

Log4j2.xml 設定ファイルの例

次の設定では、すべてのロガー階層のコンソールに ERROR レベルと WARN レベルのメッセージがログ記録されます。

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

AWS リクエスト ID のログ記録

何らかの問題が発生した場合、例外内で RequestId を検索できます。ただし、例外を生成していないリクエストの RequestID が必要な場合は、ログ記録を使用できます。

リクエスト ID は、`software.amazon.awssdk.request` ロガーによって DEBUG レベルで出力されます。次の例は、前の configuration example を拡張して、ルートロガーのレベルを ERROR、`software.amazon.awssdk` のレベルを WARN、`software.amazon.awssdk.request` のレベルを DEBUG に維持します。このようなレベルを設定すると、リクエスト ID や、エンドポイント、ステータスコードなどのその他のリクエストに関する詳細を取得するうえで役立ちます。

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="ERROR">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

ログ出力の例を次に示します。

```
2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Sending Request:
 DefaultSdkHttpFullRequest(httpMethod=POST, protocol=https, host=dynamodb.us-
east-1.amazonaws.com, encodedPath=/, headers=[amz-sdk-invocation-id, Content-Length,
 Content-Type, User-Agent, X-Amz-Target], queryParameters=[])
```

```
2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Received
successful response: 200, Request ID:
QS9DUMME2NHEDH8TGT9N5V53OJVV4KQNSO5AEMVJF66Q9ASUAAJG, Extended Request ID: not
available
```

ページ分割

Query や Scan など、リクエストによっては、1 回のリクエストで返されるデータのサイズが制限され、後続のページを取得するにはリクエストを繰り返し行う必要がある場合があります。

各ページで読み取られる項目の最大数は、Limit パラメータを使用して制御できます。Limit パラメータを使用すると、例えば、最後の 10 項目のみを取得することができます。このような制限は、フィルタリングが適用される前にテーブルから読み込まれる項目の数の上限となります。フィルタリング後に正確に 10 項目が必要であることを指定する方法はありません。事前にフィルタリング済みの数を制御し、実際に 10 項目を取得した場合にのみクライアント側でチェックできます。制限にかかわらず、すべての応答の最大サイズは常に 1 MB です。

API 応答には、カウント制限またはサイズ制限に達したために応答が終了したことを示す LastEvaluatedKey が含まれる場合があります。このキーは、応答のために評価された最後のキーです。API を直接操作して、この LastEvaluatedKey を取得し、それを ExclusiveStartKey として後続の呼び出しに渡し、その開始点から次のチャンクを読み取ることができます。LastEvaluatedKey が返されない場合は、Query または Scan API コールに一致する項目がもうなかったことを意味します。

次の例では、低レベルインターフェイスを使用しており、keyConditionExpression パラメータに基づいて項目が 100 に制限されます。

```
QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
        .expressionAttributeValues(Map.of(
                ":pk_val", AttributeValue.fromS("123"),
                ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName(TABLE_NAME);

while (true) {
    QueryResponse queryResponse = DYNAMODB_CLIENT.query(queryRequestBuilder.build());

    queryResponse.items().forEach(item -> {
        LOGGER.info("item PK: [" + item.get("pk") + "] and SK: [" + item.get("sk") +
 "]");
```

```
    });

    if (!queryResponse.hasLastEvaluatedKey()) {
        break;
    }
    queryRequestBuilder.exclusiveStartKey(queryResponse.lastEvaluatedKey());
}
```

AWS SDK for Java 2.x では、複数のサービスコールを実行して、結果の次のページを自動的に取得する自動ページ分割メソッドが提供されており、DynamoDB の操作を簡素化されています。これによりコードが簡素化されます。ただし、ページを手動で読み取ることで維持していたリソース使用量がある程度制御できなくなります。

QueryPaginator や ScanPaginator など、DynamoDB クライアントで利用可能な `Iterable` メソッドを使用すると、SDK がページ分割を処理します。このようなメソッドの戻り値の型はカスタム iterable で、これを使用してすべてのページを反復処理できます。SDK は内部でサービスコールを自動的に処理します。Java Stream API を使用すると、次の例に示すとおり `QueryPaginator` の結果を処理できます。

```
QueryPublisher queryPublisher =
    DYNAMODB_CLIENT.queryPaginator(QueryRequest.builder()
        .expressionAttributeValues(Map.of(
            ":pk_val", AttributeValue.fromS("123"),
            ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName("YourTableName")
        .build());

queryPublisher.items().subscribe(item ->
    System.out.println(item.get("itemData"))).join();
```

データクラス注釈

Java SDK には、データクラスの属性に追加できるアノテーションがいくつか提供されています。このようなアノテーションは、SDK がデータクラスを処理する方法に影響を与えます。アノテーションを追加すると、属性を暗黙的なアトミックカウンタとして動作させたり、自動生成したタイムスタンプ値を維持したり、項目のバージョン番号を追跡したりできます。詳細については、「データクラスのアノテーション」を参照してください。

テーブル、項目、クエリ、スキャン、およびインデックスの使用

このセクションでは、Amazon DynamoDB でのテーブル、項目、クエリ、その他の操作に関する詳細を説明します。

トピック

DynamoDB でのテーブルとデータの操作

このセクションでは、AWS Command Line Interface (AWS CLI) および AWS SDK を使用して Amazon DynamoDB でテーブルを作成、更新、削除する方法を説明します。

> Note
>
> このタスクは AWS Management Console を使用して実行することもできます。詳細については、「コンソールを使用する場合」を参照してください。

このセクションでは、DynamoDB Auto Scaling や手動設定のプロビジョンされたスループットを使用し、スループット容量の詳細についても示します。

トピック

- リソースへのタグとラベルの追加
- Java での DynamoDB テーブルの使用
- .NET での DynamoDB テーブルの使用

DynamoDB テーブルの基本的なオペレーション

他のデータベース管理システムと同様、Amazon DynamoDB はデータをテーブルに保存します。いくつかの基本的なオペレーションでテーブルを管理できます。

トピック

テーブルの作成

`CreateTable` オペレーションを使用して、Amazon DynamoDB でテーブルを作成します。テーブルを作成するために、以下の情報を指定する必要があります。

- テーブル名。名前は DynamoDB 命名規則に従う必要があります。また、現在の AWS アカウントとリージョンで一意である必要があります。たとえば、米国東部 (バージニア北部) に `People` テーブルを作成し、欧州 (アイルランド) に別の `People` テーブルを作成できます。ただし、これらの 2 つのテーブルは互いにまったく異なっています。詳細については、「Amazon DynamoDB でサポートされるデータ型と命名規則」を参照してください。
- プライマリキー。プライマリキーは 1 つの属性 (パーティションキー) または 2 つの属性で構成できます (パーティションキーとソートキー)。属性名、データタイプ、各属性のロール (パーティションキーでは `HASH`、ソートキーでは `RANGE`) を指定する必要があります。詳細については、「プライマリキー」を参照してください。
- スループット設定 (プロビジョニングされたテーブルの場合)。プロビジョニングモードを使用している場合、最初の読み取りと書き込みのスループット設定をテーブルに指定する必要があります。

これらの設定は後から変更できます。また、DynamoDB Auto Scaling を有効化して設定を管理することもできます。詳細については、プロビジョンドキャパシティモードおよびDynamoDB Auto Scaling によるスループットキャパシティの自動管理を参照してください。

例 1: プロビジョニングされたテーブルを作成する

以下の AWS CLI サンプルは、テーブル (Music) の作成方法を示しています。プライマリキーは、Artist (パーティションキー) と SongTitle (ソートキー) で構成されており、データ型はそれぞれ String です。このテーブルの最大スループットは、読み取りキャパシティーユニット数が 10、書き込みキャパシティーユニット数が 5 です。

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

CreateTable オペレーションは、以下のようにテーブルにメタデータを返します。

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

TableStatus エレメントは、テーブルの現在の状態を示します（CREATING）。ReadCapacityUnits および WriteCapacityUnits に指定した値によって、テーブルの作成にはしばらく時間がかかる場合があります。これらの値を大きくすると、DynamoDB はより多くのリソースをテーブルに割り当てなければならなくなります。

例 2: オンデマンドテーブルを作成する

オンデマンドモードを使用して同じテーブル Music を作成するには

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode=PAY_PER_REQUEST
```

CreateTable オペレーションは、以下のようにテーブルにメタデータを返します。

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 0,
            "ReadCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        },
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397468.348
    }
}
```

> ⚠ Important
>
> オンデマンドテーブルで DescribeTable を呼び出すと、読み込みキャパシティユニットと書き込みキャパシティユニットが 0 に設定されます。

例 3: DynamoDB Standard-Infrequent Access テーブルクラスを使用してテーブルを作成する

DynamoDB Standard-Infrequent Access テーブルクラスを使用して同じ Music テーブルを作成するには

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

CreateTable オペレーションは、以下のようにテーブルにメタデータを返します。

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableClassSummary": {
            "LastUpdateDateTime": 1542397215.37,
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

表の説明

テーブルの詳細を表示するには、DescribeTable オペレーションを使用します。テーブル名を入力する必要があります。DescribeTable の出力は CreateTable の出力と同じ形式です。このメタデータには、テーブル作成時のタイムスタンプ、キースキーマ、プロビジョンドスループット設定、推定サイズ、既存のすべてのセカンダリインデックスが含まれています。

> **Important**
>
> オンデマンドテーブルで DescribeTable を呼び出すと、読み込みキャパシティーユニットと書き込みキャパシティーユニットが 0 に設定されます。

Example

```
aws dynamodb describe-table --table-name Music
```

TableStatus が CREATING から ACTIVE に変わると、テーブルは使用できる状態になります。

> **Note**
>
> DescribeTable のリクエスト直後に CreateTable リクエストを発行した場合、DynamoDB によってエラー (ResourceNotFoundException) が返されることがあります。DescribeTable で結果整合性のあるクエリが使用されており、テーブルの

メタデータがその時点で使用できない可能性があるためです。数秒間待ってから、再び DescribeTable リクエストを試してみてください。
請求目的上、DynamoDB ストレージコストには、100 バイトの項目あたりのオーバーヘッドが含まれます。(詳細は、DynamoDB 料金表を参照してください。) 各項目の余分な 100 バイトは、容量単位の計算または DescribeTable オペレーションでは使用されません。

テーブルの更新

UpdateTable オペレーションを使用すると、以下のいずれかを実行できます。

- テーブルのプロビジョニングされたスループット設定を変更します (プロビジョニングモードのテーブルの場合)。
- テーブルの読み取り/書き込みキャパシティーモードを変更します。
- テーブルでグローバルセカンダリインデックスを操作します (「DynamoDB のグローバルセカンダリインデックスの使用」を参照)。
- テーブルの DynamoDB Streams を有効または無効にできます (DynamoDB Streams の変更データキャプチャ を参照)。

Example

次の AWS CLI の例では、テーブルのプロビジョニングされたスループット設定を変更する方法を示します。

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

Note

UpdateTable リクエストを発行すると、テーブルのステータスが AVAILABLE から UPDATING に変わります。テーブルは UPDATING 中も全面的に利用できます。この処理が完了すると、テーブルのステータスが UPDATING から AVAILABLE に変わります。

Example

次の AWS CLI の例では、テーブルの読み取り/書き込みキャパシティーモードをオンデマンドモードに変更する方法を示します。

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

テーブルの削除

DeleteTable オペレーションで未使用のテーブルを削除できます。テーブルの削除は回復不可能な操作です。

Example

次の AWS CLI の例で、テーブルを削除する方法について説明します。

```
aws dynamodb delete-table --table-name Music
```

DeleteTable リクエストを発行すると、テーブルのステータスが ACTIVE から DELETING に変わります。使用しているリソース（テーブルに保存されたデータ、ストリーム、テーブルのインデックスなど）によって、テーブルの削除には時間がかかる場合があります。

DeleteTable オペレーションが完了すると、テーブルは DynamoDB に存在しなくなります。

削除保護の使用

削除保護プロパティを使用すると、テーブルを誤って削除しないように保護できます。テーブルに対してこのプロパティを有効にすると、管理者が通常のテーブル管理オペレーションを行うときにテーブルが誤って削除されるのを防ぐことができます。これにより、通常業務が中断されるのを防ぐことができます。

テーブル所有者または権限のある管理者が、各テーブルの削除保護プロパティを制御します。すべてのテーブルで削除保護プロパティは、デフォルトでオフに設定されています。これには、グローバルレプリカやバックアップから復元されたテーブルが含まれます。テーブルの削除保護が無効になっている場合、Identity and Access Management (IAM) ポリシーによって承認されたすべてのユーザーがそのテーブルを削除できます。テーブルの削除保護が有効になっているときは、誰も削除できません。

この設定を変更するには、表の[Additional settings] (追加設定) に移動し、[Deletion Protection] (削除保護)パネルに移動して、[Enable delete protection] (削除保護を有効にする) を選択します。

削除保護プロパティは、DynamoDB コンソール、API、CLI/SDK、および AWS CloudFormation でサポートされています。CreateTable API は、テーブル作成時の削除保護プロパティをサポートし

ており、UpdateTable API は既存のテーブルに対する削除保護プロパティの変更をサポートしています。

> Note
>
> - AWS アカウントを削除しても、テーブルを含むそのアカウントのすべてのデータは 90 日以内に削除されます。
> - DynamoDB は、テーブルを暗号化したカスタマーマネージドキーにアクセスできない場合、テーブルをアーカイブします。アーカイブには、テーブルのバックアップの作成と元のテーブルの削除が含まれます。

テーブル名のリスト化

ListTables オペレーションは、現在の AWS アカウントやリージョンの DynamoDB テーブルの名前を返します。

Example

次の AWS CLI の例は、DynamoDB テーブル名をリストする方法を示しています。

```
aws dynamodb list-tables
```

プロビジョニングされたスループットクォータの説明

DescribeLimits オペレーションは、現在の AWS アカウントやリージョンの現在の読み取りおよび書き込みキャパシティーのクォータを返します。

Example

次の AWS CLI の例では、現在のプロビジョニングされたスループットクォータを記述する方法を示します。

```
aws dynamodb describe-limits
```

出力は、現在の AWS アカウントやリージョンの読み取りおよび書き込みキャパシティーユニットの上限クォータを返します。

これらのクォータの詳細およびクォータの引き上げをリクエストする方法については、「スループットのデフォルトクォータ」を参照してください。

テーブルクラスを選択する場合の考慮事項

DynamoDB には、コストの最適化に役立つように設計された 2 つのテーブルクラスが用意されています。DynamoDB 標準テーブルクラスがデフォルトで、大半のワークロードで推奨されています。DynamoDB Standard-Infrequent Access (DynamoDB 標準-IA) テーブルクラスは、ストレージが主要なコストとなるテーブル用に最適化されています。例えば、アプリケーションログ、古いソーシャルメディアの投稿、E コマースの注文履歴、過去のゲーム実績など、アクセス頻度の低いデータを格納するテーブルは、標準-IA テーブルクラスの候補として適しています。

すべての DynamoDB テーブルは、テーブルクラスに関連付けられます。テーブルに関連付けられたすべてのセカンダリインデックスは、同じテーブルクラスを使用します。テーブル (デフォルトでは DynamoDB 標準) の作成時にテーブルクラスを設定できます。また、AWS Management Console、AWS CLI、または AWS SDK を使用して、既存のテーブルのテーブルクラスを更新できます。DynamoDB は、シングルリージョンテーブル（グローバルテーブルではないテーブル）に対して AWS CloudFormation を使用したテーブルクラスの管理もサポートしています。テーブルクラスごとに、データストレージ、読み取りおよび書き込みリクエストに対して異なる料金が設定されています。テーブルのテーブルクラスを選択する場合、次の点に注意してください。

- DynamoDB 標準テーブルクラスは、DynamoDB Standard-Infrequent Access (DynamoDB 標準 – IA) よりもスループットコストが低く、スループットが主要なコストであるテーブルで最もコスト効率の高いオプションです。
- DynamoDB Standard-Infrequent Access (DynamoDB 標準 – IA) テーブルクラスは、DynamoDB 標準よりも低いストレージコストを提供し、ストレージが主要なコストであるテーブルで最もコスト効率の高いオプションです。ストレージが DynamoDB 標準テーブルクラスを使用するテーブルのスループット (読み取りと書き込み) コストの 50% を超える場合、DynamoDB Standard-Infrequent Access (DynamoDB 標準 – IA) テーブルクラスを使用すると、テーブルの総コストを削減できます。
- DynamoDB 標準 IA テーブルは、DynamoDB 標準テーブルと同じパフォーマンス、耐久性、可用性を提供します。
- DynamoDB 標準テーブルクラスと DynamoDB 標準 — IA テーブルクラスを入れ替えても、アプリケーションコードを変更する必要はありません。テーブルで使用するテーブルクラスに関係なく、同じ DynamoDB API とサービスエンドポイントを使用します。
- DynamoDB 標準 IA テーブルは、Auto Scaling、オンデマンドモード、有効期限 (TTL)、オンデマンドバックアップ、ポイントインタイムリカバリ (PITR)、グローバルセカンダリインデックスなど、既存の DynamoDB に備わる機能のすべてと互換性があります。

テーブルにとって最も費用対効果の高いテーブルクラスは、そのテーブルの予想されるストレージとスループットの使用パターンによって異なります。テーブルのストレージとスループットのコストおよび使用状況の履歴は、AWS コストと使用状況レポート、および AWS Cost Explorer から確認できます。この履歴データを使用して、テーブルに対して最も費用対効果の高いテーブルクラスを特定します。AWS コストと使用状況レポート、および AWS Cost Explorer の使用の詳細については、AWS Billing and Cost Management のドキュメントを参照してください。テーブルクラスの料金の詳細については、Amazon DynamoDB Pricing (Amazon DynamoDB の料金表) を参照してください。

> Note
>
> テーブルクラスの更新はバックグラウンドで処理されます。テーブルクラスの更新中も、通常どおりテーブルにアクセスできます。テーブルクラスを更新する時間は、テーブルトラフィック、ストレージサイズ、およびその他の関連する変数によって異なります。30 日間の追跡期間では、最大 2 つのテーブルクラスの更新が可能です。

DynamoDB 項目のサイズと形式

DynamoDB テーブルはプライマリキーを除いてスキーマレスです。そのため、テーブルの項目の属性、サイズ、データ型はすべて異なる場合があります。

項目の合計サイズは、属性名と属性値の文字列の長さの合計、および以下に説明するように該当するオーバーヘッドが追加されます。次のガイドラインを使用して属性サイズを予測することができます。

- 文字列は、UTF-8 バイナリエンコードの Unicode です。文字列のサイズは、(属性名の UTF-8 でエンコードされたバイト数) + (UTF-8 でエンコードされたバイト数) です。
- 数値は、有効桁数が最大 38 の可変長です。先頭と末尾の 0 は切り捨てられます。数値のおおよそのサイズは、(属性名の UTF-8 でエンコードされたバイト数) + (有効桁数 2 あたり 1 バイト) + (1 バイト) です。
- バイナリ値を DynamoDB に送信するには base64 形式でエンコードする必要がありますが、サイズの計算には値の実際のバイト長が使用されます。バイナリ属性のサイズは、(属性名の UTF-8 でエンコードされたバイト数) + (raw バイト数) です。
- null 属性または Boolean 属性のサイズは、(属性名の UTF-8 でエンコードされたバイト数) + (1 バイト) です。
- `List` 型または `Map` 型の属性は、その内容にかかわらず、余分な 3 バイトが必要です。`List` または `Map` のサイズは、(属性名の UTF-8 でエンコードされたバイト数) + (入れ子要素のサイズの合

計) + (3 バイト) です。空の `List` または `Map` のサイズは、(属性名の UTF-8 でエンコードされたバイト数) + (3 バイト) です。

- `List` または `Map` の各要素には、余分な 1 バイトが必要です。

> Note
>
> 属性名は長いものよりも短いものにすることをお勧めします。これにより、必要なストレージの量を減らすことができ、使用する RCU/WCU の量を減らすこともできます。

ストレージの請求において、各項目には、有効にした機能に応じて、項目あたりのストレージオーバーヘッドが含まれます。

- DynamoDB のすべての項目は、インデックス作成に 100 バイトのストレージオーバーヘッドを必要とします。
- 一部の DynamoDB 機能 (グローバルテーブル、トランザクション、DynamoDB を使用した Kinesis Data Streams の変更データキャプチャ) では、これらの機能を有効にすることでシステムが作成した属性を考慮するために、追加のストレージオーバーヘッドが必要になります。例えば、グローバルテーブルでは、48 バイトのストレージオーバーヘッドがさらに必要になります。

リソースへのタグとラベルの追加

Amazon DynamoDB リソースにラベルを付けるには、タグを使用します。タグを使用すると、リソースを目的、所有者、環境、その他の条件などさまざまな方法で分類することができます。タグは、以下のことに役立ちます。

- リソースに割り当てたタグに基づいてリソースをすばやく特定します。
- タグ別に分類された AWS 請求を表示する。

> Note
>
> タグが付けられたテーブルに関連するローカルセカンダリインデックス (LSI) およびグローバルセカンダリインデックス (GSI) は、自動的に同じタグでラベルが付けられます。現在のところ、DynamoDB Streams の使用にタグを付けることはできません。

タグ付けは、Amazon EC2、Amazon S3、DynamoDB などの AWS のサービスでサポートされています。効率的なタグ付けを行うと、特定のタグを持つサービス間でレポートを作成でき、コストインサイトを得ることができます。

タグ付けの使用を開始するには、次のことを行います。

1. DynamoDB でのタグ付けの制限 について理解します。
2. DynamoDB でのタグ付けのリソース を使用してタグを作成します。
3. コスト配分レポート を使用して、アクティブなタグごとに AWS のコストを追跡します。

最後に、最適のタグ付け手法に従うことをお勧めします。詳細については、「AWS tagging strategies (タグ付け戦略)」を参照してください。

DynamoDB でのタグ付けの制限

タグはそれぞれ、1 つのキーと 1 つの値で構成されており、どちらもお客様側が定義します。以下の制限が適用されます。

- 各 DynamoDB テーブルは同じキーを含む 1 つのタグのみを持つことができます。既存のタグ (同じキー) を追加しようとすると、既存のタグの値は新しい値に更新されます。
- タグのキーと値は大文字と小文字が区別されます。
- キーの最大長は Unicode 文字で 128 文字です。
- 値の最大長は Unicode 文字で 256 文字です。
- 使用できる文字は、文字、ホワイトスペース、数字、および特殊文字 (+ - = . _ : /) です。
- リソースあたりのタグの最大数は 50 です。
- AWS が割り当てたタグの名前と値には自動的に `aws:` プレフィックスが付けられますが、これをユーザーが割り当てることはできません。AWS が割り当てたタグの名前は、ユーザーが定義したリソースタグの制限 50 個にカウントされません。ユーザー側で割り当てたタグ名は、`user:` というプレフィックスを付けてコスト配分レポートに表示されます。
- 過去にさかのぼってタグを適用することはできません。

DynamoDB でのタグ付けのリソース

Amazon DynamoDB コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、タグを追加、リスト、編集、または削除できます。次に、これらのユーザー定義タグをアクティブ化し、

コスト配分の追跡のため、AWS Billing and Cost Management コンソールに表示できます。詳細については、「コスト配分レポート」を参照してください。

一括編集の場合は、AWS Management Console のタグエディタを使用することもできます。詳細については、「タグエディタの使用」を参照してください。

代わりに DynamoDB API を使用するには、Amazon DynamoDB API リファレンスで以下のオペレーションを参照してください。

- TagResource
- UntagResource
- ListTagsOfResource

トピック

- タグでフィルターする権限の設定
- 新しいテーブルまたは既存のテーブルへのタグの追加 (AWS Management Console)
- 新しいテーブルまたは既存のテーブルへのタグの追加 (AWS CLI)

タグでフィルターする権限の設定

タグを使用して DynamoDB コンソールでテーブルリストをフィルタリングするには、ユーザーのポリシーに以下のオペレーションへのアクセス権が含まれていることを確認します。

- `tag:GetTagKeys`
- `tag:GetTagValues`

これらのオペレーションにアクセスするには、以下の手順に従って、新しい IAM ポリシーをユーザーにアタッチします。

1. 管理者ユーザーで IAM コンソールに移動します。
2. 左のナビゲーションメニューで [Policies (ポリシー)] を選択します。
3. [Create policy (ポリシーの作成)] を選択します。
4. 以下のポリシーを JSON エディタに貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. ウィザードを完了し、ポリシーに名前を割り当てます (例:`TagKeysAndValuesReadAccess`)。
6. 左のナビゲーションメニューで [Users (ユーザー)] を選択します。
7. リストから、DynamoDB コンソールにアクセスする通常のユーザーを選択します。
8. [Add permissions (アクセス許可の追加)] を選択します。
9. [Attach existing policies directly (既存のポリシーを直接アタッチ)] を選択します。
10. リストから、前に作成したポリシーを選択します。
11. ウィザードを終了します。

新しいテーブルまたは既存のテーブルへのタグの追加 (AWS Management Console)

新しいタグを作成するときや、既存のテーブルのタグを追加、編集、削除するときは、DynamoDB コンソールを使用してタグを新しいテーブルに追加できます。

作成時にリソースにタグを付けるには (コンソール)

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ナビゲーションペインで [テーブル] を選択して、[テーブルの作成] を選択します。
3. [Create DynamoDB table (DynamoDB テーブルの作成)] ページで、名前とプライマリキーを指定します。[タグ] セクションで、[新しいタグを追加] を選択し、使用するタグを入力します。

   タグ構造の詳細については、「DynamoDB でのタグ付けの制限」を参照してください。

   テーブル作成の詳細については、「DynamoDB テーブルの基本的なオペレーション」を参照してください。

既存のリソースにタグを付けるには (コンソール)

https://console.aws.amazon.com/dynamodb/ で DynamoDB コンソールを開きます。

1. ナビゲーションペインで [テーブル] を選択します。
2. リストでテーブルを選択し、[追加設定] タブを選択します。ページの下部にある [タグ] セクションで、タグを追加、編集、削除できます。

新しいテーブルまたは既存のテーブルへのタグの追加 (AWS CLI)

次の例は、テーブルとインデックスを作成するとき、および既存のリソースへのタグを付けるときに、AWS CLI を使用してタグを指定する方法を示しています。

作成時にリソースにタグを付けるには (AWS CLI)

- 次の例では、新しい Movies テーブルを作成し、値が Owner の blueTeam タグを追加します。

```
aws dynamodb create-table \
    --table-name Movies \
    --attribute-definitions AttributeName=Title,AttributeType=S \
    --key-schema AttributeName=Title,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

既存のリソースにタグを付けるには (AWS CLI)

- 次の例では、Owner テーブル用に、値が blueTeam の Movies タグを追加します。

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

テーブルのすべてのタグを一覧表示するには (AWS CLI)

- 次の例では、Movies テーブルに関連付けられたすべてのタグを一覧表示します。

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

コスト配分レポート

AWS はタグを使用して、コスト配分レポートでリソースコストを分類します。AWS には 2 つのタイプのコスト配分タグがあります。

- AWS で生成されたタグ。AWS はユーザーのためにこのタグを定義、作成、適用します。
- ユーザー定義のタグ。これらのタグを定義、作成、適用します。

Cost Explorer またはコスト配分レポートで使用するには、事前に両方のタイプのタグを別々にアクティブ化しておく必要があります。

AWS で生成されたタグをアクティブ化するには:

1. AWS Management Console にサインインし、https://console.aws.amazon.com/billing/home#/ で請求およびコスト管理コンソールを開きます。
2. ナビゲーションペインで、[Cost Allocation Tags] (コスト配分タグ) を選択します。
3. [AWS-Generated Cost Allocation Tags] で、[Activate] を選択します。

ユーザー定義タグをアクティブにするには:

1. AWS Management Console にサインインし、https://console.aws.amazon.com/billing/home#/ で請求およびコスト管理コンソールを開きます。
2. ナビゲーションペインで、[Cost Allocation Tags] (コスト配分タグ) を選択します。
3. [User-Defined Cost Allocation Tags] (ユーザー定義のコスト配分タグ) で、[Activate] (アクティブ化) を選択します。

タグを作成してアクティブ化すると、AWS は、アクティブなタグごとにグループ化された使用量とコストを含むコスト配分レポートを生成します。コスト配分レポートには、ご利用の AWS のサービスのコストすべてが請求期間別に出力されます。タグ付きとタグなしのどちらのリソースもこのレポートに出力されるので、リソース別の請求額を明確に分類できます。

> Note
>
> 現時点では、DynamoDB から転送されたデータは、コスト配分レポートのタグによって分類されません。

詳細については、「Using cost allocation tags」(コスト配分タグの使用) を参照してください。

Java での DynamoDB テーブルの使用

AWS SDK for Java を使用して、Amazon DynamoDB テーブルの作成、更新、削除、アカウント内の全テーブルの一覧表示、特定のテーブルに関する情報収集を実行できます。

トピック

テーブルの作成

テーブルを作成するには、テーブル名、プライマリキー、およびプロビジョニングされたスループット値を指定する必要があります。以下の コードスニペットでは、数値型の属性 ID をプライマリキーとして使用するサンプルテーブルを作成します。

AWS SDK for Java API を使用してテーブルを作成するには

1. `DynamoDB` クラスのインスタンスを作成します。
2. `CreateTableRequest` をインスタンス化して、リクエスト情報を指定します。

   テーブル名、属性定義、キースキーマ、プロビジョニングされたスループット値を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、`createTable` メソッドを実行します。

以下のサンプルコードは、前述のステップの例です。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

List<AttributeDefinition> attributeDefinitions= new ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH));

CreateTableRequest request = new CreateTableRequest()
        .withTableName(tableName)
        .withKeySchema(keySchema)
        .withAttributeDefinitions(attributeDefinitions)
        .withProvisionedThroughput(new ProvisionedThroughput()
            .withReadCapacityUnits(5L)
            .withWriteCapacityUnits(6L));

Table table = dynamoDB.createTable(request);

table.waitForActive();
```

DynamoDB によりテーブルが作成され、そのステータスが ACTIVE に設定されるまでは、テーブルを使用できません。`createTable` リクエストは、テーブルについての詳細情報を取得するために使用できる `Table` オブジェクトを返します。

Example

```
TableDescription tableDescription =
    dynamoDB.getTable(tableName).describe();

System.out.printf("%s: %s \t ReadCapacityUnits: %d \t WriteCapacityUnits: %d",
    tableDescription.getTableStatus(),
    tableDescription.getTableName(),
    tableDescription.getProvisionedThroughput().getReadCapacityUnits(),
    tableDescription.getProvisionedThroughput().getWriteCapacityUnits());
```

クライアントの `describe` メソッドを呼び出せば、いつでもテーブル情報を収集できます。

Example

```
TableDescription tableDescription = dynamoDB.getTable(tableName).describe();
```

テーブルの更新

既存のテーブルのプロビジョニングされたスループット値のみを更新できます。ご利用のアプリケーションの要件によっては、これらの値を更新する必要があります。

> **Note**
>
> スループットの 1 日の増減については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。

AWS SDK for Java API を使用してテーブルを更新するには

1. Table クラスのインスタンスを作成します。
2. ProvisionedThroughput クラスのインスタンスを作成して、新しいスループット値を指定します。
3. updateTable インスタンスをパラメータとして指定して、ProvisionedThroughput メソッドを実行します。

以下のサンプルコードは、前述のステップの例です。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

ProvisionedThroughput provisionedThroughput = new ProvisionedThroughput()
    .withReadCapacityUnits(15L)
    .withWriteCapacityUnits(12L);

table.updateTable(provisionedThroughput);

table.waitForActive();
```

テーブルの削除

AWS SDK for Java API を使用してテーブルを削除するには

1. `Table` クラスのインスタンスを作成します。
2. `DeleteTableRequest` クラスのインスタンスを作成し、削除するテーブル名を指定します。
3. `deleteTable` インスタンスをパラメータとして指定して、`Table` メソッドを実行します。

以下のサンプルコードは、前述のステップの例です。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

table.delete();

table.waitForDelete();
```

テーブルの一覧表示

アカウントのテーブルを一覧表示するには、`DynamoDB` のインスタンスを作成し、`listTables` メソッドを実行します。ListTables オペレーションはパラメータを必要としません。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableCollection<ListTablesResult> tables = dynamoDB.listTables();
Iterator<Table> iterator = tables.iterator();

while (iterator.hasNext()) {
    Table table = iterator.next();
    System.out.println(table.getTableName());
}
```

例: AWS SDK for Java ドキュメント API を使用したテーブルの作成、更新、削除、一覧表示

以下のコード例では、AWS SDK for Java ドキュメント API を使用して、Amazon DynamoDB テーブル (ExampleTable) を作成、更新、および削除します。このテーブル更新の一環として、プロビジョニングされたスループット値が増加します。この例では、アカウント内にすべてのテーブルが一覧表示され、固有のテーブルの説明が取得されます。以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.TableCollection;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ListTablesResult;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.TableDescription;

public class DocumentAPITableExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ExampleTable";

    public static void main(String[] args) throws Exception {

        createExampleTable();
        listMyTables();
        getTableInformation();
        updateExampleTable();
```

```
       deleteExampleTable();
   }

   static void createExampleTable() {

       try {

           List<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
           attributeDefinitions.add(new
AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

           List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
           keySchema.add(new
KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

             // key

           CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)

.withAttributeDefinitions(attributeDefinitions).withProvisionedThroughput(
                           new
ProvisionedThroughput().withReadCapacityUnits(5L).withWriteCapacityUnits(6L));

           System.out.println("Issuing CreateTable request for " + tableName);
           Table table = dynamoDB.createTable(request);

           System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
           table.waitForActive();

           getTableInformation();

       } catch (Exception e) {
           System.err.println("CreateTable request failed for " + tableName);
           System.err.println(e.getMessage());
       }

   }

   static void listMyTables() {

       TableCollection<ListTablesResult> tables = dynamoDB.listTables();
```

```
        Iterator<Table> iterator = tables.iterator();

        System.out.println("Listing table names");

        while (iterator.hasNext()) {
            Table table = iterator.next();
            System.out.println(table.getTableName());
        }
    }

    static void getTableInformation() {

        System.out.println("Describing " + tableName);

        TableDescription tableDescription = dynamoDB.getTable(tableName).describe();
        System.out.format(
                "Name: %s:\n" + "Status: %s \n" + "Provisioned Throughput (read 
capacity units/sec): %d \n"
                        + "Provisioned Throughput (write capacity units/sec): %d \n",
                tableDescription.getTableName(), tableDescription.getTableStatus(),
                tableDescription.getProvisionedThroughput().getReadCapacityUnits(),
                tableDescription.getProvisionedThroughput().getWriteCapacityUnits());
    }

    static void updateExampleTable() {

        Table table = dynamoDB.getTable(tableName);
        System.out.println("Modifying provisioned throughput for " + tableName);

        try {
            table.updateTable(new 
ProvisionedThroughput().withReadCapacityUnits(6L).withWriteCapacityUnits(7L));

            table.waitForActive();
        } catch (Exception e) {
            System.err.println("UpdateTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    static void deleteExampleTable() {

        Table table = dynamoDB.getTable(tableName);
        try {
            System.out.println("Issuing DeleteTable request for " + tableName);
            table.delete();

            System.out.println("Waiting for " + tableName + " to be deleted...this may
 take a while...");

            table.waitForDelete();
        } catch (Exception e) {
            System.err.println("DeleteTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

}
```

.NET での DynamoDB テーブルの使用

AWS SDK for .NET を使用して、テーブルの作成、更新、削除、アカウント内の全テーブルの一覧表示、特定のテーブルに関する情報収集を実行できます。

以下に示しているのは、AWS SDK for .NET を使用した Amazon DynamoDB テーブルオペレーションの一般的なステップです。

1. `AmazonDynamoDBClient` クラスのインスタンス（クライアント）を作成します。
2. 対応するリクエストオブジェクトを作成して、オペレーションについて必要なパラメータとオプションパラメータを入力します。

   たとえば、テーブルを作成するには `CreateTableRequest` オブジェクトを作成し、既存のテーブルを更新するには `UpdateTableRequest` オブジェクトを作成します。
3. 前述のステップで作成したクライアントから提供された適切なメソッドを実行します。

> **Note**
>
> このセクションの例は、同期メソッドをサポートしていないため、.NET Core には適用されません。詳細については、「.NET 用 AWS 非同期 API」を参照してください。

トピック

- テーブルの作成
- テーブルの更新
- テーブルの削除
- テーブルの一覧表示
- 例: AWS SDK for .NET 低レベル API を使用したテーブルの作成、更新、削除、一覧表示

テーブルの作成

テーブルを作成するには、テーブル名、プライマリキー、およびプロビジョニングされたスループット値を指定する必要があります。

AWS SDK for .NET 低レベル API を使用してテーブルを作成するには

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. リクエスト情報を指定する CreateTableRequest クラスのインスタンスを作成します。

   テーブル名、プライマリキー、およびプロビジョニングされたスループット値を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、AmazonDynamoDBClient.CreateTable メソッドを実行します。

以下の C# サンプルは、前述のステップの例です。このサンプルでは、ProductCatalog をプライマリキーとして使用するテーブル (Id)、およびプロビジョニングされたスループット値のセットを作成します。ご利用のアプリケーションの要件によっては、UpdateTable API を使用して、プロビジョニングされたスループット値を更新できます。

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new CreateTableRequest
{
  TableName = tableName,
  AttributeDefinitions = new List<AttributeDefinition>()
  {
    new AttributeDefinition
    {
      AttributeName = "Id",
      AttributeType = "N"
    }
  },
  KeySchema = new List<KeySchemaElement>()
  {
    new KeySchemaElement
    {
      AttributeName = "Id",
      KeyType = "HASH"  //Partition key
    }
  },
  ProvisionedThroughput = new ProvisionedThroughput
  {
    ReadCapacityUnits = 10,
    WriteCapacityUnits = 5
  }
};

var response = client.CreateTable(request);
```

DynamoDB がテーブルを作成し、そのステータスを ACTIVE に設定するまで待機する必要があります。CreateTable 応答には、必要なテーブル情報が記載された TableDescription プロパティが含まれます。

Example

```
var result = response.CreateTableResult;
 var tableDescription = result.TableDescription;
 Console.WriteLine("{1}: {0} \t ReadCapacityUnits: {2} \t WriteCapacityUnits: {3}",
                tableDescription.TableStatus,
                tableDescription.TableName,
                tableDescription.ProvisionedThroughput.ReadCapacityUnits,
                tableDescription.ProvisionedThroughput.WriteCapacityUnits);

 string status = tableDescription.TableStatus;
 Console.WriteLine(tableName + " - " + status);
```

また、クライアントの DescribeTable メソッドを呼び出せば、いつでもテーブル情報を収集できます。

Example

```
var res = client.DescribeTable(new DescribeTableRequest{TableName = "ProductCatalog"});
```

テーブルの更新

既存のテーブルのプロビジョニングされたスループット値のみを更新できます。ご利用のアプリケーションの要件によっては、これらの値を更新する必要があります。

> Note
>
> スループットキャパシティーは、必要に応じて増やしたり、特定の制限内で減らしたりできます。スループットの 1 日の増減については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。

AWS SDK for .NET 低レベル API を使用してテーブルを更新するには

1. `AmazonDynamoDBClient` クラスのインスタンスを作成します。
2. リクエスト情報を指定する `UpdateTableRequest` クラスのインスタンスを作成します。

   テーブル名およびプロビジョニングされたスループット値を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、`AmazonDynamoDBClient.UpdateTable` メソッドを実行します。

以下の C# サンプルは、前述のステップの例です。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ExampleTable";

var request = new UpdateTableRequest()
 {
   TableName = tableName,
   ProvisionedThroughput = new ProvisionedThroughput()
   {
     // Provide new values.
     ReadCapacityUnits = 20,
     WriteCapacityUnits = 10
   }
 };
var response = client.UpdateTable(request);
```

テーブルの削除

.NET 低レベル API を使用してテーブルを削除するには次の手順に従います。

AWS SDK for .NET 低レベル API を使用してテーブルを削除するには

1. `AmazonDynamoDBClient` クラスのインスタンスを作成します。
2. `DeleteTableRequest` クラスのインスタンスを作成し、削除するテーブル名を指定します。
3. リクエストオブジェクトをパラメータとして指定して、`AmazonDynamoDBClient.DeleteTable` メソッドを実行します。

以下の C# サンプルコードは、前述のステップの例です。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ExampleTable";

var request = new DeleteTableRequest{ TableName = tableName };
var response = client.DeleteTable(request);
```

テーブルの一覧表示

AWS SDK for .NET 低レベル API を使用してアカウント内にテーブルを一覧表示するには、`AmazonDynamoDBClient` のインスタンスを作成し、`ListTables` メソッドを実行します。

ListTables オペレーションはパラメータを必要としません。ただし、オプションパラメータを指定できます。たとえば、ページングを使用して 1 ページあたりのテーブル名を制限する場合には、`Limit` パラメータを設定できます。この場合は、以下の C# サンプルに示すように、`ListTablesRequest` オブジェクトを作成し、オプションパラメータを指定する必要があります。ページサイズに加え、このリクエストでは、`ExclusiveStartTableName` パラメータを設定します。最初は、`ExclusiveStartTableName` は null です。だし、最初のページの結果を取り出した後に次ページの結果を取り出すには、このパラメータ値を、現在の結果の `LastEvaluatedTableName` プロパティに設定する必要があります。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

// Initial value for the first page of table names.
```

```
string lastEvaluatedTableName = null;
do
{
   // Create a request object to specify optional parameters.
   var request = new ListTablesRequest
   {
     Limit = 10, // Page size.
     ExclusiveStartTableName = lastEvaluatedTableName
   };

   var response = client.ListTables(request);
   ListTablesResult result = response.ListTablesResult;
   foreach (string name in result.TableNames)
     Console.WriteLine(name);

   lastEvaluatedTableName = result.LastEvaluatedTableName;

} while (lastEvaluatedTableName != null);
```

例: AWS SDK for .NET 低レベル API を使用したテーブルの作成、更新、削除、一覧表示

次の C# サンプルでは、テーブル (ExampleTable) を作成、更新、および削除します。アカウント内にすべてのテーブルが一覧表示され、固有のテーブルの説明が取得されます。このテーブル更新によって、プロビジョニングされたスループット値が増加します。次の例をテストするための詳しい手順については、「.NET コード例」を参照してください。

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelTableExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "ExampleTable";

        static void Main(string[] args)
        {
            try
            {
                CreateExampleTable();
                ListMyTables();
                GetTableInformation();
                UpdateExampleTable();

                DeleteExampleTable();

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateExampleTable()
        {
            Console.WriteLine("\n*** Creating table ***");
            var request = new CreateTableRequest
            {
                AttributeDefinitions = new List<AttributeDefinition>()
            {
                new AttributeDefinition
                {
                    AttributeName = "Id",
                    AttributeType = "N"
                },
                new AttributeDefinition
                {
                    AttributeName = "ReplyDateTime",
                    AttributeType = "N"
                }
            },
                KeySchema = new List<KeySchemaElement>
            {
                new KeySchemaElement
                {
                    AttributeName = "Id",
                    KeyType = "HASH" //Partition key
                },
                new KeySchemaElement
                {
                    AttributeName = "ReplyDateTime",
                    KeyType = "RANGE" //Sort key
                }
            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 6
                },
                TableName = tableName
            };

            var response = client.CreateTable(request);

            var tableDescription = response.TableDescription;
            Console.WriteLine("{1}: {0} \t ReadsPerSec: {2} \t WritesPerSec: {3}",
                      tableDescription.TableStatus,
                      tableDescription.TableName,
                      tableDescription.ProvisionedThroughput.ReadCapacityUnits,
                      tableDescription.ProvisionedThroughput.WriteCapacityUnits);

            string status = tableDescription.TableStatus;
            Console.WriteLine(tableName + " - " + status);

            WaitUntilTableReady(tableName);
        }

        private static void ListMyTables()
        {
            Console.WriteLine("\n*** listing tables ***");
            string lastTableNameEvaluated = null;
            do
            {
                var request = new ListTablesRequest
                {
                    Limit = 2,
                    ExclusiveStartTableName = lastTableNameEvaluated
                };

                var response = client.ListTables(request);
                foreach (string name in response.TableNames)
                    Console.WriteLine(name);

                lastTableNameEvaluated = response.LastEvaluatedTableName;
            } while (lastTableNameEvaluated != null);
        }

        private static void GetTableInformation()
        {
            Console.WriteLine("\n*** Retrieving table information ***");
            var request = new DescribeTableRequest
            {
                TableName = tableName
            };

            var response = client.DescribeTable(request);

            TableDescription description = response.Table;
            Console.WriteLine("Name: {0}", description.TableName);
            Console.WriteLine("# of items: {0}", description.ItemCount);
            Console.WriteLine("Provision Throughput (reads/sec): {0}",
                      description.ProvisionedThroughput.ReadCapacityUnits);
            Console.WriteLine("Provision Throughput (writes/sec): {0}",
                      description.ProvisionedThroughput.WriteCapacityUnits);
        }

        private static void UpdateExampleTable()
        {
            Console.WriteLine("\n*** Updating table ***");
            var request = new UpdateTableRequest()
            {
                TableName = tableName,
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 6,
                    WriteCapacityUnits = 7
                }
            };

            var response = client.UpdateTable(request);

            WaitUntilTableReady(tableName);
        }

        private static void DeleteExampleTable()
        {
            Console.WriteLine("\n*** Deleting table ***");
            var request = new DeleteTableRequest
            {
                TableName = tableName
            };

            var response = client.DeleteTable(request);

            Console.WriteLine("Table is being deleted...");
        }

        private static void WaitUntilTableReady(string tableName)
        {
            string status = null;
            // Let us wait until table is created. Call DescribeTable.
            do
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                catch (ResourceNotFoundException)
                {
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }
    }
}
```

グローバルテーブル – DynamoDB の複数リージョンレプリケーション

Amazon DynamoDB グローバルテーブルにより、フルマネージドで、マルチリージョン、マルチアクティブなデータベースオプションが提供されます。このオプションでは、大規模に拡張されたグローバルアプリケーションからの読み込み/書き込みのパフォーマンスが高速に、かつローカルに実現されます。

グローバルテーブルは、マルチリージョン、マルチアクティブデータベースをデプロイするためのフルマネージドソリューションです。独自のレプリケーションソリューションを構築および管理する必要はありません。テーブルを利用したい AWS リージョンを指定すると、DynamoDB が進行中のデータ変更をそれらすべてに伝播します。

グローバルテーブルを使用する具体的な利点は次のとおりです。

- 選択する AWS リージョン全体で DynamoDB テーブルを自動的にレプリケートする
- リージョン間でデータをレプリケートし、更新の競合を解決するという困難な作業が不要になるため、アプリケーションのビジネスロジックに集中できます。
- 可能性は低いものの、リージョン全体が分離またはパフォーマンスが低下した場合でも、アプリケーションの可用性を高く保つのに役立ちます。

DynamoDB グローバルテーブルは、グローバルに分散したユーザーがいる場合など、大規模にスケールされたアプリケーションを使用する場合に最適です。このような環境では、ユーザーは、非常に高速なアプリケーションパフォーマンスを期待します。グローバルテーブルでは、マルチアクティブレプリケーションを世界中の AWS リージョンに自動的に提供できます。ユーザーがどこにいても、低レイテンシーでデータを提供することができます。

次の動画では、グローバルテーブルの概要を紹介します。

グローバルテーブルは、AWS 管理コンソールまたは AWS CLI で設定できます。グローバルテーブルは既存の DynamoDB API を使用するため、アプリケーションを変更する必要はありません。プロビジョニングされたリソースに対してのみ料金が発生します。前払い料金や契約はありません。

リージョン間レプリケーションのグローバルテーブル

トピック

- グローバルテーブルを使用してリージョン間でシームレスにデータをレプリケートする
- AWS KMS を使用してグローバルテーブルにセキュリティとアクセスを提供する

グローバルテーブルを使用してリージョン間でシームレスにデータをレプリケートする

米国東海岸、米国西海岸、西ヨーロッパという 3 つの地域領域にわたる大規模な顧客ベースがあるとします。これらのユーザーは、アプリケーションの使用中に自分のプロファイル情報を更新できます。このユースケースを満たすには、ユーザーが所在する 3 つの異なる `CustomerProfiles` リージョンで、AWS という名前の同一の DynamoDB テーブルを 3 つ作成する必要があります。これらの 3 つのテーブルは互いに完全に分離され、1 つのテーブルのデータを変更しても他のテーブルに反映されることはありません。マネージド型のレプリケーションソリューションを使用しない場合、コードを書き込んで、データ変更をレプリケートする必要があります。ただし、これを行うには時間も労力もかかります。

独自のコードを書き込む代わりに、3 つのリージョン固有の `CustomerProfiles` テーブルで構成されるグローバルテーブルを作成できます。DynamoDB は、これらのテーブル間でデータの変更を自動的にレプリケートし、1 つのリージョンの `CustomerProfiles` データへの変更が他のリージョンにシームレスに伝播されるようにします。さらに、いずれかの AWS リージョンが一時的に利用できない場合は、顧客は他のリージョンの同じ `CustomerProfiles` データにアクセスできます。

> **Note**
>
> - グローバルテーブル グローバルテーブルバージョン 2017.11.29 (レガシー) のリージョンのサポートは、米国東部 (バージニア北部)、米国東部 (オハイオ)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、欧州 (アイルランド)、欧州 (ロンドン)、欧州 (フランクフルト)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、アジアパシフィック (ソウル) に制限されています。
> - トランザクションオペレーションは、書き込みが最初に行われたリージョン内でのみ、不可分性、一貫性、分離性、および耐久性 (ACID) を保証します。グローバルテーブルの

リージョン間では、トランザクションはサポートされていません。たとえば、米国東部（オハイオ）および米国西部（オレゴン）リージョンにレプリカを持つグローバルテーブルがあり、米国東部（バージニア北部）リージョンで TransactWriteItems オペレーションを実行する場合、変更がレプリケートされると米国西部（オレゴン）で部分的に完了したトランザクションを確認できます。変更は、ソースリージョンでコミットされると、他のリージョンにのみレプリケートされます。

- AWS リージョンを無効にすると、DynamoDB は、AWS リージョンにアクセスできないことが検出されてから 20 時間後に、このレプリカをレプリケーショングループから削除します。レプリカは削除されず、このリージョンとの間でレプリケーションが停止します。
- ソーステーブルを正常に削除するには、リードレプリカを追加してから 24 時間待つ必要があります。リードレプリカを追加してから最初の 24 時間以内にテーブルを削除しようとすると、「過去 24 時間以内にテーブルに追加された新しいレプリカのソースリージョンとして機能しているため、レプリカを削除できません」というエラーメッセージが表示されます。
- 新しいレプリカを追加しても、ソースリージョンのパフォーマンスへの影響はありません。
- レプリカの読み取りキャパシティと書き込みキャパシティを変更すると、新しい書き込みキャパシティは他の同期レプリカに反映されますが、新しい読み取りキャパシティは反映されません。

AWS リージョンの対応状況と料金については、「Amazon DynamoDB 料金」を参照してください。

AWS KMS を使用してグローバルテーブルにセキュリティとアクセスを提供する

- カスタマーマネージドキーまたはレプリカの暗号化に使用される AWS マネージドキー に対して `AWSServiceRoleForDynamoDBReplication` サービスにリンクされたロールを使用して、グローバルテーブルの AWS KMS オペレーションを実行できます。
- レプリカの暗号化に使用されるカスタマーマネージドキーにアクセスできない場合、DynamoDB はこのレプリカをレプリケーショングループから削除します。KMS キーにアクセスできないことが検出されてから 20 時間後に、レプリカは削除されず、このリージョンとの間でレプリケーションが停止します。
- レプリカテーブルの暗号化に使用されるカスタマーマネージドキー を無効にする場合は、キーがレプリカテーブルの暗号化に使用されなくなった場合にのみ無効にする必要があります。レプリ

カテーブルを削除するコマンドを発行した後、キーを無効にする前に、削除オペレーションが完了し、グローバルテーブルが `Active` になるのを待つ必要があります。そうしないと、レプリカテーブルとの間で部分的なデータレプリケーションが行われる可能性があります。

- レプリカテーブルの IAM ロールポリシーを変更または削除する場合は、レプリカテーブルが `Active` 状態のときに行う必要があります。この操作を行わないと、レプリカテーブルの作成、更新、削除が失敗する可能性があります。
- グローバルテーブルは、作成時、デフォルトで削除保護が無効になっています。グローバルテーブルで削除保護が有効になっている場合でも、そのテーブルのレプリカの最初の状態ではデフォルトで削除保護が無効になっています。
- テーブルの削除保護が無効になっている間は、誤って削除される可能性があります。テーブルの削除保護が有効になっている間は、誰も削除できません。
- 1 つのレプリカテーブルの削除保護設定を変更しても、グループ内の他のレプリカは更新されません。

> Note
>
> カスタマーマネージドキーは、グローバルテーブルバージョン 2017.11.29 (レガシー) でサポートされていません。DynamoDB グローバルテーブルでカスタマーマネージドキーを使用する場合は、テーブルをグローバルテーブルバージョン 2019.11.21 (現行) にアップグレードしてから有効にする必要があります。

グローバルテーブル: 仕組み

以下のセクションでは、Amazon DynamoDB におけるグローバルテーブルの概念および動作について説明します。

グローバルテーブルの概念

グローバルテーブルは 1 つ以上のレプリカテーブルの集合体であり、すべて単一の AWS アカウントが所有します。

レプリカテーブル (または、略してレプリカ) は、グローバルテーブルの一環として機能する単一の DynamoDB テーブルです。各レプリカには、同じデータ項目のセットが保存されます。指定のグローバルテーブルは、AWS リージョンごとに 1 つのレプリカテーブルのみを持つことができます。グローバルテーブルの詳しい使用方法については、チュートリアル: グローバルテーブルの作成 を参照してください。

DynamoDB グローバルテーブルを作成すると、DynamoDB は 1 つの単位として扱う複数のレプリカテーブル (リージョンごとに 1 つ) で構成されます。すべてのレプリカは、同じテーブル名と同じプライマリキーのスキーマを持っています。アプリケーションが 1 つのリージョンのレプリカテーブルにデータを書き込むと、DynamoDB はその書き込みを他の AWS リージョンの他のレプリカテーブルに自動的に伝播します。

レプリカテーブルをグローバルテーブルに追加して、追加のリージョンで使用できるようにすることができます。

バージョン 2019.11.21 (現行) では、あるリージョンでグローバルセカンダリインデックスを作成すると、自動的に他のリージョンにレプリケートされ、自動的にバックフィルされます。

一般的なタスク

グローバルテーブルの一般的なタスクは以下のように機能します。

グローバルテーブルのレプリカテーブルは、通常のテーブルと同じように削除できます。これにより、そのリージョンへのレプリケーションが停止し、そのリージョンに保存されているテーブルコピーが削除されます。レプリケーションを切断して、テーブルのコピーを独立したエンティティとすることはできません。グローバルテーブルをリージョンのローカルテーブルにコピーしてから、リージョンのグローバルレプリカを削除することはできます。

> Note
>
> ソーステーブルは、新しいリージョンの作成に使用されてから 24 時間以上経たないと削除できません。削除を試みるのが早すぎると、エラーが発生します。

アプリケーションが異なるリージョンにある同一項目をほぼ同時に更新すると、競合が発生する可能性があります。結果整合性を確保するために、DynamoDB グローバルテーブルでは同時更新間の調整のために、「最新書き込み優先」方法が使用されます。すべてのレプリカが最新の更新に合意し、すべてのレプリカが同一のデータを持つ状態に収束します。

> Note
>
> 競合を回避する方法は、以下を含めていくつかあります。
>
> - 1 つのリージョンのテーブルへの書き込みのみを許可します。
> - 競合が発生しないように、書き込みポリシーに従ってユーザートラフィックを別のリージョンにルーティングします。

- Bookmark = Bookmark + 1 のような非冪等の更新は避け、Bookmark=25 のような静的な更新を優先します。
- 書き込みまたは読み込みを 1 つのリージョンにルーティングする場合、そのフローを確保するのはアプリケーションの責任であることに注意してください。

グローバルテーブルのモニタリング

CloudWatch を使用して、`ReplicationLatency` メトリクスをモニタリングできます。これは、項目がレプリカテーブルに書き込まれてから、その項目がグローバルテーブルの別のレプリカに表示されるまでの経過時間を追跡します。ミリ秒単位で表され、ソースリージョンとレプリケーション先リージョンのすべてのペアに対して出力されます。このメトリクスはソースリージョンで保持されます。これは、グローバルテーブル v2 が提供する唯一の CloudWatch メトリクスです。

ユーザーが経験するレプリケーションレイテンシーは、選択した AWS リージョン 間の距離やその他の変数によって異なります。元のテーブルが米国西部 (北カリフォルニア) (us-west-1) リージョンにある場合、米国西部 (オレゴン) (us-west-2) リージョンなどの近いリージョンのレプリカは、アフリカ (ケープタウン) (af-south-1) リージョンなどの遠く離れたリージョンのレプリカよりもレプリケーションレイテンシーが低くなります。

> **Note**
>
> レプリケーションレイテンシーは API のレイテンシーには影響しません。リージョン A にクライアントとテーブルがあり、リージョン B にグローバルテーブルレプリカを追加した場合、リージョン A のクライアントとテーブルのレイテンシーはリージョン B への追加前と同じになります。リージョン B で PutItem API オペレーションを呼び出すと、Amazon CloudWatch で利用できる `ReplicationLatency` 統計とほぼ同じ遅延の後に、リージョン A で読み込みが可能になります。レプリケートされる前は空のレスポンスを受け取り、レプリケートされた後はアイテムを受け取ります。どちらの呼び出しでも API レイテンシーはほぼ同じです。

有効期限 (TTL)

Time To Live (TTL) を使用して、項目の有効期限を示す値を含む属性名を指定できます。この値は、Unix エポックの開始からの秒単位の数値として指定します。その後、DynamoDB は書き込みコストを発生させずに項目を削除できます。

グローバルテーブルでは、1 つのリージョンで TTL を設定すると、その設定が他のリージョンに自動的にレプリケートされます。TTL ルールによって項目が削除された場合、その作業はソーステーブルの書き込みユニットを消費することなく実行されますが、ターゲットテーブルにはレプリケーション書き込みユニットのコストが発生します。

ソーステーブルとターゲットテーブルのプロビジョニングされた書き込みキャパシティが非常に少ない場合、TTL 削除には書き込みキャパシティが必要となるため、スロットリングが発生する可能性があります。

グローバルテーブルでのストリームとトランザクション

各グローバルテーブルは、書き込みの開始点に関係なく、すべての書き込みに基づいて独立したストリームを生成します。この DynamoDB ストリームを 1 つのリージョンで使用するか、すべてのリージョンで個別に使用するかを選択できます。

レプリケートされた書き込みではなくローカル書き込みを処理する場合は、各項目に独自のリージョン属性を追加できます。次に、Lambda イベントフィルターを使用して、ローカルリージョンへの書き込みに対して Lambda のみを呼び出すことができます。

トランザクションオペレーションは、書き込みが最初に行われたリージョン内でのみ、ACID (不可分性、一貫性、分離性、耐久性) を保証します。グローバルテーブルのリージョン間では、トランザクションはサポートされていません。

例えば、米国東部 (オハイオ) リージョンと米国西部 (オレゴン) リージョンにレプリカを持つグローバルテーブルがある場合、米国東部 (オハイオ) リージョンで `TransactWriteItems` オペレーションを実行すると、変更がレプリケートされたときに米国西部 (オレゴン) では部分的に完了したトランザクションが観察されることがあります。変更は、ソースリージョンでコミットされた場合にのみ、他のリージョンにレプリケートされます。

> **Note**
>
> - グローバルテーブルは DynamoDB を直接更新することで、DynamoDB Accelerator を「書き込み迂回」します。その結果、DAX は古いデータを保持していることを認識しません。DAX キャッシュは、キャッシュの TTL が期限切れになったときにのみ更新されます。
> - グローバルテーブルのタグは自動的には伝播されません。

読み取りおよび書き込みスループット

グローバルテーブルは、次の方法で読み取りと書き込みのスループットを管理します。

- 書き込みキャパシティーは、リージョン間のすべてのテーブルインスタンスで同じでなければなりません。
- バージョン 2019.11.21 (現行) では、テーブルが自動スケーリングをサポートするように設定されているか、オンデマンドモードになっている場合、書き込みキャパシティは自動的に同期されます。つまり、1 つのテーブルへの書き込みキャパシティーの変更は、他のテーブルにもレプリケートされます。
- 読み取りキャパシティーはリージョンによって異なる場合があります。これは、読み取りキャパシティーが等しくない場合があるためです。グローバルレプリカをテーブルに追加すると、ソースリージョンのキャパシティーが反映されます。作成後、1 つのレプリカの読み込みキャパシティを調整できますが、この新しい設定は他方には転送されません。

整合性と競合の解決

レプリカテーブル内の項目に加えられた変更は、同じグローバルテーブル内の他のすべてのレプリカにレプリケートされます。グローバルテーブルでは、新しく書き込まれた項目は通常、1 秒以内にすべてのレプリカテーブルに伝播されます。

グローバルテーブルでは、各レプリカテーブルには同じデータ項目のセットが保存されます。DynamoDB では、一部の項目のみの部分的なレプリケーションはサポートされていません。

アプリケーションは、任意のレプリカテーブルとの間でデータを読み込みおよび書き込みできます。アプリケーションが結果整合性のある読み込みのみを使用し、1 つの AWS リージョンに対して読み込みのみを発行する場合、変更を加えずに動作します。ただし、アプリケーションで強力な整合性のある読み込みが必要な場合は、同じリージョンですべての強力な整合性のある読み込みと書き込みを実行する必要があります。DynamoDB は、リージョン間での強力な整合性のある読み込みをサポートしていません。したがって、あるリージョンに書き込んで別のリージョンから読み込む場合、読み込みの応答には、他のリージョンで最近完了した書き込みの結果を反映していない古いデータが含まれる可能性があります。

アプリケーションが異なるリージョンにある同一項目をほぼ同時に更新すると、競合が発生する可能性があります。結果整合性を確保するために、DynamoDB グローバルテーブルでは最終書き込み者優先を使用して、同時更新間の調整を行い、DynamoDB は最終書き込み者を判断するために最善を尽くします。これは項目レベルで実行されます。この競合解決メカニズムでは、すべてのレプリカが最新の更新に合意し、すべてのレプリカが同一のデータを持つ状態に収束します。

可用性と耐久性

単一の AWS リージョンが分離または低下した場合、アプリケーションは別のリージョンにリダイレクトし、別のレプリカテーブルに対して読み込みと書き込みを実行できます。カスタムビジネスロジックを適用して、リクエストを他のリージョンにリダイレクトするタイミングを決定できます。

リージョンの分離または機能低下が発生した場合、DynamoDB は、実行されてもすべてのレプリカテーブルにはまだ反映されていない書き込みを追跡します。リージョンがオンラインに戻ると、DynamoDB はそのリージョンから他のリージョンのレプリカテーブルへの保留中の書き込みの伝播を再開します。また、他のレプリカテーブルから現在オンラインに戻っているリージョンへの書き込みの伝播も再開します。

グローバルテーブルを管理するためのベストプラクティスと要件

Amazon DynamoDB グローバルテーブルを使用すると、テーブルデータを AWS リージョン間でレプリケートできます。データが適切にレプリケートされるように、グローバルテーブル内のレプリカテーブルとセカンダリインデックスにも、書き込みキャパシティーが同じように設定されていることが重要です。

後でわかりやすくするために、いつかグローバルテーブルになる可能性のあるテーブルの名前にリージョンを入れない方が便利な場合があります。

> ⚠ Warning
>
> 各グローバルテーブルのテーブル名は、AWS アカウント内で一意である必要があります。

グローバルテーブルのバージョン

使用しているグローバルテーブルのバージョンを確認するには、使用しているグローバルテーブルのバージョンを確認する を参照してください。

キャパシティを管理するための要件

グローバルテーブルには、次の 2 つの方法のいずれかでスループットキャパシティが設定されている必要があります。

1. オンデマンドキャパシティモードは、レプリケートされた書き込み要求ユニット (rWRU) で測定されます

2. オートスケーリングでプロビジョニングされたキャパシティモードは、レプリケートされた書き込みキャパシティユニット (rWCU) で測定されます

オートスケーリングまたはオンデマンドキャパシティモードでプロビジョニングされたャパシティモードを使用すると、グローバルテーブルのすべてのリージョンにレプリケートされた書き込みを実行するのに十分なキャパシティが確保されます。

> **Note**
>
> いずれかのリージョンで 1 つのテーブルキャパシティモードから別のキャパシティモードに切り替えると、すべてのレプリカのモードが切り替わります。

グローバルテーブルのデプロイ

AWS CloudFormation では、各グローバルテーブルは、単一のリージョン内の単一のスタックによって制御されます。これは、レプリカの数に関係なく行われます。テンプレートをデプロイすると、CloudFormation は単一のスタックオペレーションの一部としてすべてのレプリカを作成/更新します。このため、同じ `AWS::DynamoDB::GlobalTable` リソースを複数のリージョンにデプロイしないでください。デプロイはサポートされておらず、エラーが発生します。

アプリケーションテンプレートを複数のリージョンにデプロイする場合は、条件を使用して 1 つのリージョンにのみリソースを作成できます。または、アプリケーションスタックとは別のスタック内に `AWS::DynamoDB::GlobalTable` リソースを定義し、単一リージョンにのみデプロイされることを確認してください。詳細については、「CloudFormation のグローバルテーブル」を参照してください。

DynamoDB テーブルは `AWS::DynamoDB::Table` によって参照され、グローバルテーブルは `AWS::DynamoDB::GlobalTable` になります。CloudFormation に関する限り、これは本質的に 2 つの異なるリソースになります。そのため、1 つの方法として、グローバルになる可能性のあるすべてのテーブルを `GlobalTable` コンストラクトを使用して作成する方法があります。その後、最初はスタンドアロンテーブルとして保持し、必要に応じて後でリージョンに追加できます。

通常のテーブルがあり、CloudFormation の使用中にそれを変換する場合は、以下の方法をおすすめします。

1. 削除ポリシーを保持するように設定します。
2. テーブルをスタックから削除します。

3. テーブルをコンソールでグローバルテーブルに変換します。
4. グローバルテーブルを新しいリソースとしてスタックにインポートします。

> **Note**
>
> クロスアカウントレプリケーションは現在サポートされていません。

グローバルテーブルを使用して、リージョンが停止する可能性への対処に役立てる

それぞれローカルの DynamoDB エンドポイントにアクセスする別のリージョンに、実行スタックの独立したコピーを所有、または迅速に作成できるようにします。

Route53 または AWS Global Accelerator を使用して、最も近い正常なリージョンにルーティングします。または、使用する可能性がある複数のエンドポイントをクライアントに認識させます。

各リージョンでヘルスチェックを行うと、DynamoDB のパフォーマンスが低下しているなど、スタックに問題があるかどうかを確実に判断できます。例えば、DynamoDB エンドポイントが稼働していることを、ping を実行するだけで確認しないでください。データベースフローが完全に成功するように、実際に呼び出しを行います。

ヘルスチェックに失敗した場合、トラフィックは他のリージョンにルーティングできます (DNS エントリを Route53 で更新するか、Global Accelerator のルートを変更するか、クライアントに別のエンドポイントを選択させます)。グローバルテーブルは、データが継続的に同期されるため RPO (復旧ポイント目標) が良好で、両方のリージョンで常に読み取りトラフィックと書き込みトラフィックの両方に対応できるため、RTO (目標復旧時間) も良好です。

ヘルスチェックの詳細については、「ヘルスチェックのタイプ」を参照してください。

> **Note**
>
> DynamoDB は、他のサービスが頻繁にコントロールプレーンオペレーションを構築するコアサービスであるため、DynamoDB がリージョン内のサービスを低下させているのに、他のサービスには影響がないというシナリオが発生することはほとんどありません。

グローバルテーブルのバックアップ

グローバルテーブルをバックアップする場合、1 つのリージョンのテーブルをバックアップするだけで十分です。すべてのリージョンのすべてのテーブルをバックアップする必要はありません。誤って削除または変更されたデータを回復することが目的であれば、1 つのリージョンの PITR で十分です。同様に、規制要件などの過去の目的でスナップショットを保存する場合は、1 つのリージョンにバックアップすれば十分です。バックアップされたデータは、AWS Backup を介して複数のリージョンにレプリケートできます。

レプリカと書き込みユニットの計算

計画を立てるには、あるリージョンが実行する書き込み数を取得し、それを他のリージョンごとに発生する書き込み数に加算する必要があります。1 つのリージョンで実行されるすべての書き込みは、すべてのレプリカリージョンでも実行する必要があるため、これは重要です。すべての書き込みを処理するのに十分なキャパシティがない場合、キャパシティの例外が発生します。さらに、リージョン間レプリケーションの待機時間が長くなります。

たとえば、オハイオ州のレプリカテーブルに 1 秒間に 5 回の書き込みを、バージニア北部のレプリカテーブルに 1 秒間に 10 回の書き込みを、アイルランドのレプリカテーブルに 1 秒あたり 5 回の書き込みを行うとします。この場合、オハイオ州、バージニア州北部、アイルランドの各リージョンで 20 rWCU または rWCU を消費すると予想されます。つまり、3 つのリージョンすべてで合計 60 rWCU を消費すると予想されます。

オートスケーリングによるプロビジョニングされたキャパシティと DynamoDB の詳細については、「DynamoDB Auto Scaling によるスループットキャパシティの自動管理」を参照してください。

> **Note**
>
> テーブルが自動スケーリングでプロビジョニングされたキャパシティーモードで実行されている場合、プロビジョニングされた書き込みキャパシティーは、各リージョンの自動スケーリング設定の範囲内で変動させることができます。

チュートリアル: グローバルテーブルの作成

このセクションでは、Amazon DynamoDB コンソールまたは AWS Command Line Interface (AWS CLI) を使用してグローバルテーブルを作成する方法について説明します。

トピック

- グローバルテーブルの作成 (コンソール)
- グローバルテーブル (AWS CLI) の作成
- グローバルテーブル (Java) の作成

グローバルテーブルの作成 (コンソール)

コンソールを使用してグローバルテーブルを作成するには、次の手順に従います。以下の例では、レプリカテーブルを持つグローバルテーブルを米国およびヨーロッパに作成します。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/home) を開きます。この例では、米国東部 (オハイオ) リージョンを選択します。
2. コンソールの左側のナビゲーションペインで、[Tables] (テーブル) を選択します。
3. [テーブルの作成] を選択します。

    a. [テーブル名] に「`Music`」と入力します。

    b. [パーティションキー] に「`Artist`」と入力します。[ソートキー] に「`SongTitle`」と入力します (`Artist` と `SongTitle` はいずれも文字列)。

    テーブルを作成するには、[テーブルを作成] を選択します。このテーブルは、新しいグローバルテーブルの最初のレプリカテーブルとして機能します。これは、後で追加する他のレプリカテーブルのプロトタイプです。
4. [グローバルテーブル] タブを選択後、[レプリカを作成] を選択します。
5. [利用できるレプリケーションリージョン] ドロップダウンで、米国西部 (オレゴン) を選択します。

    コンソールは、選択したリージョンに同一名のテーブルが存在しないことを確認します。同一名のテーブルが存在する場合は、そのリージョンで新しいレプリカテーブルを作成する前に既存のテーブルを削除する必要があります。
6. [レプリカを作成] を選択します。これにより、米国西部 (オレゴン) でテーブル作成プロセスが開始されます。

    選択したテーブル (およびその他すべてのレプリカテーブル) の [グローバルテーブル] タブに、そのテーブルが複数のリージョンでレプリケートされたことが示されます。
7. ここで、別のリージョンを追加して、グローバルテーブルがレプリケートされ、米国およびヨーロッパに同期されるようにします。これを行うには、ステップ 5 を繰り返しますが、今回は米国西部 (オレゴン) の代わりに欧州 (フランクフルト) を指定します。

8. 引き続き、米国東部 (オハイオ)リージョンで AWS Management Console を使用する必要があります。左のナビゲーションメニューで [項目] を選択し、[音楽] テーブルを選択してから、[項目を作成] を選択します。

   a. [Artist] に「**`item_1`**」と入力します。
   b. [SongTitle] に「**`Song Value 1`**」と入力します。
   c. 項目を書き込むには、[項目を作成] を選択します。

9. 間もなくすると、この項目は、グローバルテーブルの 3 つのすべてのリージョンにレプリケートされます。これを確認するには、コンソールの右上隅にあるリージョンセレクターで、[Europe (Frankfurt) (欧州 (フランクフルト))] を選択します。欧州 (フランクフルト) の `Music` テーブルに新しい項目が追加されます。
10. ステップ 9 を繰り返し、米国西部 (オレゴン) を選択して、そのリージョンでのレプリケーションを確認します。

グローバルテーブル (AWS CLI) の作成

`Music` を使用してグローバルテーブル AWS CLI を作成するには、次の手順に従います。以下の例では、米国およびヨーロッパのレプリカテーブルを使用して、グローバルテーブルを作成します。

1. DynamoDB Streams を有効にして (`Music`)、米国東部 (オハイオ) で新規テーブル (`NEW_AND_OLD_IMAGES`) DynamoDB Streams を作成します。

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. 米国東部 (バージニア北部) に同じ `Music` テーブルを作成します。

```
aws dynamodb update-table --table-name Music --cli-input-json  \
'{
  "ReplicaUpdates":
```

```
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region=us-east-2
```

3. ステップ 2 を繰り返して、欧州 (アイルランド) (eu-west-1) にテーブルを作成します。
4. `describe-table` を使用して、作成されたレプリカのリストを表示できます。

```
aws dynamodb describe-table --table-name Music --region us-east-2
```

5. レプリケーションが機能していることを確認するには、新しい項目を米国東部 (オハイオ) の `Music` テーブルに追加します。

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. 数秒間待ってから、項目が米国東部 (バージニア北部) および欧州 (アイルランド) に正常にレプリケートされたかどうかを確認します。

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1
```

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

7. 欧州 (アイルランド) リージョンに作成されたレプリカテーブルを削除します。

```
aws dynamodb update-table --table-name Music --cli-input-json \
'{
  "ReplicaUpdates":
```

```
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}'
```

グローバルテーブル (Java) の作成

次の Java コード例は、欧州 (アイルランド) リージョンに `Music` テーブルを作成し、アジアパシフィック (ソウル) リージョンにレプリカを作成します。

```
package com.amazonaws.codesamples.gtv2
import java.util.logging.Logger;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AmazonDynamoDBException;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.BillingMode;
import com.amazonaws.services.dynamodbv2.model.CreateReplicationGroupMemberAction;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableRequest;
import com.amazonaws.services.dynamodbv2.model.GlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputOverride;
import com.amazonaws.services.dynamodbv2.model.ReplicaGlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.ReplicationGroupUpdate;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.model.UpdateTableRequest;
import com.amazonaws.waiters.WaiterParameters;
```

```
public class App
{
    private final static Logger LOGGER = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);

    public static void main( String[] args )
    {

        String tableName = "Music";
        String indexName = "index1";

        Regions calledRegion = Regions.EU_WEST_1;
        Regions destRegion = Regions.AP_NORTHEAST_2;

        AmazonDynamoDB ddbClient = AmazonDynamoDBClientBuilder.standard()
                .withCredentials(new ProfileCredentialsProvider("default"))
                .withRegion(calledRegion)
                .build();


        LOGGER.info("Creating a regional table - TableName: " + tableName +",
 IndexName: " + indexName + " .....");
        ddbClient.createTable(new CreateTableRequest()
                .withTableName(tableName)
                .withAttributeDefinitions(
                        new AttributeDefinition()

 .withAttributeName("Artist").withAttributeType(ScalarAttributeType.S),
                        new AttributeDefinition()

 .withAttributeName("SongTitle").withAttributeType(ScalarAttributeType.S))
                .withKeySchema(
                        new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH),
                        new
 KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE))
                .withBillingMode(BillingMode.PAY_PER_REQUEST)
                .withGlobalSecondaryIndexes(new GlobalSecondaryIndex()
                        .withIndexName(indexName)
                        .withKeySchema(new KeySchemaElement()
                                .withAttributeName("SongTitle")
                                .withKeyType(KeyType.HASH))
                        .withProjection(new
 Projection().withProjectionType(ProjectionType.ALL)))
                .withStreamSpecification(new StreamSpecification()
                        .withStreamEnabled(true)
                        .withStreamViewType(StreamViewType.NEW_AND_OLD_IMAGES)));

        LOGGER.info("Waiting for ACTIVE table status .....");
        ddbClient.waiters().tableExists().run(new WaiterParameters<>(new
 DescribeTableRequest(tableName)));


        LOGGER.info("Testing parameters for adding a new Replica in " + destRegion +
 " .....");

        CreateReplicationGroupMemberAction createReplicaAction = new
 CreateReplicationGroupMemberAction()
                .withRegionName(destRegion.getName())
                .withGlobalSecondaryIndexes(new ReplicaGlobalSecondaryIndex()
                        .withIndexName(indexName)
                        .withProvisionedThroughputOverride(new
 ProvisionedThroughputOverride()
                                .withReadCapacityUnits(15L)));


        ddbClient.updateTable(new UpdateTableRequest()
                .withTableName(tableName)
                .withReplicaUpdates(new ReplicationGroupUpdate()
                        .withCreate(createReplicaAction.withKMSMasterKeyId(null))));





    }
}
```

グローバルテーブルのモニタリング

Amazon CloudWatch を使用して、グローバルテーブルの動作とパフォーマンスをモニタリングできます。Amazon DynamoDB は、グローバルテーブルのレプリカごとに `ReplicationLatency` メトリクスを公開します。

- **`ReplicationLatency`** - 項目がレプリカテーブルに書き込まれてから、その項目がグローバルテーブルの別のレプリカに表示されるまでの経過時間。`ReplicationLatency` はミリ秒単位で表し、すべての送信元リージョンと送信先リージョンのペアに対して出力されます。

通常オペレーション中、ReplicationLatency は完全に一定にする必要があります。ReplicationLatency で評価された値は、1 つのレプリカを更新しても、他のレプリカテーブルにタイムリーに伝達されません。これが原因で、アップデートは一貫して受け取ることができないため、時間の経過とともに、他のレプリカテーブルは遅延します。この場合、各レプリカテーブルの読み込みキャパシティーユニット (RCU) と書き込みキャパシティーユニット (WCU) が同一であることを確認する必要があります。また、WCU 設定を選択する場合は、「グローバルテーブルのバージョン」の推奨事項に従います。

ReplicationLatency は、AWS リージョンのパフォーマンスが低下し、そのリージョンにレプリカテーブルが含まれる場合は増加することがあります。この場合、アプリケーションの読み込みおよび書き込みアクティビティを別の AWS リージョンに一時的にリダイレクトすることができます。

詳細については、「DynamoDB のメトリクスとディメンション」を参照してください。

グローバルテーブルで IAM を使用します

グローバルテーブルを初めて作成する場合は、Amazon DynamoDB によって、AWS Identity and Access Management (IAM) サービスにリンクされたロールが自動的に作成されます。このロールの名前は AWSServiceRoleForDynamoDBReplication です。このロールを使用して、お客様に代わって DynamoDB でグローバルテーブルのクロスリージョンレプリケーションを管理できます。このサービスにリンクされたロールは削除しないでください。削除すると、グローバルテーブルはすべて使用できなくなります。

サービスリンクロールの詳細については、「IAM ユーザーガイド」の「サービスリンクロールの使用」を参照してください。

DynamoDB でレプリカテーブルを作成するには、送信元リージョンで、次の許可が必要です。

- dynamodb:UpdateTable

DynamoDB でレプリカテーブルを作成するには、送信先リージョンで、次の許可が必要です。

- dynamodb:CreateTable
- dynamodb:CreateTableReplica
- dynamodb:Scan
- dynamodb:Query

- dynamodb:UpdateItem
- dynamodb:PutItem
- dynamodb:GetItem
- dynamodb:DeleteItem
- dynamodb:BatchWriteItem

DynamoDB でレプリカテーブルを削除するには、送信先リージョンで、次の許可が必要です。

- dynamodb:DeleteTable
- dynamodb:DeleteTableReplica

UpdateTableReplicaAutoScaling で自動スケーリングポリシーを更新するには、テーブルレプリカが存在するすべてのリージョンで、次のアクセス許可が必要です。

- application-autoscaling:DeleteScalingPolicy
- application-autoscaling:DeleteScheduledAction
- application-autoscaling:DeregisterScalableTarget
- application-autoscaling:DescribeScalableTargets
- application-autoscaling:DescribeScalingActivities
- application-autoscaling:DescribeScalingPolicies
- application-autoscaling:DescribeScheduledActions
- application-autoscaling:PutScalingPolicy
- application-autoscaling:PutScheduledAction
- application-autoscaling:RegisterScalableTarget

UpdateTimeToLive を使用するには、テーブルレプリカが存在するすべてのリージョンで、dynamodb:UpdateTimeToLive のアクセス許可が必要です。

例: レプリカの追加

次の IAM ポリシーは、グローバルテーブルにレプリカを追加するアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:CreateTableReplica",
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*"
        }
    ]
}
```

例: Auto Scaling ポリシーの更新

次の IAM ポリシーは、レプリカの自動スケーリングポリシーを更新するアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:RegisterScalableTarget",
                "application-autoscaling:DeleteScheduledAction",
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:DescribeScheduledActions",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:PutScheduledAction",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

例: 特定のテーブル名およびリージョンのレプリカ作成を許可します

次の IAM ポリシーは、3 つのリージョンで、レプリカを使用して、`Customers` テーブルのテーブルおよびレプリカの作成を許可するアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable"
            ],

            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers",
                "arn:aws:dynamodb:eu-east-2:123456789012:table/Customers"
            ]
        }
    ]
}
```

使用しているグローバルテーブルのバージョンを確認する

DynamoDB グローバルテーブルには、グローバルテーブルバージョン 2019.11.21 (現行) と グローバルテーブルバージョン 2017.11.29 (レガシー) の 2 つのバージョンがあります。グローバルテーブルバージョン 2019.11.21 (現行) を使用することをおすすめします。グローバルテーブルバージョン 2017.11.29 (レガシー) よりも効率的で、消費される書き込みキャパシティーが少なくなります。現行バージョンの利点は次のとおりです。

- ソーステーブルとターゲットテーブルはまとめて管理され、スループット、TTL 設定、自動スケーリング設定、その他の有益な属性について自動的に調整されます。
- グローバルセカンダリインデックスも調整されます。
- データが入力されたテーブルから新しいレプリカテーブルを動的に追加できます。

- レプリケーションの管理に必要なメタデータ属性は非表示になっているため、レプリケーションで問題が発生する原因となる、その書き込みの防止に役立ちます。
- 現行バージョンは、レガシーバージョンよりも多くのリージョンをサポートしており、既存のテーブルでリージョンを追加または削除できます。これはレガシーバージョンではサポートされていません。
- グローバルテーブルバージョン 2019.11.21 (現行) は、グローバルテーブルバージョン 2017.11.29 (レガシー) よりも効率的で，消費する書き込みキャパシティが少ないため、費用対効果が高くなります。具体的な説明:
  - あるリージョンに新しい項目を挿入してから他のリージョンにレプリケートするには、バージョン 2017.11.29 (レガシー) ではリージョンごとに 2 つの rWCU が必要ですが、バージョン 2019.11.21 (現行) では 1 つの rWCU しか必要ありません。
  - 項目を更新するには、バージョン 2017.11.29 (レガシー) ではソースリージョンに 2 つの rWCU、レプリケーション先リージョンごとに 1 つの rWCU が必要ですが、バージョン 2019.11.21 (現行) では、ソースまたはレプリケーション先ごとに 1 つの rWCU しか必要ありません。
  - 項目を削除するには、バージョン 2017.11.29 (レガシー) ではソースリージョンに 1 つの rWCU、レプリケーション先リージョンごとに 2 つの rWCU が必要ですが、バージョン 2019.11.21 (現行) では、ソースまたはレプリケーション先ごとに 1 つの rWCU しか必要ありません。

  詳細については、「Amazon DynamoDB の料金表」を参照してください。

CLI を通じたバージョンの確認

AWS CLI を通じて、使用しているグローバルテーブルのバージョンを確認するには、`DescribeTable` と `DescribeGlobalTable` をチェックします。`DescribeTable` では、バージョン 2019.11.21 (最新) の場合はテーブルバージョンが表示され、バージョン 2017.11.29 (レガシー) の場合は `DescribeGlobalTable` プロパティにテーブルバージョンが表示されます。

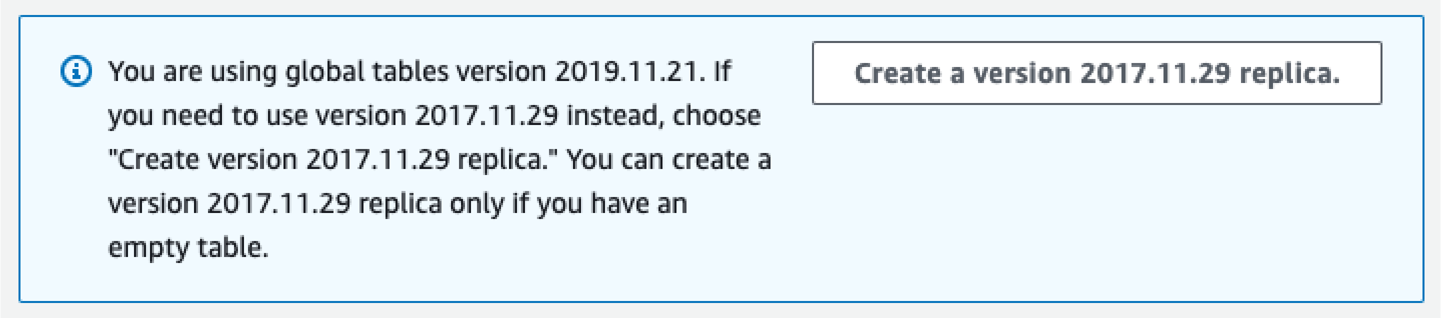
コンソールを通じたバージョンの確認

コンソールを通じたバージョンの確認

コンソールを通じて使用しているグローバルテーブルのバージョンを確認するには、以下を実行します。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/home) を開きます。

2. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
3. 使用するテーブルを選択します。
4. [グローバルテーブル] タブを選択します。
5. [グローバルテーブルのバージョン] に、使用しているグローバルテーブルのバージョンが表示されます。



You are using global tables version 2019.11.21. If you need to use version 2017.11.29 instead, choose "Create version 2017.11.29 replica." You can create a version 2017.11.29 replica only if you have an empty table.

バージョン 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) へのグローバルテーブルのアップグレードについては、こちらを参照してください。アップグレードプロセス全体は、ライブテーブルの中断なしで、1 時間以内に完了します。詳細については、「2019.11.21 (現行) への更新」を参照してください。

Note

- [グローバルテーブルバージョン] メッセージがコンソールに表示されない場合は、別のリージョンに同じ名前のテーブルがもう 1 つあることを意味します。この場合、現在のテーブルをグローバルテーブルにすることはできません。現在のテーブルを一意の名前の新しいテーブルにコピーするか、同じ名前の他のすべてのテーブルを削除する必要があります。
- グローバルテーブルバージョン 2019.11.21 (現行) を使用しており、Time to Live 機能も使用している場合、DynamoDB は TTL による削除をすべてのレプリカテーブルにレプリケートします。最初の TTL 削除では、TTL 有効期限が発生するリージョンでの書き込みキャパシティーは消費されません。ただし、レプリカテーブルにレプリケートされた TTL 削除では、プロビジョンドキャパシティーを使用する場合は、レプリケートされた書き込みキャパシティーユニットが消費され、オンデマンドのキャパシティーモードを使用する場合は、レプリケートされた書き込みが各レプリカリージョンで消費されます。該当する料金が適用されます。
- グローバルテーブルバージョン 2019.11.21 (現行) では、TTL 削除が発生すると、すべてのレプリカリージョンにレプリケートされます。これらのレプリケートされた書き込みに

は、type または principalID プロパティが含まれません。これにより、レプリケートされたテーブル内でユーザー削除と TTL 削除を区別することが難しくなります。

レガシーバージョン (2017.11.29) から現行バージョン (2019.11.21) へのアップグレード

DynamoDB グローバルテーブルには、グローバルテーブルバージョン 2019.11.21 (現行) と グローバルテーブルバージョン 2017.11.29 (レガシー) の 2 つのバージョンがあります。可能な限り、2019.11.21 (現行) バージョンを使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティーが少ないためです。使用中のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。

このセクションでは、DynamoDB コンソールを使用して、グローバルテーブルをバージョン 2019.11.21 (現行) にアップグレードする方法について説明します。バージョン 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) へのアップグレードは 1 回限りのアクションであり、元に戻すことはできません。現在、グローバルテーブルはコンソールからのみアップグレードできます。

トピック

レガシーバージョンと現行バージョンの動作の違い

以下のリストは、グローバルテーブルのレガシーバージョンと現行バージョンの動作の違いを示しています。

- 複数の DynamoDB オペレーションにおいて、現行バージョン 2019.11.21 は、レガシーバージョン 2017.11.29 よりも書き込みキャパシティの消費が少ないため、ほとんどのお客様のコスト効率が改善します。DynamoDB オペレーションの違いは次のとおりです。

- あるリージョンの 1KB アイテムに対して PutItem を呼び出し、他のリージョンにレプリケートする場合、2017.11.29 (レガシー) ではリージョンごとに 2 rWRU が必要ですが、2019.11.21 (現行) では 1 rWRU のみ必要です。
- 1KB アイテムに対して UpdateItem を呼び出す場合、2017.11.29 (レガシー) ではソースリージョンで 2 rWRU、送信先リージョンで 1 rWRU が必要ですが、2019.11.21 (現行) ではソースリージョンおよび送信先リージョンでそれぞれ 1 rWRU のみ必要です。
- 1KB アイテムに対して DeleteItem を呼び出す場合、2017.11.29 (レガシー) ではソースリージョンで 1 rWRU、送信先リージョンで 2 rWRU が必要ですが、2019.11.21 (現行) ではソースリージョンおよび送信先リージョンでそれぞれ 1 rWRU のみ必要です。

次の表は、2017.11.29 (レガシー) テーブルと 2019.11.21 (現行) テーブルの rWRU 消費量を示しています。

2 つのリージョンの 1KB アイテムの 2017.11.29 (レガシー) テーブルおよび 2019.11.21 (現行) テーブルの rWRU 消費量

操作	2017.11.29 (レガシー)	2019.11.21 (現行)	削減量
PutItem	4 rWRU	2 rWRU	50%
UpdateItem	3 rWRU	2 rWRU	33%
DeleteItem	3 rWRU	2 rWRU	33%

- バージョン 2017.11.29 (レガシー) は、11 の AWS リージョン でのみ利用可能です。ただし、バージョン 2019.11.21 (現行) は、すべての AWS リージョン で使用できます。
- バージョン 2017.11.29 (レガシー) グローバルテーブルを作成するには、まず空のリージョンテーブルのセットを作成し、次に CreateGlobalTable API を呼び出してグローバルテーブルを作成します。バージョン 2019.11.21 (現行) グローバルテーブルを作成するには、UpdateTable API を呼び出して、既存のリージョンテーブルにレプリカを追加します。
- バージョン 2017.11.29 (レガシー) では、新しいリージョンにレプリカを追加する前に、テーブル内のすべてのレプリカを空にする必要があります (作成時を含む)。バージョン 2019.11.21 (現行) では、すでにデータが存在するテーブルのリージョンにレプリカを追加および削除できます。
- バージョン 2017.11.29 (レガシー) は、レプリカを管理するために次の専用コントロールプレーン API セットを使用します。
  - CreateGlobalTable

  - DescribeGlobalTable
  - DescribeGlobalTableSettings
  - ListGlobalTables
  - UpdateGlobalTable
  - UpdateGlobalTableSettings

  バージョン 2019.11.21 (現行) は、DescribeTable API および UpdateTable API を使用してレプリカを管理します。

- バージョン 2017.11.29 (レガシー) は、1 回の書き込みで 2 つの DynamoDB Streams レコードを公開します。バージョン 2019.11.21 (現行) は、1 回の書き込みで 1 つの DynamoDB Streams レコードのみを公開します。
- バージョン 2017.11.29 (レガシー) は、`aws:rep:deleting`、`aws:rep:updateregion`、および `aws:rep:updatetime` 属性の入力と更新を行います。バージョン 2019.11.21 (現行) は、これらの属性の入力または更新を行いません。
- バージョン 2017.11.29 (レガシー) は、レプリカ間で Time to Live (TTL) 設定の同期を行いません。バージョン 2019.11.21 (現行) は、レプリカ間で TTL 設定を同期します。
- バージョン 2017.11.29 (レガシー) は、TTL の削除を他のレプリカにレプリケートしません。バージョン 2019.11.21 (現行) は、TTL の削除をすべてのレプリカにレプリケートします。
- バージョン 2017.11.29 (レガシー) は、レプリカ間で自動スケーリング設定の同期を行いません。バージョン 2019.11.21 (現行) は、レプリカ間で自動スケーリング設定を同期します。
- バージョン 2017.11.29 (レガシー) は、レプリカ間でグローバルセカンダリインデックス (GSI) 設定の同期を行いません。バージョン 2019.11.21 (現行) は、レプリカ間で GSI 設定を同期します。
- バージョン 2017.11.29 (レガシー) は、レプリカ間で保管時の暗号化設定の同期を行いません。バージョン 2019.11.21 (現行) は、レプリカ間で保管時の暗号化設定を同期します。
- バージョン 2017.11.29 (レガシー) は、`PendingReplicationCount` メトリクスを公開します。バージョン 2019.11.21 (現行) は、このメトリクスを公開しません。

アップグレードの前提条件

バージョン 2019.11.21 (現行) グローバルテーブルへのアップグレードを開始する前に、次の前提条件を満たす必要があります。

- レプリカの Time to Live (TTL) 設定がリージョン間で一貫していること。
- レプリカのグローバルセカンダリインデックス (GSI) 定義がリージョン間で一貫していること。

- レプリカの保管時の暗号化設定がリージョン間で一貫していること。
- すべてのレプリカの WCU で DynamoDB 自動スケーリングが有効になっていること、またはすべてのレプリカでオンデマンドキャパシティモードが有効になっていること。
- アプリケーションで、テーブルアイテムの aws:rep:deleting、aws:rep:updateregion、aws:rep:updatetime 属性が必須でないこと。

グローバルテーブルのアップグレードに必要なアクセス許可

バージョン 2019.11.21 (現行) にアップグレードするには、レプリカのあるすべてのリージョンで dynamodb:UpdateGlobalTableVersion アクセス許可が必要です。DynamoDB コンソールへのアクセス許可、およびテーブル表示に必要なアクセス許可に加えて、このアクセス許可が必要です。

次の IAM ポリシーは、グローバルテーブルをバージョン 2019.11.21 (現行) にアップグレードするアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableVersion",
            "Resource": "*"
        }
    ]
}
```

次の IAM ポリシーは、2 つのリージョンにレプリカがある Music グローバルテーブルのみをバージョン 2019.11.21 (現行) にアップグレードするアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableVersion",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Music",
                "arn:aws:dynamodb:ap-southeast-1:123456789012:table/Music",
                "arn:aws:dynamodb:us-east-2:123456789012:table/Music"
                ]
            }
        ]
}
```

アップグレード中の注意点

- すべてのグローバルテーブルレプリカは、アップグレード中も読み取りと書き込みのトラフィックを処理します。
- テーブルのサイズとレプリカの数にもよりますが、アップグレードプロセスには、数分から数時間かかります。
- アップグレードプロセス中、TableStatus の値は ACTIVE から UPDATING に変わります。テーブルのステータスを表示するには、DescribeTable API を呼び出すか、DynamoDB コンソールのテーブルビューを使用します。
- テーブルのアップグレード中、自動スケーリングはグローバルテーブルのプロビジョニング済のキャパシティ設定を調整しません。アップグレード中は、テーブルをオンデマンドキャパシティモードに設定することを強くお勧めします。
- アップグレード中に自動スケーリングでプロビジョニングされた容量を使用する場合は、アップグレード中に予想されるトラフィックの増加に対応しスロットリングを回避するために、ポリシーの最小読み取りスループットと最小書き込みスループットを増やす必要があります。
- アップグレードプロセスが完了すると、テーブルのステータスは ACTIVE に変わります。

アップグレード前、アップグレード中、アップグレード後の DynamoDB Streams の動作

操作	レプリカリージョン	アップグレード前の動作	アップグレード中の動作	アップグレード後の動作
Put または Update	ソース	タイムスタンプの生成は UpdateItem を使用して行われます。	タイムスタンプの生成は PutItem を使用して行われます。	カスタマー表示タイムスタンプは生成されません。
		2 つの Streams レコードが生	2 つの Streams レコードが生	お客様が作成した属性を含む 1

操作	レプリカリージョン	アップグレード前の動作	アップグレード中の動作	アップグレード後の動作
		成されます。最初のレコードには、お客様が書き込んだ属性が含まれます。2 つ目のレコードには、`aws:rep:*` 属性が含まれます。	成されます。最初のレコードには、お客様が書き込んだ属性が含まれます。2 つ目のレコードには、`aws:rep:*` 属性が含まれます。	つの Streams レコードが生成されます。
		お客様による書き込みごとに 2 つの rWCU が消費されます。	お客様による書き込みごとに 2 つの rWCU が消費されます。	お客様による書き込みごとに 1 つの rWCU が消費されます。
		`ReplicationLatency` メトリクスおよび `PendingReplicationCount` メトリクスは、CloudWatch に公開されます。	`ReplicationLatency` メトリクスおよび `PendingReplicationCount` メトリクスは、CloudWatch に公開されます。	`ReplicationLatency` メトリクスは、CloudWatch に公開されます。
	送信先	レプリケーションは PutItem を使用して行われます。	レプリケーションは PutItem を使用して行われます。	レプリケーションは PutItem を使用して行われます。

操作	レプリカリージョン	アップグレード前の動作	アップグレード中の動作	アップグレード後の動作
		お客様が作成した属性と aws:rep:* 属性の両方を含む 1 つの Streams レコードが生成されます。	お客様が作成した属性と aws:rep:* 属性の両方を含む 1 つの Streams レコードが生成されます。	お客様が作成した属性のみを含みレプリケーション属性を含まない 1 つの Streams レコードが生成されます。
		アイテムが送信先のリージョンに存在する場合、1 rWCU が消費されます。アイテムが送信先のリージョンに存在しない場合、2 rWCU が消費されます。	アイテムが送信先のリージョンに存在する場合、1 rWCU が消費されます。アイテムが送信先のリージョンに存在しない場合、2 rWCU が消費されます。	お客様による書き込みごとに 1 つの rWCU が消費されます。
		ReplicationLatency メトリクスおよび PendingReplicationCount メトリクスは、CloudWatch に公開されます。	ReplicationLatency メトリクスおよび PendingReplicationCount メトリクスは、CloudWatch に公開されます。	ReplicationLatency メトリクスは、CloudWatch に公開されます。

操作	レプリカリージョン	アップグレード前の動作	アップグレード中の動作	アップグレード後の動作
[Delete] (削除)	ソース	[DeleteItem](#) を使用して、より小さいタイムスタンプを持つすべてのアイテムを削除します。	DeleteItem を使用して、より小さいタイムスタンプを持つすべてのアイテムを削除します。	DeleteItem を使用して、より小さいタイムスタンプを持つすべてのアイテムを削除します。
		お客様が作成した属性と `aws:rep:*` 属性の両方を含む 1 つの Streams レコードが生成されます。	お客様が作成した属性と `aws:rep:*` 属性の両方を含む 1 つの Streams レコードが生成されます。	お客様が作成した属性を含む 1 つの Streams レコードが生成されます。
		お客様による削除ごとに 1 rWCU が消費されます。	お客様による削除ごとに 1 rWCU が消費されます。	お客様による削除ごとに 1 rWCU が消費されます。
		`ReplicationLatency` メトリクスおよび `PendingReplicationCount` メトリクスは、CloudWatch に公開されます。	`ReplicationLatency` メトリクスおよび `PendingReplicationCount` メトリクスは、CloudWatch に公開されます。	`ReplicationLatency` メトリクスは、CloudWatch に公開されます。

操作	レプリカリージョン	アップグレード前の動作	アップグレード中の動作	アップグレード後の動作
	送信先	2 つのフェーズでの削除が行われます。 - フェーズ 1 では、UpdateItem によって削除フラグが設定されます。 - フェーズ 2 では、DeleteItem によってアイテムが削除されます。	DeleteItem を使用してアイテムを削除します。	DeleteItem を使用してアイテムを削除します。
		2 つの Streams レコードが生成されます。1 つ目のレコードには、`aws:rep:deleting` フィールドへの変更が含まれます。2 つ目のレコードには、お客様が作成した属性と `aws:rep:*` 属性が含まれます。	お客様が作成した属性を含む 1 つの Stream レコードが生成されます。	お客様が作成した属性を含む 1 つの Stream レコードが生成されます。

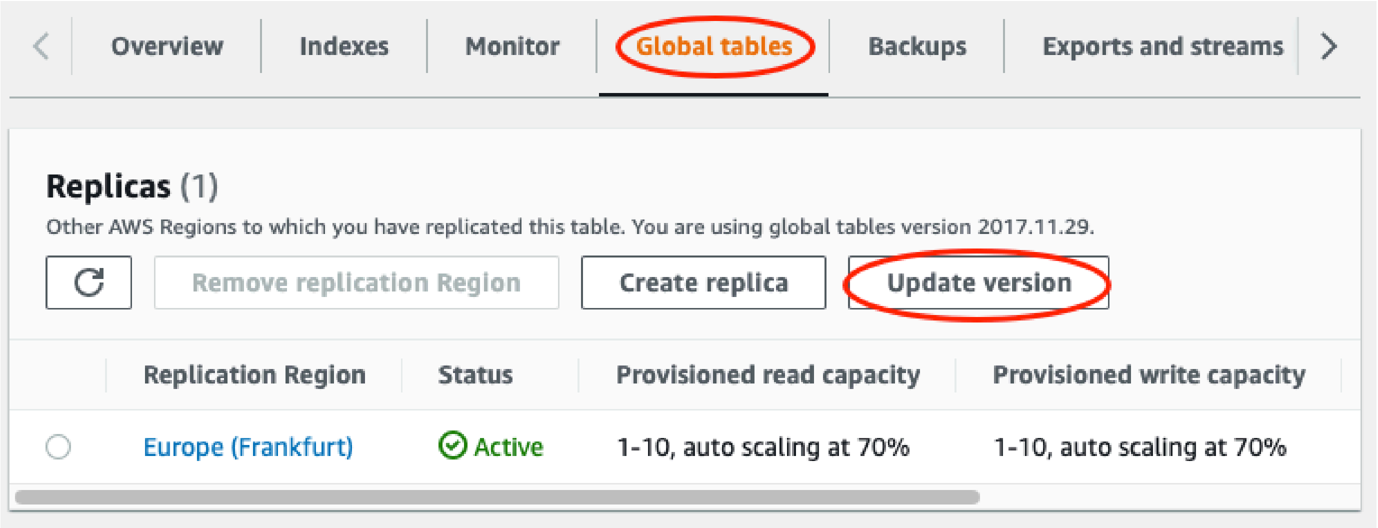
操作	レプリカリージョン	アップグレード前の動作	アップグレード中の動作	アップグレード後の動作
		お客様による削除ごとに 2 つの rWCU が消費されます。	お客様による削除ごとに 1 rWCU が消費されます。	お客様による削除ごとに 1 rWCU が消費されます。
		`ReplicationLatency` メトリクスおよび `PendingReplicationCount` メトリクスは、CloudWatch に公開されます。	`ReplicationLatency` メトリクスは、CloudWatch に公開されます。	`ReplicationLatency` メトリクスは、CloudWatch に公開されます。

バージョン 2019.11.21 (現行) へのアップグレード

AWS Management Console を使用して DynamoDB グローバルテーブルのバージョンをアップグレードするには、次の手順に従います。

グローバルテーブルをバージョン 2019.11.21 (現行) にアップグレードするには

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/home) を開きます。
2. コンソールの左側のナビゲーションペインで、[テーブル] を選択し、バージョン 2019.11.21 (現行) にアップグレードするグローバルテーブルを選択します。
3. [グローバルテーブル] タブを選択します。
4. [Update version (バージョンの更新)] を選択します。



Replicas (1)

Other AWS Regions to which you have replicated this table. You are using global tables version 2017.11.29.

	Replication Region	Status	Provisioned read capacity	Provisioned write capacity
○	Europe (Frankfurt)	✓ Active	1-10, auto scaling at 70%	1-10, auto scaling at 70%

5. 新しい要件を読んで同意してから、[バージョンを更新] を選択します。

6. アップグレードプロセスが完了すると、コンソールに表示されるグローバルテーブルのバージョンが 2019.11.21 に変更されます。

読み込み操作と書き込み操作の使用

読み込み操作と書き込み操作は、DynamoDB API または DynamoDB 用 PartiQL のいずれかを使用して実行できます。これらの操作により、テーブル内の項目を操作して、作成、読み込み、更新、および削除 (CRUD) の基本的な機能を実行できます。

次のセクションでは、このトピックについてさらに詳しく説明します。

トピック

- DynamoDB API
- PartiQL: Amazon DynamoDB 用の SQL 互換クエリ言語

DynamoDB API

トピック

- 項目と属性の操作
- 項目コレクション - DynamoDB で一対多リレーションシップをモデル化する方法
- DynamoDB でのスキャンの使用

項目と属性の操作

Amazon DynamoDB では、項目は属性の集まりです。各属性には名前と値があります。属性値はスカラー型、セット型、ドキュメント型のいずれかです。詳細については、「Amazon DynamoDB: 仕組み」を参照してください。

DynamoDB では、作成、読み込み、更新、および削除 (CRUD) の 4 つの基本的な操作機能を使用できます。これらの操作はすべてアトミックです。

- `PutItem` — 項目を作成します。
- `GetItem` — 項目を読み込みます。
- `UpdateItem` — 項目を更新します。
- `DeleteItem` — 項目を削除します。

これらの各オペレーションでは、作業対象の項目のプライマリキーを指定する必要があります。たとえば、`GetItem` を使用して項目を読み込むには、その項目のパーティションキーとソートキー (該当する場合) を指定する必要があります。

4 つの基本的な CRUD オペレーションに加えて、DynamoDB は以下も提供します。

- `BatchGetItem` — 1 つ以上のテーブルから最大 100 個の項目を読み込みます。
- `BatchWriteItem` — 1 つ以上のテーブルから最大 25 個の項目を作成または削除します。

これらのバッチ操作は、複数の CRUD オペレーションを単一のリクエストにまとめます。さらに、応答のレイテンシーを最小限に抑えるため、バッチオペレーションは項目を並列で読み書きします。

このセクションには、これらのオペレーションを使用する方法の説明、および、条件付き更新やアトミックカウンターなどの関連するトピックが含まれています。このセクションには、AWS SDK を使用するサンプルコードも含まれています。

トピック

- 条件付きの書き込み
- DynamoDB での式の使用
- Time to Live (TTL)
- 項目の操作: Java
- 項目の操作: .NET

項目の読み込み

DynamoDB テーブルから項目を読み込むには、`GetItem` オペレーションを使用します。必要な項目のプライマリキーと共にテーブルの名前を指定する必要があります。

Example

次の AWS CLI の例は、`ProductCatalog` テーブルから項目を読み込む方法を示しています。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> `GetItem` では、プライマリキーの一部だけではなく全体を指定する必要があります。たとえば、テーブルに複合プライマリキー (パーティションキーおよびソートキー) がある場合、パーティションキーおよびソートキーの値を指定する必要があります。

デフォルトでは、`GetItem` リクエストは結果的に整合性のある読み込みを行います。代わりに、`ConsistentRead` パラメータを使用して、強い整合性のある読み込みをリクエストできます。(これは、追加の読み込みキャパシティーユニットを消費しますが、項目の最新バージョンを返します)。

`GetItem` は、項目のすべての属性を返します。一部の属性のみが返されるように、プロジェクション式を使用できます。詳細については、「プロジェクション式」を参照してください。

`GetItem` で消費される読み込みキャパシティーユニットの数を返すには、`ReturnConsumedCapacity` パラメータを `TOTAL` に設定します。

Example

次の AWS Command Line Interface (AWS CLI) の例は、オプションの GetItem パラメータを示しています。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

項目を書き込みます

DynamoDB テーブルの項目を作成、更新、または削除するには、次のオペレーションのいずれかを使用します。

- PutItem
- UpdateItem
- DeleteItem

これらの各オペレーションで、プライマリキーの一部の属性ではなくすべての属性を指定する必要があります。たとえば、テーブルに複合プライマリキー (パーティションキーおよびソートキー) がある場合、パーティションキーおよびソートキーの値を提供する必要があります。

これらのオペレーションのいずれかで消費される書き込みキャパシティーユニットの数を返すには、ReturnConsumedCapacity パラメータを次のいずれかに設定します。

- TOTAL — 消費された書き込み容量単位の総数を返します。
- INDEXES — 消費された書き込み容量単位の総数とともに、テーブルおよびオペレーションに影響を受けたセカンダリインデックスの小計を返します。
- NONE — 書き込み容量の詳細は返されません。(これがデフォルトです)

PutItem

PutItem は新しい項目を作成します。同じキーを持つ項目がテーブルにすでに存在する場合は、新しい項目に置き換えられます。

Example

Thread テーブルに新しい項目を書き込みます。Thread のプライマリキーは、ForumName (パーティションキー) と Subject (ソートキー) で構成されます。

```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

--item の引数は、ファイル item.json に保存されます。

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"},
    "Message": {"S": "First post in this thread"},
    "LastPostedBy": {"S": "fred@example.com"},
    "LastPostDateTime": {"S": "201603190422"}
}
```

UpdateItem

指定されたキーを持つ項目が存在しない場合は、UpdateItem により新しい項目が作成されます。または、既存の項目の属性が変更されます。

更新式を使用して、変更する属性と新しい値を指定します。詳細については、「更新式」を参照してください。

更新式では、実際の値にプレースホルダーとして式の属性値を使用します。詳細については、「式の属性値」を参照してください。

Example

Thread 項目のさまざまな属性を変更します。オプションの ReturnValues パラメータは、更新後に表示されるように項目を表示します。詳細については、「戻り値」を参照してください。

```
aws dynamodb update-item \
    --table-name Thread \
    --key file://key.json \
    --update-expression "SET Answered = :zero, Replies = :zero, LastPostedBy
 = :lastpostedby" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

--key の引数は、ファイル key.json に保存されます。

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

--expression-attribute-values の引数は、ファイル expression-attribute-values.json に保存されます。

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

DeleteItem

DeleteItem は指定されたキーを持つ項目を削除します。

Example

次の AWS CLI の例で、Thread 項目を削除する方法について説明します。

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

戻り値

場合によっては、ある属性値を変更する前後にその属性値が表示されるように DynamoDB で返すこともできます。PutItem、UpdateItem、および DeleteItem オペレーションには、ReturnValues パラメータがあり、これを使用することで、属性の変更前後に、その属性値を返すことができます。

ReturnValues のデフォルトの値は NONE で、DynamoDB は変更された属性の情報を返しません。

以下は、DynamoDB API オペレーションごとに整理された ReturnValues のその他の有効な設定です。

PutItem

- ReturnValues: ALL_OLD
  - 既存の項目に上書きする場合、ALL_OLD は上書きの前に表示されたように、項目全体を返します。
  - 存在しない項目を書き込んだ場合は、ALL_OLD による影響はありません。

UpdateItem

UpdateItem の最も一般的な使用方法は、既存の項目の更新です。ただし、UpdateItem は実際にはアップサートを実行します。つまり、項目が存在しない場合は、自動的に作成します。

- ReturnValues: ALL_OLD
  - 既存の項目を更新する場合、ALL_OLD は更新前に表示されたように、項目全体を返します。
  - 存在しない項目を更新 (アップサート) すると、ALL_OLD による影響はありません。
- ReturnValues: ALL_NEW
  - 既存の項目を更新する場合、ALL_NEW は更新後に表示されるように、項目全体が返されます。
  - 存在しない項目を更新 (アップサート) すると、ALL_NEW は項目全体を返します。
- ReturnValues: UPDATED_OLD
  - 既存の項目を更新した場合、UPDATED_OLD は、更新前に表示されたように、更新された属性だけを返します。
  - 存在しない項目を更新 (アップサート) すると、UPDATED_OLD による影響はありません。
- ReturnValues: UPDATED_NEW
  - 既存の項目を更新した場合、UPDATED_NEW は、更新後に表示されるように、影響のある属性だけを返します。
  - 存在しない項目を更新 (アップサート) する場合、UPDATED_NEW は更新後に表示される、更新された属性だけを返します。

DeleteItem

- ReturnValues: ALL_OLD
  - 既存の項目を削除すると、ALL_OLD は削除の前に表示されたように、項目全体を返します。
  - 存在しない項目を削除すると、ALL_OLD はデータを返しません。

バッチオペレーション

複数の項目の読み込みや書き込みを必要とするアプリケーションのために、DynamoDB は `BatchGetItem` および `BatchWriteItem` オペレーションを提供します。これらのオペレーションを使用すると、アプリケーションから DynamoDB へのネットワークラウンドトリップの数を減らすことができます。さらに、DynamoDB は個別の読み込みまたは書き込みオペレーションを並行して実行します。同時実行またはスレッディングの管理をする必要がないので、アプリケーションにはこの並列処理が役立ちます。

バッチオペレーションは、基本的に複数の読み込みまたは書き込みリクエストをまとめます。たとえば、`BatchGetItem` リクエストに 5 つの項目が含まれている場合、DynamoDB によって 5 回の `GetItem` オペレーションが実行されます。同様に、`BatchWriteItem` リクエストに 2 つの PUT リクエストと 4 つの DELETE リクエストが含まれている場合、DynamoDB によって 2 つの `PutItem` リクエストと 4 つの `DeleteItem` リクエストが実行されます。

一般的に、バッチのすべてのリクエストが失敗しない限り、バッチオペレーションは失敗しません。たとえば、`BatchGetItem` オペレーションを実行する際、バッチの個々の `GetItem` リクエストが失敗したとします。この場合、`BatchGetItem` は失敗した `GetItem` リクエストからキーとデータを返します。バッチのその他の `GetItem` リクエストは影響を受けません。

BatchGetItem

1 回の `BatchGetItem` オペレーションには、最大 100 の個々の `GetItem` リクエストが含まれていて、最大 16 MB のデータを取得できます。さらに、`BatchGetItem` オペレーションで、複数のテーブルから項目を取得できます。

Example

一部の属性のみが返されるようにプロジェクション式を使用して `Thread` テーブルから 2 つの項目を取得します。

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

`--request-items` の引数は、ファイル `request-items.json` に保存されます。

```
{
    "Thread": {
        "Keys": [
            {
                "ForumName":{"S": "Amazon DynamoDB"},
                "Subject":{"S": "DynamoDB Thread 1"}
            },
            {
                "ForumName":{"S": "Amazon S3"},
                "Subject":{"S": "S3 Thread 1"}
            }
        ],
        "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"
    }
}
```

BatchWriteItem

BatchWriteItem オペレーションには最大 25 の個々の PutItem リクエストと DeleteItem リクエストを含むことができ、最大 16 MB のデータを書き込めます。(個々の項目の最大サイズは 400 KB です。) さらに、BatchWriteItem オペレーションで、複数のテーブルの項目を入力したり削除したりできます。

> **Note**
>
> BatchWriteItem では UpdateItem リクエストはサポートされません。

Example

ProductCatalog テーブルに 2 つの項目を書き込みます。

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

--request-items の引数は、ファイル request-items.json に保存されます。

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
            }
        }
    ]
}
```

アトミックカウンタ

この UpdateItem オペレーションを使用して、アトミックカウンター (他の書き込みリクエストに干渉することなく無条件に増分される数値属性) を実装できます。（すべての書き込みリクエストは、受信された順に適用されます）。アトミックカウンターでは、更新はべき等ではありません。つまり、UpdateItem を呼び出すたびに数値はインクリメントまたはデクリメントされます。アトミックカウンタの更新に使用されるインクリメント値が正の場合、オーバーカウントが発生する可能性があります。インクリメント値が負の場合、アンダーカウントの原因となります。

ウェブサイトの訪問者数を追跡するためにアトミックカウンターを使用できます。この場合、アプリケーションでは、現在値に関係なく、数値はインクリメントされます。UpdateItem オペレーションが失敗した場合、アプリケーションはオペレーションを再試行します。これには、カウンターを 2 度更新する恐れがありますが、ウェブサイトの訪問者数のカウントに多少の誤差があっても許容できるでしょう。

アトミックカウンターはカウントの誤差が許容されない場合にはふさわしくありません (銀行業務用のアプリケーションなど)。この場合は、アトミックカウンターの代わりに条件付き更新を使用する方が安全です。

詳細については、「数値属性の増減」を参照してください。

Example

次の AWS CLI の例では、商品の Price が 5 でインクリメントされます。この例では、カウンタが更新される前に項目が存在することがわかっていました。(UpdateItem は冪等性ではないので、Price はこのコードを実行するたびに増えていきます)。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": { "N": "601" }}' \
    --update-expression "SET Price = Price + :incr" \
    --expression-attribute-values '{":incr":{"N":"5"}}' \
    --return-values UPDATED_NEW
```

条件付きの書き込み

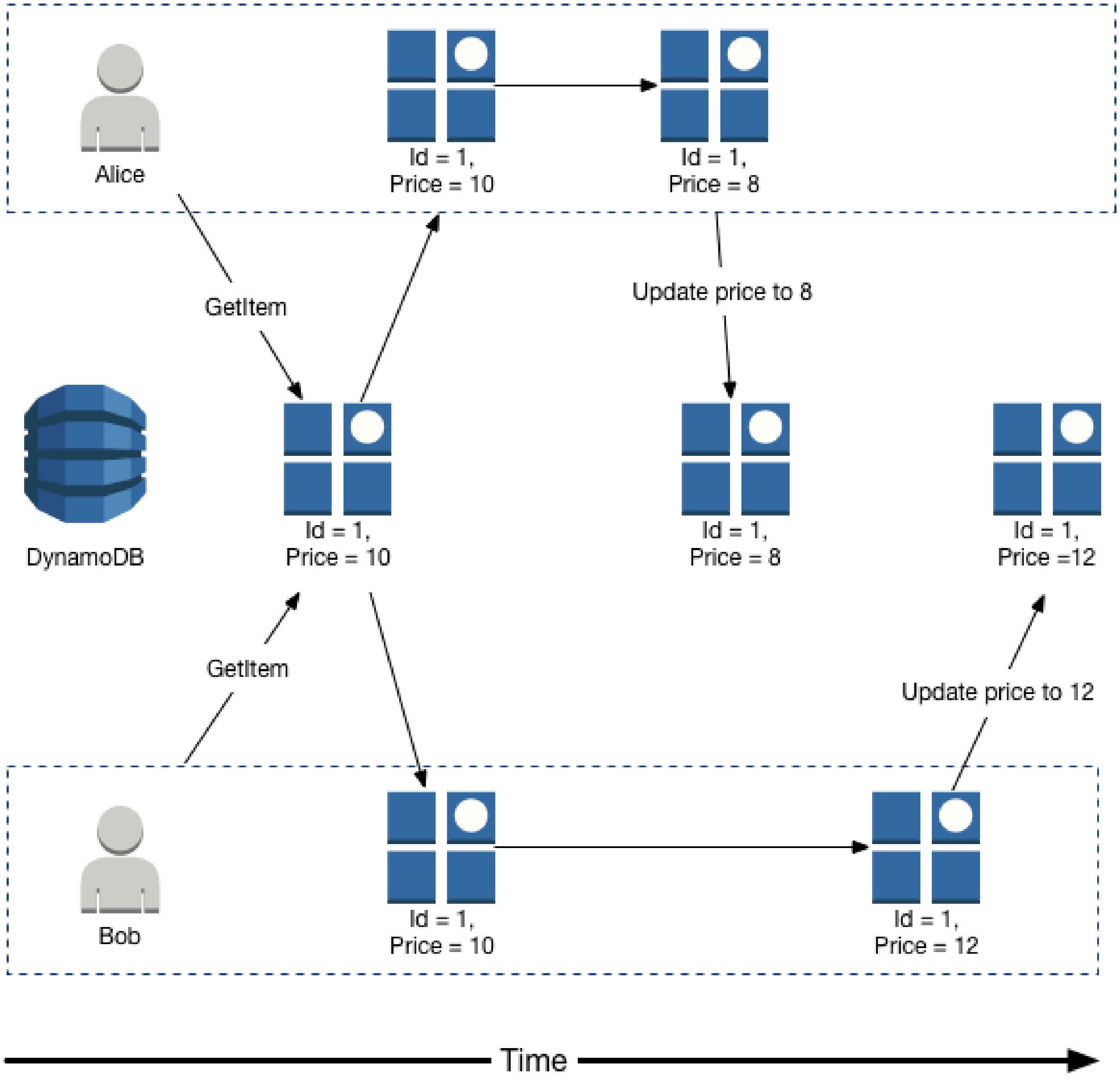
デフォルトでは、DynamoDB 書き込みオペレーション (PutItem、UpdateItem、DeleteItem) は無条件です。つまり、これらの各オペレーションでは、指定されたプライマリキーを持つ既存の項目が上書きされます。

DynamoDB はオプションでこれらのオペレーションの条件付き書き込みをサポートしています。条件付き書き込みが成功するのは、項目の属性が 1 つ以上の想定条件を満たす場合のみです。それ以外の場合は、エラーが返されます。

条件付き書き込みでは、その条件について項目の最新更新バージョンと照合します。なお、項目が以前に存在しなかった場合や、その項目に対して最後に成功した操作が削除であった場合、条件付き書き込みでは以前の項目は検出されません。

条件付き書き込みは多くの状況で役立ちます。たとえば、同じプライマリキーを持つ既存の項目がない場合にのみ、PutItem オペレーションが成功するようにできます。または、属性の 1 つに特定の値がある場合に UpdateItem オペレーションが項目を変更することを防ぐことができます。

条件付き書き込みは、複数のユーザーが同じ項目を変更しようとする場合に役立ちます。2 人のユーザー (Alice と Bob) が DynamoDB テーブルから同じ項目を処理している以下の図を考慮します。



Alice が AWS CLI を使用して `Price` 属性を 8 に更新するとします。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
    --expression-attribute-values file://expression-attribute-values.json
```

`--expression-attribute-values` の引数は、ファイル `expression-attribute-values.json` に保存されます。

```
{
    ":newval":{"N":"8"}
}
```

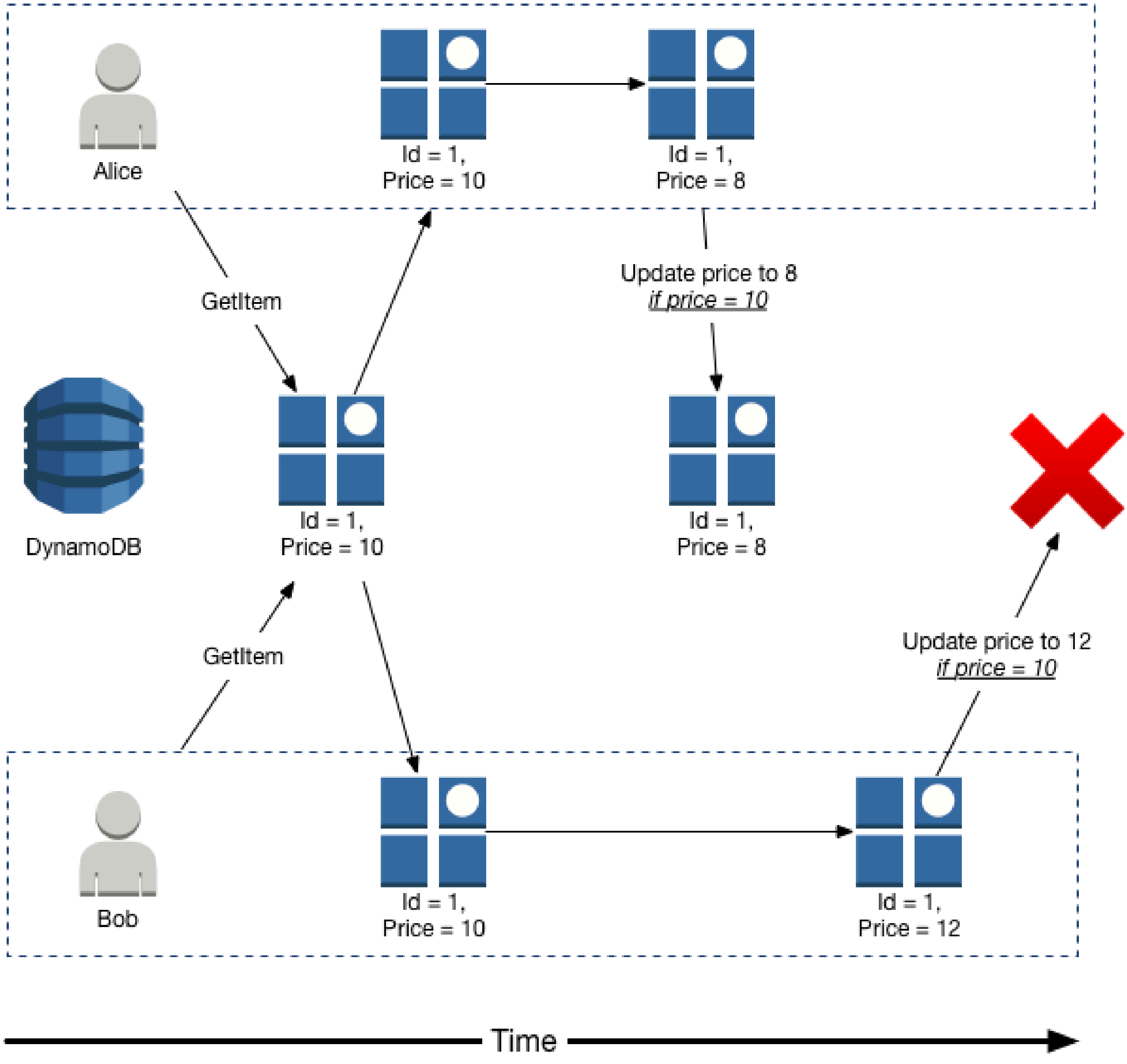
次に、Bob が後で同様の `UpdateItem` リクエストを発行しますが、`Price` を 12 に変更します。Bob には、`--expression-attribute-values` パラメータは次のように見えます。

```
{
    ":newval":{"N":"12"}
}
```

Bob のリクエストは成功しますが、その前の Alice の更新は失われます。

条件付き `PutItem`、`DeleteItem`、または `UpdateItem` をリクエストするには、条件式を指定します。条件式は、属性名、条件付き演算子および組み込み関数を含む文字列です。式全体の評価が true になる必要があります。それ以外の場合は、このオペレーションは失敗します。

次は、条件付き書き込みにより Alice の更新が上書きされるのを防ぐ方法について、以下の図を考慮します。



Alice はまず `Price` を 8 に更新を試行しますが、現在の `Price` が 10 である場合にのみ、という条件でそうします。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
    --condition-expression "Price = :currval" \
    --expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values の引数は、ファイル expression-attribute-values.json に保存されます。

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

条件が true に評価されるため、Alice の更新は成功します。

次に、Bob は、Price を 12 に 更新しますが、現在の Price が 10 である場合にのみ、という条件でそうします。Bob には、--expression-attribute-values パラメータは次のように見えます。

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Alice がすでに Price を 8 に変更していたので、条件式は false 評価され Bob の更新は失敗します。

詳細については、「条件式」を参照してください。

条件付き書き込みの冪等性

条件付き書き込みは、条件チェックが更新される同じ属性で行われる場合に、べき等性が保たれます。つまり、リクエスト時に項目の特定の属性値が想定されたものである場合にのみ、DynamoDB によって指定された書き込みリクエストが実行されます。

たとえば、UpdateItem リクエストを発行し、項目の Price を、現在の Price が 20 である場合のみ、3 増加させるとします。リクエストを送信した後、その結果を得る前にネットワークエラーが発生したため、リクエストが成功したかどうか不明です。この条件付き書き込みはべき等のオペレーションであるため、同じ UpdateItem リクエストを再試行できます。すると、DynamoDB は、現在の Price が 20 である場合のみ項目を更新します。

条件付き書き込みで消費されるキャパシティユニット

条件付き書き込み中に `ConditionExpression` が false と評価された場合でも、DynamoDB はテーブルの書き込みキャパシティを消費します。消費量は、既存の項目のサイズ (または最低 1) によって異なります。例えば、既存の項目が 300 KB で、作成または更新しようとしている新しい項目が 310 KB の場合、消費される書き込みキャパシティーユニットは、条件が満たされないかったら 300、満たされたら 310 になります。これが新しい項目 (既存の項目なし) の場合、消費される書き込みキャパシティーユニットは条件が満たされれば 1、条件が満たされなかったら 310 になります。

> **Note**
>
> 書き込みオペレーションでは、書き込みキャパシティユニットのみが消費されます。読み込みキャパシティユニットが消費されることはありません。

失敗した条件付き書き込みは `ConditionalCheckFailedException` を返します。これが起きると、消費された書き込みキャパシティに関する情報はレスポンスで返されません。

条件付き書き込みの際に消費された書き込みキャパシティーユニットの数を返すには、`ReturnConsumedCapacity` パラメータを使用します。

- `TOTAL` — 消費された書き込み容量単位の総数を返します。
- `INDEXES` — 消費された書き込み容量単位の総数とともに、テーブルおよびオペレーションに影響を受けたセカンダリインデックスの小計を返します。
- `NONE` — 書き込み容量の詳細は返されません。(これがデフォルトです)

> **Note**
>
> グローバルセカンダリインデックスとは異なり、ローカルセカンダリインデックスは、プロビジョンドスループット性能をそのテーブルと共有します。ローカルセカンダリインデックスでの読み込みと書き込みのアクティビティは、テーブルからプロビジョンドスループット性能を消費します。

DynamoDB での式の使用

Amazon DynamoDB では、式を使用して、項目から読み取る属性を示します。また、項目を書き込むときも式を使用して、満たす必要がある条件 (条件付き更新とも呼ばれます) と、属性を更新する方法を示します。このセクションでは、式の基本的な文法と利用可能な式の種類について説明します。

> Note
>
> 下位互換性のために、DynamoDB は式を使用しない条件パラメータもサポートします。詳細については、「レガシー条件パラメータ」を参照してください。
>
> 新しいアプリケーションでは、レガシーパラメータではなく式を使用する必要があります。

トピック

- 式を使用する時の項目属性の指定
- プロジェクション式
- DynamoDB の式の属性名
- 式の属性値
- 条件式
- 更新式

式を使用する時の項目属性の指定

このセクションでは、Amazon DynamoDB で式の項目属性を参照する方法を説明します。複数のリストやマップ内で深くネストされている場合でも、属性を使用できます。

トピック

- 最上位属性
- 入れ子の属性
- ドキュメントパス

サンプル項目: ProductCatalog

以下は、`ProductCatalog` テーブルの項目を示しています。(このテーブルについては、「テーブルとデータの例」で説明されています)。

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "Description": "123 description",
    "BicycleType": "Hybrid",
    "Brand": "Brand-Company C",
    "Price": 500,
    "Color": ["Red", "Black"],
    "ProductCategory": "Bicycle",
    "InStock": true,
    "QuantityOnHand": null,
    "RelatedItems": [
        341,
        472,
        649
    ],
    "Pictures": {
        "FrontView": "http://example.com/products/123_front.jpg",
        "RearView": "http://example.com/products/123_rear.jpg",
        "SideView": "http://example.com/products/123_left_side.jpg"
    },
    "ProductReviews": {
     "FiveStar": [
       "Excellent! Can't recommend it highly enough! Buy it!",
       "Do yourself a favor and buy this."
     ],
     "OneStar": [
       "Terrible product! Do not buy this."
     ]
    },
    "Comment": "This product sells out quickly during the summer",
    "Safety.Warning": "Always wear a helmet"
 }
```

次の点に注意してください。

- パーティションキー値 (Id) は 123 です。ソートキーはありません。
- ほとんどの属性に、String、Number、Boolean、Null などのスカラーデータ型があります。
- 1 つの属性 (Color) は String Set です。
- 次の属性はドキュメントデータ型です。
  - RelatedItems のリスト。各要素は関連製品の Id です。

- Pictures のマップ。各要素は対応するイメージファイルの URL と、写真の短い説明です。
- ProductReviews のマップ。各要素は、レーティングと、そのレーティングに対応する評価のリストを表します。最初に、このマップには 5 つ星と 1 つ星の評価が入力されます。

最上位属性

属性が別の属性に組み込まれていない場合、最上位属性と呼ばれます。ProductCatalog 項目の最上位属性は次のようになります。

- Id
- Title
- Description
- BicycleType
- Brand
- Price
- Color
- ProductCategory
- InStock
- QuantityOnHand
- RelatedItems
- Pictures
- ProductReviews
- Comment
- Safety.Warning

Color (リスト)、RelatedItems (リスト)、Pictures (マップ)、および ProductReviews (マップ) を除くすべての最上位属性はスカラーです。

入れ子の属性

属性が別の属性に組み込まれている場合、入れ子の属性と呼ばれます。入れ子の属性にアクセスするには、間接参照演算子を使用します。

- [n] — リストの要素

- . (ドット) — マップの要素

リスト要素へのアクセス

リスト要素の間接参照演算子は N で、n は要素数です。リストの要素はゼロベースであるため、[0] はリスト内の最初の要素、[1] は 2 番目の要素、という順番で表されます。次に例を示します。

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

要素 `ThisList[5]` は、それ自体がネストされたリストです。したがって、`ThisList[5][11]` は、そのリストの 12 番目の要素を参照します。

角括弧内の数は、負以外の整数である必要があります。そのため、次の式は無効です。

- `MyList[-1]`
- `MyList[0.4]`

マップ要素へのアクセス

マップ要素の間接参照演算子は . (ドット) です。マップの要素間の区切り文字として、ドットを使用します。

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

ドキュメントパス

式では、ドキュメントパスを使用して、属性の場所を DynamoDB に伝えます。最上位属性の場合、ドキュメントパスは単純に属性名になります。ネストされた属性の場合は、間接参照演算子を使用してドキュメントパスを構築します。

ドキュメントパスのいくつかの例を次に示します。(式を使用する時の項目属性の指定 に示された項目を参照してください)

- 最上位のスカラー属性。

```
Description
```

- 最上位のリスト属性。(これはいくつかの要素だけではなく、すべてを示すリストを返します)

```
RelatedItems
```

- `RelatedItems` リストの 3 番目の要素 (このリスト要素はゼロベースであることに注意してください)。

```
RelatedItems[2]
```

- 製品の正面の写真。

```
Pictures.FrontView
```

- すべての 5 つ星の評価。

```
ProductReviews.FiveStar
```

- 最初の 5 つ星の評価。

```
ProductReviews.FiveStar[0]
```

> Note
>
> ドキュメントパスの最大深度は 32 です。したがって、任意のパスの間接参照演算子の数はこの制限を超えることはできません。

次の要件を満たしている限り、ドキュメントパスには任意の属性名を使用できます。

- 属性名はポンド記号 (#) で始める必要がある
- 最初の文字は `a-z`、`A-Z`、または `0-9`
- 2 番目の文字 (存在する場合) は `a-z`、`A-Z`

> Note
>
> 属性名がこの要件を満たさない場合は、式の属性名をプレースホルダーとして定義する必要があります。

詳細については、「DynamoDB の式の属性名」を参照してください。

プロジェクション式

テーブルからデータを読み込むには、次のような GetItem、Query、または Scan のオペレーションを使用します。Amazon DynamoDB は、デフォルトですべての項目属性を返します。すべての属性ではなく、一部の属性のみを取得するには、プロジェクション式を使用します。

プロジェクション式は、任意の属性を識別する文字列です。1 つの属性を取得するには、名前を指定します。複数の属性の場合、名前をカンマで区切る必要があります。

ProductCatalog からの 式を使用する時の項目属性の指定 項目に基づく、プロジェクション式のいくつかの例を次に示します。

- 1 つの最上位属性。

  ```
  Title
  ```

- 3 つの最上位属性。DynamoDB は Color セット全体を取得します。

  ```
  Title, Price, Color
  ```

- 4 つの最上位属性。DynamoDB は RelatedItems および ProductReviews のコンテンツ全体を返します。

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

DynamoDB には予約語と特殊文字のリストもあります。最初の文字が a-z または A-Z であり、2 番目の文字 (ある場合) が a-z、A-Z、または 0-9 である場合は、投影式で任意の属性値を使用できます。属性名がこの要件を満たさない場合は、式の属性名をプレースホルダーとして定義する必要があります。詳細な一覧については、「DynamoDB の予約語」を参照してください。DynamoDB では、# (ハッシュ) および : (コロン) に特別な意味があります。

DynamoDB では、命名目的でこれらの予約語と特殊文字を使用することができますが、お勧めしません。これは、式でこれらの名前を使用するたびに、プレースホルダー変数を定義する必要があるためです。詳細については、「DynamoDB の式の属性名」を参照してください。

以下の AWS CLI の例に示しているのは、GetItem オペレーションでプロジェクション式を使用する方法です。この式は最上位スカラー射影内の属性 (Description)、リスト内の最初の要素 (RelatedItems[0])、およびマップ内に入れ子にするリスト (ProductReviews.FiveStar) を取得します。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key file://key.json \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

この例では、次の JSON が返されます。

```
{
    "Item": {
        "Description": {
            "S": "123 description"
        },
        "ProductReviews": {
            "M": {
                "FiveStar": {
                    "L": [
                        {
                            "S": "Excellent! Can't recommend it highly enough! Buy it!"
                        },
                        {
                            "S": "Do yourself a favor and buy this."
                        }
                    ]
                }
            }
        },
        "RelatedItems": {
            "L": [
                {
                    "N": "341"
                }
            ]
        }
    }
}
```

`--key` の引数は、ファイル `key.json` に保存されます。

```
{
    "Id": { "N": "123" }
}
```

プログラミング言語別のコード例については、「DynamoDB および AWS SDK の使用開始」を参照してください。

DynamoDB の式の属性名

式の属性名は、実際の属性名の代わりとして Amazon DynamoDB 式で使用するプレースホルダーです。式の属性名はシャープ記号 (#) で始まり、1 つ以上の英数字とアンダースコア (_) 文字が続きます。

このセクションでは、式の属性名を使用する必要があるいくつかの状況について説明します。

> **Note**
>
> このセクションの例では AWS Command Line Interface (AWS CLI) を使用します。プログラミング言語別のコード例については、「DynamoDB および AWS SDK の使用開始」を参照してください。

トピック

予約語

DynamoDB 予約語と競合する属性名を含む式を書く必要が生じることもあります。予約語の一覧については、「DynamoDB の予約語」を参照してください。

たとえば、AWS CLI は予約語であるため、次の COMMENT の例は失敗します。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

この問題に対処するには、Comment を、#c などの式の属性名で置き換えることができます。#（シャープ記号）は必須であり、これが属性名のプレースホルダーであることを示します。AWS CLI の例は次のようになります。

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
     --expression-attribute-names '{"#c":"Comment"}'
```

> Note
>
> 属性名が数値で始まるか、スペースまたは予約語を含む場合、式の属性名を使用して式のその属性名を置き換える必要があります。

特殊文字を含む属性名

式では、ドット (".") はドキュメントパスの区切り文字として解釈されます。ただし、DynamoDB では属性名の一部としてドット文字や、ハイフン (「-」) などの特殊文字を使用することもできます。これは、あいまいな意味を持つことがあります。例として、ProductCatalog 項目から Safety.Warning 属性を取得するとします (式を使用する時の項目属性の指定 を参照してください)。

プロジェクション式を使用して、Safety.Warning にアクセスするとします。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

DynamoDB は、予想される文字列 ("Always wear a helmet") ではなく空の結果を返します。これは、DynamoDB が式のドットをドキュメントパスの区切り文字として解釈するためです。この場合、#sw の置換として式の属性名 (Safety.Warning など) を定義する必要があります。その後、次のプロジェクション式を使用します。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

次に、DynamoDB が正しい結果を返します。

Note

属性名にドット (「.」) またはハイフン (「-」) が含まれている場合は、式の属性名を使用して、式でその属性名を置き換える必要があります。

入れ子の属性

ここでは、次のプロジェクション式を使用して、入れ子の属性 ProductReviews.OneStar にアクセスするとします。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.OneStar"
```

結果は、1 つ星の製品レビューすべてを含みます (正常です)。

ただし、代わりに式属性名を使用することにした場合はどうなるでしょうか。たとえば、#pr1star の置換として ProductReviews.OneStar を定義した場合はどうなるでしょうか。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr1star"  \
    --expression-attribute-names '{"#pr1star":"ProductReviews.OneStar"}'
```

DynamoDB は予想される 1 つ星の評価のマップの代わりに空の結果を返します。これは、DynamoDB が式属性名のドットを属性名内の文字として解釈するためです。DynamoDB が式の属性名 #pr1star を評価すると、ProductReviews.OneStar がスカラー属性を参照していると判断します。これは意図したものではありません。

ドキュメントパス内の要素ごとに式の属性名を定義するのが正しい方法です。

- #pr — ProductReviews
- #1star — OneStar

その後、プロジェクション式として #pr.#1star を使用します。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

次に、DynamoDB が正しい結果を返します。

属性名の繰り返し

式の属性名は、同じ属性名を繰り返し参照する必要がある場合に役立ちます。たとえば、`ProductCatalog` 項目からいくつかの評価を取得する次の式を考えてみます。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

これをより簡潔にするために、`ProductReviews` を、`#pr` などの式の属性名で置き換えることができます。変更された式は次のようになります。

- `#pr.FiveStar, #pr.ThreeStar, #pr.OneStar`

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \
    --expression-attribute-names '{"#pr":"ProductReviews"}'
```

式属性名を定義した場合、式全体で一貫して使用する必要があります。さらに、`#` 記号を省略することはできません。

式の属性値

属性を値と比較する必要がある場合は、式の属性値をプレースホルダーとして定義します。Amazon DynamoDB の式の属性値は、比較する実際の値、つまりランタイムするまでわからない可能性のある値の代わりになります。式の属性値はコロン (`:`) で始まり、1 つ以上の英数字が続きます。

たとえば、`ProductCatalog` で入手でき、費用が `Black` 以下の `500` 項目のすべてを返すとします。この `Scan` (AWS Command Line Interface) の例のように AWS CLI オペレーションでフィルタ式を使用できます。

```
aws dynamodb scan \
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

`--expression-attribute-values` の引数は、ファイル `values.json` に保存されます。

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

Note

Scan オペレーションは、テーブルのすべての項目を読み込みます。したがって、大きなテーブルでの Scan の使用を避ける必要があります。
フィルタ式は Scan 結果に適用され、フィルタ式に一致しない項目は破棄されます。

式属性値を定義した場合、式全体で一貫して使用する必要があります。さらに、: 記号を省略することはできません。

式の属性値は、主な条件式、条件式、更新式、およびフィルター式で使用されます。

Note

プログラミング言語別のコード例については、「DynamoDB および AWS SDK の使用開始」を参照してください。

条件式

Amazon DynamoDB テーブルのデータを操作するには、`PutItem`、`UpdateItem` および `DeleteItem` オペレーションを使用します。(1 回の呼び出しで複数の `BatchWriteItem` または `PutItem` オペレーションを実行するために `DeleteItem` も使用できます)

これらのデータ操作オペレーションでは、どの項目を修正する必要があるかを判断するために、条件式を指定できます。条件式が true と評価される場合、オペレーションは成功で、それ以外の場合、オペレーションは失敗です。

PutItem、UpdateItem、DeleteItem の操作には ReturnValues パラメータがあり、これを使用することで、属性の変更前または変更後の属性値をそのまま返すことができます。詳細については、「ReturnValues」を参照してください。

条件式を使用する AWS Command Line Interface (AWS CLI) の例のいくつかを次に示します。これらの例は、「ProductCatalog」で紹介されている 式を使用する時の項目属性の指定 テーブルに基づいています。このテーブルのパーティションキーは Id です。ソートキーはありません。次の PutItem オペレーションは、例で参照するサンプル ProductCatalog 項目を作成します。

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

--item の引数は、ファイル item.json に保存されます。(分かりやすいように、いくつかの項目属性のみが使用されます)

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

トピック

- 条件付き配置
- 条件付き削除
- 条件付き更新
- 条件式の例
- 比較演算子および関数リファレンス

条件付き配置

PutItem オペレーションはプライマリキーが同じ項目を上書きします (存在する場合)。これを回避するには、条件式を使用します。これにより、問題の項目が同じプライマリキーを持っていない場合にのみ書き込みが続行されます。

次の例では、attribute_not_exists() を使用して、書き込み操作を試みる前に、プライマリキーがテーブルに存在するかどうかを確認しています。

> **Note**
>
> プライマリキーがパーティションキー (pk) とソートキー (sk) の両方で構成されている場合、このパラメータは、書き込み操作を試みる前に `attribute_not_exists(pk)` と `attribute_not_exists(sk)` の両方が true または false のどちらに評価されるのかを確認します。

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

条件式が false と評価し、DynamoDB は次のエラーメッセージを返します。条件付きリクエストが失敗しました。

> **Note**
>
> `attribute_not_exists` および他の関数についての詳細は、「比較演算子および関数リファレンス」を参照してください。

条件付き削除

条件付き削除を実行するには、条件式とともに `DeleteItem` オペレーションを使用します。条件式は、オペレーションが成功するためには true に評価される必要があります。それ以外の場合、オペレーションは失敗します。

条件式 からの項目を検討してください。

```
{
    "Id": {
        "N": "456"
    },
    "Price": {
        "N": "650"
    },
    "ProductCategory": {
        "S": "Sporting Goods"
    }
}
```

次の条件を満たしている場合にだけ項目を削除するとします。

- ProductCategory は「Sporting Goods」または「Gardening Supplies」のどちらかです。
- Price は 500～600 です。

次の例では、項目を削除しようとしています。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo
 and :hi)" \
    --expression-attribute-values file://values.json
```

--expression-attribute-values の引数は、ファイル values.json に保存されます。

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> Note
>
> 条件式では、: (コロン文字) は式の属性値 (実際の値のプレースホルダー) を示します。詳細については、「」を参照してください式の属性値
> IN、AND およびその他のキーワードについての詳細は、「比較演算子および関数リファレンス」を参照してください。

この例では、ProductCategory 比較の評価結果は true になりますが、Price 比較は false と評価されます。これにより、条件式は false と評価され、DeleteItem オペレーションは失敗します。

条件付き更新

条件付き更新を実行するには、条件式とともに UpdateItem オペレーションを使用します。条件式は、オペレーションが成功するためには true に評価される必要があります。それ以外の場合、オペレーションは失敗します。

> Note
>
> UpdateItem は更新式もサポートします。項目に加える変更を指定する変更内容を指定します。詳細については、「更新式」を参照してください。

条件式 に示す項目から開始するとします。

```
{
    "Id": { "N": "456"},
    "Price": {"N": "650"},
    "ProductCategory": {"S": "Sporting Goods"}
}
```

次の例では、UpdateItem オペレーションを実行します。製品の Price を 75 減らしようとしていますが、現在の Price が 500 以下の場合、条件式が更新を防ぎます。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --update-expression "SET Price = Price - :discount" \
    --condition-expression "Price > :limit" \
    --expression-attribute-values file://values.json
```

--expression-attribute-values の引数は、ファイル values.json に保存されます。

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

開始 Price が 650 の場合、UpdateItem オペレーションによって Price は 575 に下げられます。UpdateItem アクションを再度実行すると、Price は 500 に減らされます。3 度目に実行した場合、条件式は false と評価されて、更新は失敗します。

> Note
>
> 条件式では、: (コロン文字) は式の属性値 (実際の値のプレースホルダー) を示します。詳細については、「式の属性値」を参照してください。
> 「>」およびその他の演算子についての詳細は、「比較演算子および関数リファレンス」を参照してください。

条件式の例

次の例で使用される関数の詳細については、「比較演算子および関数リファレンス」を参照してください。式で異なる属性タイプを指定する方法の詳細については、「式を使用する時の項目属性の指定」を参照してください。

項目の属性を確認

属性が存在するか (または存在しないか) を確認できます。条件式が true と評価される場合、オペレーションは成功で、それ以外の場合、オペレーションは失敗です。

次の例では、attribute_not_exists 属性がない場合にのみ製品を削除するために Price を使用します。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB は attribute_exists 関数も提供します。次の例では、悪いレビューを受け取った場合のみ製品を削除します。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

属性タイプの確認

属性値のデータ型は、attribute_type 関数を使用して確認できます。条件式が true と評価される場合、オペレーションは成功で、それ以外の場合、オペレーションは失敗です。

次の例では、attribute_type を使用して、文字列セットタイプの Color 属性がある場合にのみ、製品を削除します。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_type(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values の引数は、expression-attribute-values.json ファイルに格納されます。

```
{
    ":v_sub":{"S":"SS"}
}
```

文字列開始値の確認

文字列属性値が特定のサブ文字列で始まるかどうかを確認するには、begins_with 関数を使用します。条件式が true と評価される場合、オペレーションは成功で、それ以外の場合、オペレーションは失敗です。

次の例では、begins_with マップの FrontView 要素が特定の値で始まる場合にのみ、Pictures を使用して製品を削除します。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

--expression-attribute-values の引数は、expression-attribute-values.json ファイルに格納されます。

```
{
    ":v_sub":{"S":"http://"}
}
```

セット内の要素の確認

`contains` 関数を使用して、セット内の要素を確認したり、文字列内のサブ文字列を検索したりできます。条件式が true と評価される場合、オペレーションは成功で、それ以外の場合、オペレーションは失敗です。

次の例では、`contains` 文字列セットに特定の値を持つ要素がある場合にのみ、`Color` を使用して製品を削除します。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "contains(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

`--expression-attribute-values` の引数は、expression-attribute-values.json ファイルに格納されます。

```
{
    ":v_sub":{"S":"Red"}
}
```

属性値のサイズの確認

属性値のサイズを確認するには、`size` 関数を使用します。条件式が true と評価される場合、オペレーションは成功で、それ以外の場合、オペレーションは失敗です。

次の例では、`size` バイナリ属性のサイズが `VideoClip` バイトより大きい場合にのみ `64000` を使用して、製品を削除します。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "size(VideoClip) > :v_sub" \
    --expression-attribute-values file://expression-attribute-values.json
```

`--expression-attribute-values` の引数は、expression-attribute-values.json ファイルに格納されます。

```
{
    ":v_sub":{"N":"64000"}
}
```

比較演算子および関数リファレンス

このセクションでは、Amazon DynamoDB の書き込みフィルター式と条件式の組み込み関数およびキーワードについて説明します。DynamoDB で使用する関数とプログラミングの詳細については、「DynamoDB と AWS SDK を使用したプログラミング」と「DynamoDB API リファレンス」を参照してください。

トピック

- フィルター式と条件式の構文
- 比較の実行
- 関数
- 論理評価
- 括弧
- 条件の優先順位

フィルター式と条件式の構文

以下の構文の概要で、*operand* は次のいずれかです。

- 最上位の属性名 (Id、Title、Description、ProductCategory など)
- 入れ子の属性を参照するドキュメントパス

```
condition-expression ::=
      operand comparator operand
    | operand BETWEEN operand AND operand
    | operand IN ( operand (',' operand (, ...) ))
    | function
    | condition AND condition
    | condition OR condition
    | NOT condition
    | ( condition )

comparator ::=
    =
    | <>
    | <
    | <=
    | >
    | >=

function ::=
    attribute_exists (path)
    | attribute_not_exists (path)
    | attribute_type (path, type)
    | begins_with (path, substr)
    | contains (path, operand)
    | size (path)
```

比較の実行

これらのコンパレータを使用して、値の範囲または値の列挙リストに対してオペランドを比較します。

- *a* = *b* — *a* が *b* と等しい場合、True
- *a* <> *b* — *a* が *b* と等しくない場合、True
- *a* < *b* — *a* が *b* 未満の場合、True
- *a* <= *b* — *a* が *b* 以下である場合、True
- *a* > *b* — *a* が *b* より大きい場合、True
- *a* >= *b* — *a* が *b* 以上である場合、True

値の範囲または値の列挙リストに対してオペランドを比較するには、BETWEEN および IN キーワードを使用します。

- *a* BETWEEN *b* AND *c* - *a* が *b* 以上で、*c* 以下である場合、True。
- *a* IN (*b*, *c*, *d*) — *a* がリスト内の任意の値と等しい場合、True。例では、*b*、*c*、*d* のいずれかと等しい場合。リストには、コンマで区切って最大 100 個の値を指定できます。

関数

以下の関数を使用して、ある属性が項目に存在するか判定したり、属性の値を評価したりします。これらの関数名では大文字と小文字が区別されます。入れ子の属性では、完全ドキュメントパスを指定する必要があります。

関数	説明
`attribute_exists (` *`path`*`)`	項目に、path で指定した属性が含まれる場合、true。 例: Product テーブルの項目に側面図があるかどうかを確認します。 - `attribute_exists (#Pictures.#SideView)`
`attribute_not_exists (` *`path`*`)`	path で指定した属性が項目に存在しない場合、true。 例: 項目に Manufacturer 属性があるかどうかを確認します。 - `attribute_not_exists (Manufacturer)`
`attribute_type (` *`path`*`,` *`type`*`)`	指定したパスの属性が、特定のデータ型のものである場合、true。type パラメータは次のいずれかである必要があります。 - S — 文字列 - SS — 文字列セット - N — 数値 - NS — 数値セット - B — バイナリ - BS — バイナリセット - BOOL — ブール

関数	説明
	• NULL — Null • L — リスト • M — マップ type パラメータには、式の属性値を使用する必要があります。 例: QuantityOnHand 属性がリスト型のものであるかどうかを確認します。この例では、:v_sub は文字列 L のプレースホルダーです。 • `attribute_type (ProductReviews.FiveStar, :v_sub)` type パラメータには、式の属性値を使用する必要があります。
`begins_with (` *`path`*`,` *`substr`*`)`	path で指定された属性が特定のサブ文字列から始まる場合、true。 例: 正面図 URL の最初の数文字が `http://` かどうかを確認します。 • `begins_with (Pictures.FrontView, :v_sub)` 式の属性値 `:v_sub` は、`http://` のプレースホルダーです。

関数	説明
`contains (path, operand)`	path で指定された属性が次の属性である場合、true。 - 特定のサブ文字列を含む String。 - 設定の中に特定の要素を含む Set。 - リスト内に特定の要素を含む List。 path で指定された属性が String である場合、operand は String である必要があります。path で指定された属性が Set の場合、operand は一連の要素タイプである必要があります。 パスとオペランドは区別する必要があります。つまり、contains (a, a) がエラーを返します。 例: Brand 属性にサブ文字列 Company が含まれているかどうかを確認します。 - `contains (Brand, :v_sub)` 式の属性値 :v_sub は、Company のプレースホルダーです。 例: 製品が赤で入手可能かどうかを確認します。 - `contains (Color, :v_sub)`

関数	説明
	式の属性値 :v_sub は、Red のプレースホルダーです。

関数	説明
`size (path)`	属性のサイズを表す数値を返します。以下は、`size` で使用できる有効なデータ型です。 属性は `String` 型で、`size` は文字列の長さを返します。 例: 文字列 `Brand` が 20 文字以下であるかどうかを確認します。式の属性値 `:v_sub` は、`20` のプレースホルダーです。 - `size (Brand) <= :v_sub` 属性が `Binary` バイナリ型の場合、`size` は属性値のバイト数を返します。 例: `ProductCatalog` 項目に `VideoClip` という名前のバイナリ属性があるとします。この属性には使用中の製品の短いビデオが含まれます。次の式は、`VideoClip` が 64,000 バイトを超えるかどうかを確認します。式の属性値 `:v_sub` は、`64000` のプレースホルダーです。 - `size(VideoClip) > :v_sub` 属性が `Set` データ型の場合、`size` は設定の要素数を返します。 例: 製品が複数の色で入手可能かどうかを確認します。式の属性値 `:v_sub` は、`1` のプレースホルダーです。

関数	説明
	• `size (Color) < :v_sub` 属性が `List` 型または `Map` のものである場合、`size` は子要素の数を返します。 例: `OneStar` のレビューの数が特定のしきい値を超えたかどうかを確認します。式の属性値 `:v_sub` は、`3` のプレースホルダーです。 • `size(ProductReviews.OneStar) > :v_sub`

論理評価

論理評価を実行するには、`AND`、`OR`、`NOT` キーワードを使用します。以下のリストでは、*a* と *b* は評価される条件を示しています。

- *a* `AND` *b* — *a* と *b* の両方が true である場合、True。
- *a* `OR` *b* — *a* または *b* のどちらか (または両方) が true の場合、True。
- `NOT` *a*— *a* が false の場合は True。*a* が true の場合は Falsee。

以下は、オペレーションの AND のコード例です。

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

括弧

論理評価の優先順位を変更するには括弧を使用します。たとえば、条件 *a* と *b* が true で、条件 *c* が false であるとします。次の式は true と評価されます。

- *a* `OR` *b* `AND` *c*

しかし、条件を括弧で囲むと、それが最初に評価されます。たとえば、次の式は false と評価されます。

- `(a OR b) AND c`

> Note
>
> 式では括弧を入れ子にできます。最も内側の括弧が最初に評価されます。

以下は、論理評価で括弧を付けたコード例です。

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

条件の優先順位

DynamoDB では、条件は次の優先順位ルールを使用して左から右に評価されます。

- `= <> < <= > >=`
- `IN`
- `BETWEEN`
- `attribute_exists attribute_not_exists begins_with contains`
- 括弧
- `NOT`
- `AND`
- `OR`

更新式

`UpdateItem` オペレーションは、既存項目を更新します。また、存在しない場合は新しい項目をテーブルに追加します。更新する項目のキーを指定する必要があります。また、変更する属性を示す更新式と、その式に割り当てる値も指定する必要があります。

更新式は、`UpdateItem` が項目の属性を変更する方法を指定します。たとえば、スカラー値を設定したり、リストまたはマップから要素を削除したりします。

更新式の構文の概要を次に示します。

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

更新式は、1 つ以上の句で構成されます。各句は、SET、REMOVE、ADD、または DELETE キーワードで始まります。これらのいずれの句も、任意の順序で更新式に含めることができます。ただし、各アクションキーワードは 1 回のみ表示できます。

各句内には、カンマで区切った 1 つ以上のアクションがあります。各アクションはデータ変更を表します。

このセクションの例は、ProductCatalog の プロジェクション式 項目に基づいています。

以下のトピックでは、SET アクションのさまざまなユースケースについて説明します。

トピック

- SET — 項目属性の変更または追加
- REMOVE — 項目から属性を削除
- ADD — 数値とセットの更新
- DELETE — セットから要素を削除
- 複数の更新式を使用する

SET — 項目属性の変更または追加

1 つ以上の属性を項目に追加するには、更新式で SET アクションを使用します。これらのいずれかの属性が既に存在する場合、新しい値で上書きされます。

SET を使用して、Number 型である属性を増減することもできます。複数の SET アクションを実行するには、オペレーションをカンマで区切ります。

次の構文の概要について説明します。

- *path* 要素は、項目へのドキュメントパスです。
- *operand* 要素は、項目へのドキュメントパスまたは関数とすることができます。

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function
```

次の PutItem オペレーションは、例で参照するサンプル項目を作成します。

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

--item の引数は、ファイル item.json に保存されます。(分かりやすいように、いくつかの項目属性のみが使用されます)

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

トピック

属性の変更

Example

ProductCategory 属性および Price 属性を更新します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET ProductCategory = :c, Price = :p" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

--expression-attribute-values の引数は、ファイル values.json に保存されます。

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> UpdateItem オペレーションでは、--return-values ALL_NEW を指定すると、DynamoDB は更新後に表示される項目を返します。

リストおよびマップの追加

Example

新しいリストおよび新しいマップを追加します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems = :ri, ProductReviews = :pr" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

--expression-attribute-values の引数は、ファイル values.json に保存されます。

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

リストに要素を追加

Example

新しい属性を RelatedItems リストに追加します。リストの要素はゼロベースであるため、[0] はリスト内の最初の要素、[1] は 2 番目の要素、という順番で表されることに注意してください。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :ri" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

--expression-attribute-values の引数は、ファイル values.json に保存されます。

```
{
    ":ri": { "S": "Nails" }
}
```

> Note
>
> SET を使用してリスト要素を更新すると、その要素のコンテンツは、指定した新しいデータで置き換えられます。要素が既に存在していない場合、SET はリストの末尾に新しい要素を追加します。

1 つの SET オペレーションに複数の要素を追加すると、要素は要素番号で順にソートされます。

ネストされたマップ属性の追加

Example

ネストされたマップ属性を追加します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #pr.#5star[1] = :r5, #pr.#3star = :r3" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

`--expression-attribute-names` の引数は、ファイル `names.json` に保存されます。

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

`--expression-attribute-values` の引数は、ファイル `values.json` に保存されます。

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

数値属性の増減

既存の数値属性は増減することができます。これを行うには、+ (プラス) および - (マイナス) 演算子を使用します。

Example

項目の Price を下げます。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Price を上げるために、更新式で + 演算子を使用します。

リストに要素を追加

リストの最後に要素を追加できます。これを行うには、SET 関数の list_appendを使用します。(関数名では、大文字と小文字が区別されます)。list_append 関数は SET アクションに固有で、更新式でのみ使用できます。構文は次のとおりです。

- list_append (*list1*, *list2*)

この関数は、入力として 2 つのリストを受け取り、*list2* のすべての要素を *list1* に追加します。

Example

リストに要素を追加 で、RelatedItems リストを追加し Hammer と Nails の 2 つの要素を追加します。次に、RelatedItems の末尾にさらに 2 つの要素を追加します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(#ri, :vals)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values file://values.json  \
    --return-values ALL_NEW
```

--expression-attribute-values の引数は、ファイル values.json に保存されます。

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
}
```

最後に、RelatedItems の先頭にもう 1 つの要素を追加します。これを実行するには、list_append 要素の順序を入れ替えます。(list_append は 2 つのリストを入力し、1 番目のリストに 2 番目のリストを追加することに注意してください)。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(:vals, #ri)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values '{":vals": {"L": [ { "S": "Chisel" }]}}' \
    --return-values ALL_NEW
```

生成される RelatedItems 属性には次の順序で 5 つの要素が含まれます。Chisel、Hammer、Nails、Screwdriver、 Hacksaw。

既存の属性の上書きを防止

既存の属性の上書きを回避するには、SET で if_not_exists 関数を使用できます。(関数名では、大文字と小文字が区別されます)。if_not_exists 関数は SET アクションに固有で、更新式でのみ使用できます。構文は次のとおりです。

- if_not_exists (*path*, *value*)

指定された *path* で項目が属性を含まない場合、if_not_exists は *value* に評価されます。それ以外の場合は、*path* に評価されます。

Example

項目にすでに Price 属性がない場合にのみ、項目の Price を設定します (Price がすでに存在する場合は何も起こりません)。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = if_not_exists(Price, :p)" \
    --expression-attribute-values '{":p": {"N": "100"}}' \
    --return-values ALL_NEW
```

REMOVE — 項目から属性を削除

1 つ以上の属性を Amazon DynamoDB の項目から削除するには、更新式で REMOVE アクションを使用します。複数の REMOVE アクションを実行するには、オペレーションをカンマで区切ります。

更新式の REMOVE の構文の概要を次に示します。唯一のオペランドは、削除する属性のドキュメントパスです。

```
remove-action ::=
    path
```

Example

項目から属性を削除します。(属性が存在しない場合は、何も起こりません)。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

リストから要素を削除

REMOVE を使用して、リストから個別要素を削除できます。

Example

リストに要素を追加 で、5 つの要素を含むようにリスト属性 (RelatedItems) を変更します。

- [0]—Chisel
- [1]—Hammer
- [2]—Nails
- [3]—Screwdriver
- [4]—Hacksaw

次の AWS Command Line Interface (AWS CLI) の例では、リストから Hammer と Nails を削除します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Hammer と Nails を削除すると、残りの要素は変わります。リストには以下が含まれます。

- [0]—Chisel
- [1]—Screwdriver
- [2]—Hacksaw

ADD — 数値とセットの更新

> ⓘ Note
>
> 通常は、ADD ではなく SET を使用することをお勧めします。

項目に新しい属性および値を追加するには、更新式で ADD アクションを使用します。

属性が既に存在する場合、ADD の動作は属性のデータ型によって決まります。

- 属性が数値で、追加する値も数値である場合、値は既存の属性に数学的に追加されます (値が負の数値である場合は、既存の属性から減算されます)。
- 属性が設定され、追加する値も設定された場合、値は既存のセットに付加されます。

> ⓘ Note
>
> ADD アクションでは、数値とセットデータ型のみがサポートされます。

複数の ADD アクションを実行するには、オペレーションをカンマで区切ります。

次の構文の概要について説明します。

- *path* 要素は、属性へのドキュメントパスです。属性は Number またはセットデータ型である必要があります。

- *value* 要素は、属性に追加する数値 (Number データ型の場合)、または属性に付加するセット (セット型の場合) です。

```
add-action ::=
    path value
```

以下のトピックでは、ADD アクションのさまざまなユースケースについて説明します。

トピック

- 数値の追加
- セットに要素を追加

数値の追加

QuantityOnHand 属性が存在しないと想定します。次の AWS CLI の例では、QuantityOnHand を 5 に設定します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD QuantityOnHand :q" \
    --expression-attribute-values '{":q": {"N": "5"}}' \
    --return-values ALL_NEW
```

これで QuantityOnHand が存在するようになったので、QuantityOnHand が毎回 5 増分するように例を再実行できます。

セットに要素を追加

Color 属性が存在しないと想定します。次の AWS CLI の例では、Color を 2 つの要素を持つ文字列セットに設定します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Orange", "Purple"]}}' \
    --return-values ALL_NEW
```

これで Color が存在するので、さらに要素を追加できます。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Yellow", "Green", "Blue"]}}' \
    --return-values ALL_NEW
```

DELETE — セットから要素を削除

> ⚠ Important
>
> DELETE は Set データ型のみをサポートします。

1 つ以上の要素をセットから削除するには、更新式で DELETE アクションを使用します。複数の DELETE アクションを実行するには、オペレーションをカンマで区切ります。

次の構文の概要について説明します。

- *path* 要素は、属性へのドキュメントパスです。属性はセットデータ型である必要があります。
- *#####*は *path* から削除する 1 つ以上の要素です。*#####*はセット型として指定する必要があります。

```
delete-action ::=
    path subset
```

Example

セットに要素を追加 で、Color 文字列セットを作成します。この例では、セットから要素の一部を削除します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "DELETE Color :p" \
    --expression-attribute-values '{":p": {"SS": ["Yellow", "Purple"]}}' \
    --return-values ALL_NEW
```

複数の更新式を使用する

単一のステートメントに複数の更新式を使用できます。

Example

属性の値を変更して別の属性を完全に削除したい場合は、SET アクションと REMOVE アクションを 1 つのステートメントで使用できます。このオペレーションでは、Price 値を 15 に減らし、項目から InStock 属性も削除します。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p REMOVE InStock" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Example

リストに追加しつつ別の属性の値も変更したい場合は、1 つのステートメントで 2 つの SET アクションを使用できます。このオペレーションでは、RelatedItems リスト属性に「Nails」が追加され、Price 値も 21 に設定されます。

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :newValue, Price = :newPrice" \
    --expression-attribute-values '{":newValue": {"S":"Nails"}, ":newPrice":
 {"N":"21"}}'  \
    --return-values ALL_NEW
```

Time to Live (TTL)

DynamoDB の Time to Live (TTL) は、不要になった項目を削除するためのコスト効率に優れた方法です。TTL では、項目がいつ不要になるかを示す有効期限タイムスタンプを項目ごとに定義できます。DynamoDB は、書き込みスループットを消費することなく、有効期限が切れてから数日以内に期限切れの項目を自動的に削除します。

TTL を使用するには、まずテーブルで TTL を有効にし、次に TTL の有効期限タイムスタンプを格納する特定の属性を定義します。タイムスタンプは UNIX エポック時間形式で秒単位で保存する必要があります。項目が作成または更新されるたびに、有効期限を計算して TTL 属性に保存できます。

期限切れの有効な TTL 属性を持つ項目は、随時システムによって削除される可能性があります。削除は、通常は有効期限が切れてから数日以内に行われます。削除待ちの期限切れ項目は、TTL 属性の変更や削除を含め、引き続き更新できます。期限切れの項目を更新する際には、その項目がその後削除されないように条件式を使用することをおすすめします。フィルター式を使用して、期限切れの項目を Scan 結果と Query 結果から削除します。

削除済みの項目は、通常の削除操作で削除された項目と同様に機能します。削除すると、項目はユーザーによる削除ではなくサービスによる削除として DynamoDB ストリームに入り、他の削除操作と同様にローカルセカンダリインデックスとグローバルセカンダリインデックスから削除されます。

グローバルテーブルバージョン 2019.11.21 (現行) を使用しており、TTL 機能も使用している場合、DynamoDB は TTL による削除をすべてのレプリカテーブルにレプリケートします。最初の TTL による削除の場合、TTL の有効期限切れが発生したリージョンでは、書き込みキャパシティユニット (WCU) は消費されません。ただし、レプリカテーブルにレプリケートされた TTL 削除では、プロビジョンドキャパシティを使用する場合はレプリケートされた書き込みキャパシティユニットが消費され、オンデマンドのキャパシティモードを使用する場合はレプリケートされた書き込みユニットが各レプリカリージョンで消費され、該当する料金が適用されます。

TTL の詳細については、次のトピックを参照してください。

トピック

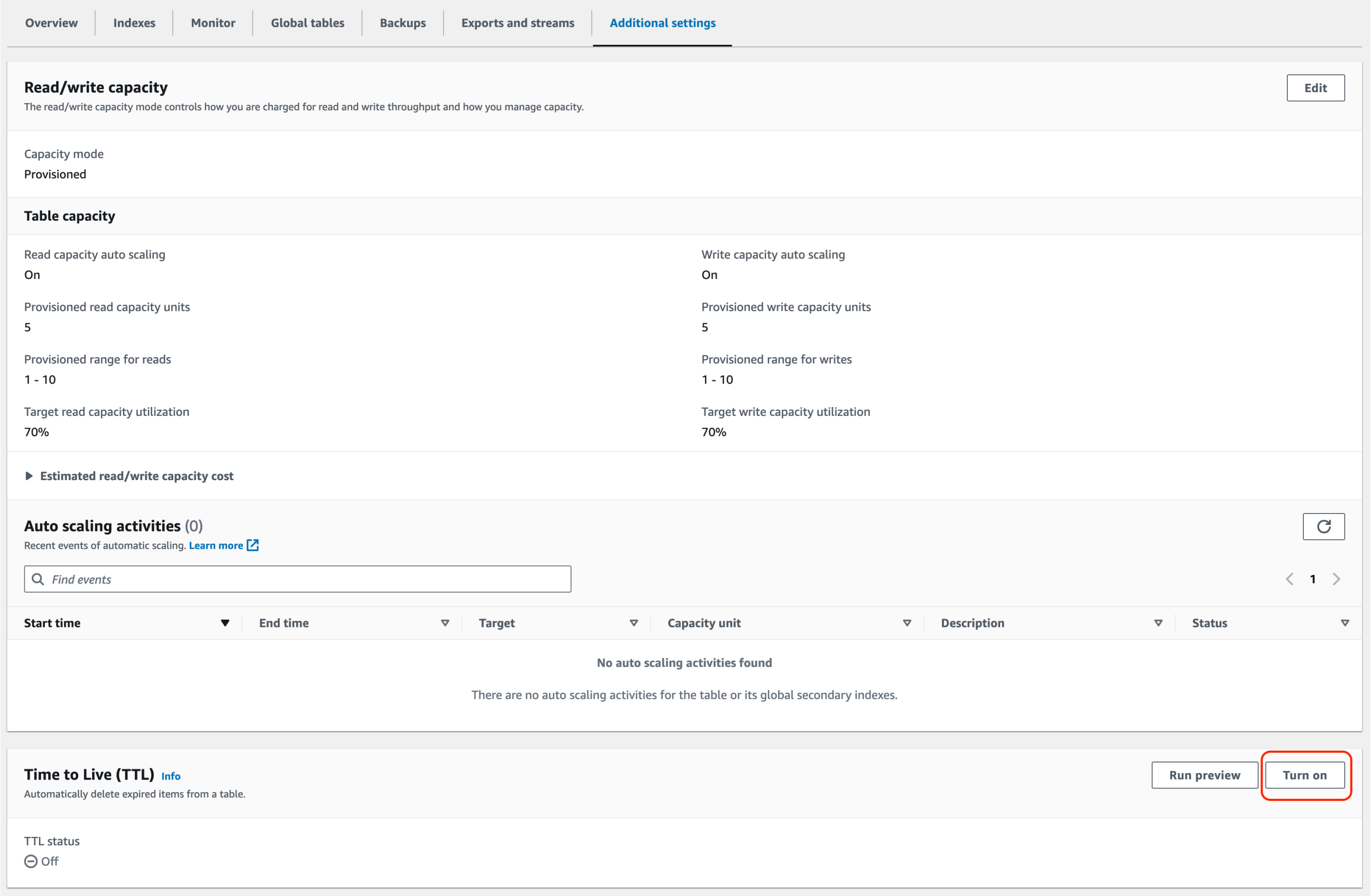
- Time to live (TTL) の有効化
- Time to Live (TTL) の計算
- 期限切れ項目の操作

Time to live (TTL) の有効化

TTL を有効にするには、Amazon DynamoDB コンソールまたは AWS Command Line Interface (AWS CLI) を使用するか、「Amazon DynamoDB API リファレンス」の該当する AWS SDK のいずれかを使用できます。すべてのパーティションで TTL を有効にするには約 1 時間かかります。

AWS コンソールを使用して DynamoDB TTL を有効にする

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. [テーブル] を選択し、変更するテーブルを選択します。
3. [追加の設定] タブの [Time to Live (TTL)] セクションで、[オンにする] を選択します。



Read/write capacity

The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.

Capacity mode
Provisioned

Table capacity

Read capacity auto scaling
On

Provisioned read capacity units
5

Provisioned range for reads
1 - 10

Target read capacity utilization
70%

Write capacity auto scaling
On

Provisioned write capacity units
5

Provisioned range for writes
1 - 10

Target write capacity utilization
70%

▶ Estimated read/write capacity cost

Auto scaling activities (0)

Recent events of automatic scaling. Learn more

Start time ▼	End time ▽	Target ▽	Capacity unit ▽	Description ▽	Status ▽

No auto scaling activities found

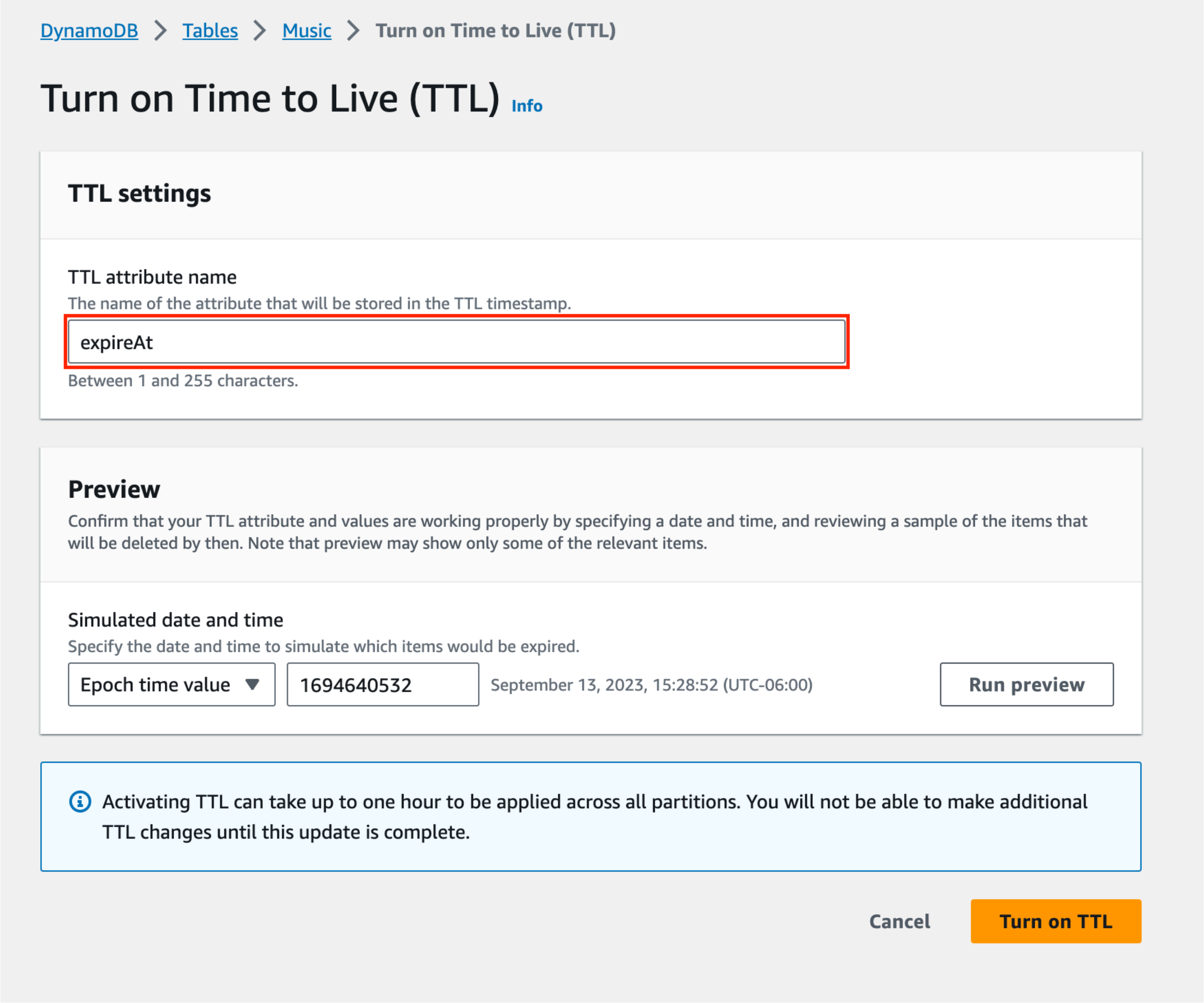
There are no auto scaling activities for the table or its global secondary indexes.

Time to Live (TTL) Info

Automatically delete expired items from a table.

TTL status
⊖ Off

4. テーブルで TTL を有効にする場合、DynamoDB では、項目が失効可能かどうかを判断するときにサービスが検索する、特定の属性名を識別する必要があります。以下に示す TTL 属性名では大文字と小文字が区別されます。また、TTL 属性名は読み取り/書き込み操作で定義された属性と一致する必要があります。不一致があると、期限切れの項目は削除されなくなります。TTL 属性の名前を変更するには、TTL を無効にした後で、新しい属性で TTL を再度有効にする必要があります。TTL は、無効化されてから約 30 分間、削除処理を続けます。TTL は復元されたテーブルで再設定する必要があります。



Turn on Time to Live (TTL)

TTL settings

TTL attribute name

The name of the attribute that will be stored in the TTL timestamp.

expireAt

Between 1 and 255 characters.

Preview

Confirm that your TTL attribute and values are working properly by specifying a date and time, and reviewing a sample of the items that will be deleted by then. Note that preview may show only some of the relevant items.

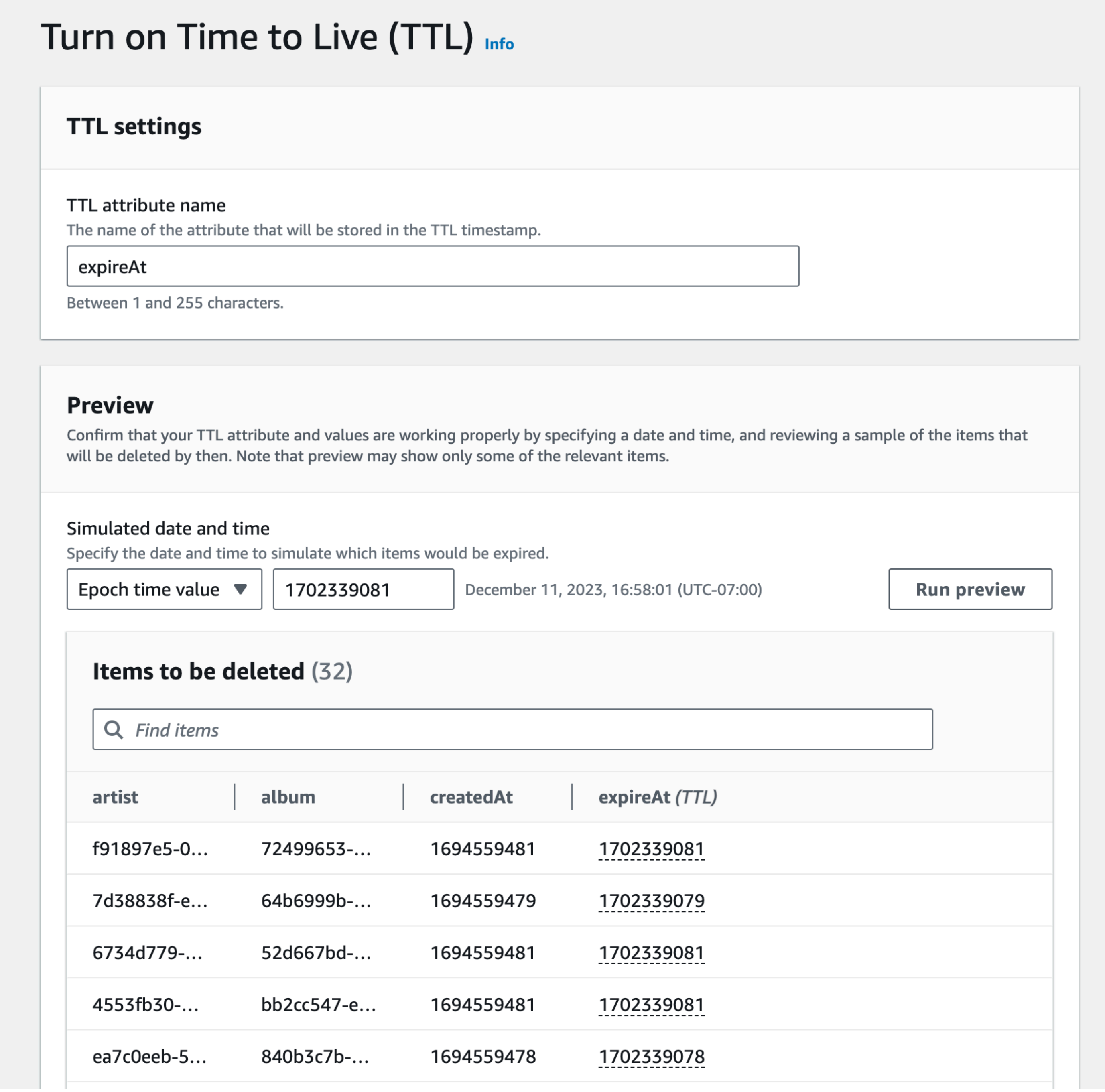
Simulated date and time

Specify the date and time to simulate which items would be expired.

Epoch time value ▼ 1694640532 September 13, 2023, 15:28:52 (UTC-06:00)

Activating TTL can take up to one hour to be applied across all partitions. You will not be able to make additional TTL changes until this update is complete.

5. (オプション) 有効期限の日付と時刻をシミュレートし、いくつかの項目を照合することでテストを実行できます。これにより、項目のサンプルリストが提供され、指定した TTL 属性名と有効期限を含む項目があることが確認されます。



Turn on Time to Live (TTL) Info

TTL settings

TTL attribute name
The name of the attribute that will be stored in the TTL timestamp.
expireAt
Between 1 and 255 characters.

Preview
Confirm that your TTL attribute and values are working properly by specifying a date and time, and reviewing a sample of the items that will be deleted by then. Note that preview may show only some of the relevant items.

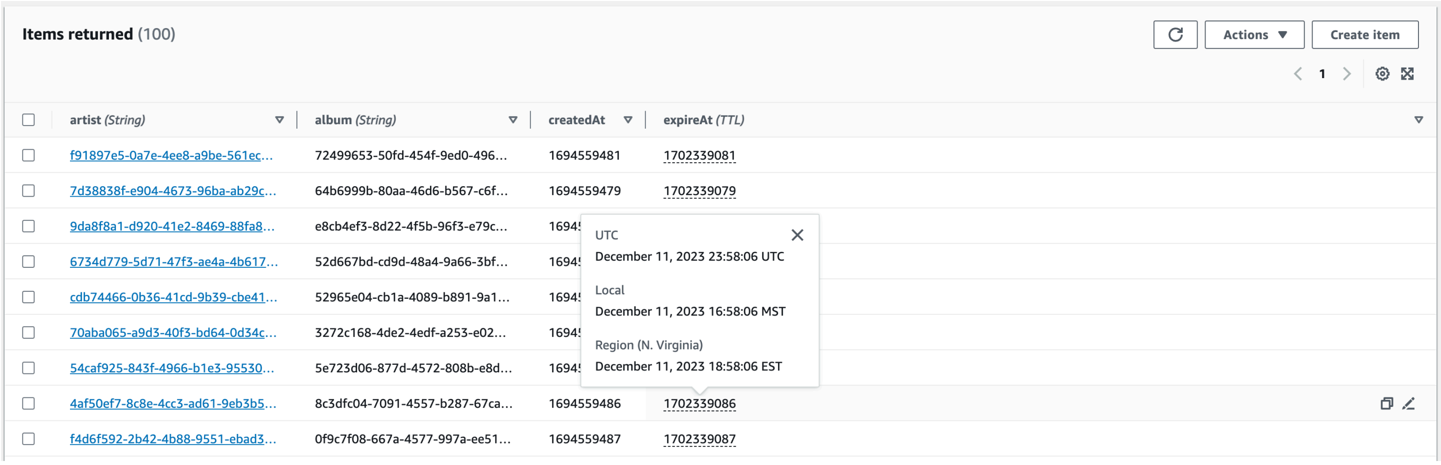
Simulated date and time
Specify the date and time to simulate which items would be expired.
Epoch time value ▼ 1702339081 December 11, 2023, 16:58:01 (UTC-07:00) Run preview

Items to be deleted (32)

Find items

artist	album	createdAt	expireAt *(TTL)*
f91897e5-0…	72499653-…	1694559481	1702339081
7d38838f-e…	64b6999b-…	1694559479	1702339079
6734d779-…	52d667bd-…	1694559481	1702339081
4553fb30-…	bb2cc547-e…	1694559481	1702339081
ea7c0eeb-5…	840b3c7b-…	1694559478	1702339078

TTL を有効にすると、DynamoDB コンソールで項目を表示するときに TTL 属性に TTL のマークが付けられます。属性の上にポインターを合わせると、項目が期限切れになる日時を表示できます。



Items returned (100)

□	artist (String)	album (String)	createdAt	expireAt (TTL)
□	f91897e5-0a7e-4ee8-a9be-561ec…	72499653-50fd-454f-9ed0-496…	1694559481	1702339081
□	7d38838f-e904-4673-96ba-ab29c…	64b6999b-80aa-46d6-b567-c6f…	1694559479	1702339079
□	9da8f8a1-d920-41e2-8469-88fa8…	e8cb4ef3-8d22-4f5b-96f3-e79c…	1694	
□	6734d779-5d71-47f3-ae4a-4b617…	52d667bd-cd9d-48a4-9a66-3bf…	1694	
□	cdb74466-0b36-41cd-9b39-cbe41…	52965e04-cb1a-4089-b891-9a1…	1694	
□	70aba065-a9d3-40f3-bd64-0d34c…	3272c168-4de2-4edf-a253-e02…	1694	
□	54caf925-843f-4966-b1e3-95530…	5e723d06-877d-4572-808b-e8d…	1694	
□	4af50ef7-8c8e-4cc3-ad61-9eb3b5…	8c3dfc04-7091-4557-b287-67ca…	1694559486	1702339086
□	f4d6f592-2b42-4b88-9551-ebad3…	0f9c7f08-667a-4577-997a-ee51…	1694559487	1702339087

UTC
December 11, 2023 23:58:06 UTC

Local
December 11, 2023 16:58:06 MST

Region (N. Virginia)
December 11, 2023 18:58:06 EST

API を使用して DynamoDB TTL を有効にする

Python

コードで UpdateTimeToLive オペレーションを使用して TTL を有効にすることができます。

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to the
 table.
    """
    try:
        dynamodb = boto3.client('dynamodb')

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={
                'Enabled': True,
                'AttributeName': ttl_attribute_name
            }
        )
        # In the returned response, check for a successful status code.
        if response['ResponseMetadata']['HTTPStatusCode'] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}")
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name, ex))
        raise


# your values
enable_ttl('your-table-name', 'expirationDate')
```

TTL が有効になっていることを確認するには、DescribeTimeToLive オペレーションを使用してテーブルの TTL ステータスを記述します。`TimeToLive` ステータスは `ENABLED` または `DISABLED` です。

```
# create a DynamoDB client
dynamodb = boto3.client('dynamodb')

# set the table name
table_name = 'YourTable'

# describe TTL
response = dynamodb.describe_time_to_live(TableName=table_name)
```

JavaScript

コードで UpdateTimeToLiveCommand オペレーションを使用して TTL を有効にすることができます。

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

AWS CLI を使用して Time to Live を有効にする

1. TTLExample テーブルで TTL を有効にします。

```
aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
specification "Enabled=true, AttributeName=ttl"
```

2. TTLExample テーブルで TTL を無効にします。

```
aws dynamodb describe-time-to-live --table-name TTLExample
{
    "TimeToLiveDescription": {
        "AttributeName": "ttl",
        "TimeToLiveStatus": "ENABLED"
    }
}
```

3. BASH シェルおよび AWS CLI を使用して 有効期限 (TTL) 属性が設定された項目を TTLExample テーブルに追加するには:

```
EXP=`date -d '+5 days' +%s`
aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
 {"N": "'$EXP'"}}'
```

この例では、現在の日付から開始し、有効期限を作成するために 5 日を追加します。次に、有効期限をエポック時間形式に変換してから、最後に項目を「TTLExample」テーブルに追加します。

> **Note**
>
> 有効期限 (TTL) の有効期限切れの値を設定する 1 つの方法は、有効期限に追加する秒数を計算することです。たとえば、5 日間は 432,000 秒です。ただし、多くの場合は最初に日付で開始し、そこから作業を開始することをお勧めします。

次の例のように、現在の時間をエポック時間形式で取得するのは非常に簡単です。

- Linux ターミナル: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

AWS CloudFormation を使用して DynamoDB TTL を有効にします。

1. `TTLExample` テーブルで TTL を有効にします。

   ```
   aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
   specification "Enabled=true, AttributeName=ttl"
   ```

2. `TTLExample` テーブルで TTL を無効にします。

   ```
   aws dynamodb describe-time-to-live --table-name TTLExample
   {
       "TimeToLiveDescription": {
           "AttributeName": "ttl",
           "TimeToLiveStatus": "ENABLED"
       }
   }
   ```

3. BASH シェルおよび AWS CLI を使用して 有効期限 (TTL) 属性が設定された項目を TTLExample テーブルに追加するには:

   ```
   EXP=`date -d '+5 days' +%s`
   aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
    {"N": "'$EXP'"}}'
   ```

この例では、現在の日付から開始し、有効期限を作成するために 5 日を追加します。次に、有効期限をエポック時間形式に変換してから、最後に項目を「TTLExample」テーブルに追加します。

> **Note**
>
> 有効期限 (TTL) の有効期限切れの値を設定する 1 つの方法は、有効期限に追加する秒数を計算することです。たとえば、5 日間は 432,000 秒です。ただし、多くの場合は最初に日付で開始し、そこから作業を開始することをお勧めします。

次の例のように、現在の時間をエポック時間形式で取得するのは非常に簡単です。

- Linux ターミナル: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Time to Live (TTL) の計算

TTL を実装する一般的な方法は、作成日または最終更新日に基づいて項目の有効期限を設定することです。これは、createdAt タイムスタンプと updatedAt タイムスタンプに時間を追加することで実現できます。例えば、新しく作成されたアイテムの TTL を createdAt + 90 日に設定できます。項目が更新されると、TTL は updatedAt + 90 日に再計算されます。

計算された有効期限は、秒単位のエポック形式である必要があります。TTL が 5 年を超えて経過すると、有効期限と削除の対象外になります。他の形式を使用すると、TTL プロセスはその項目を無視します。項目の有効期限を未来の時点に設定した場合、その期間が過ぎると項目は期限切れになります。例えば、有効期限を 1724241326 (2024 年 8 月 21 日 (月) 11:55:26 (GMT)) に設定したとします。指定した時刻以降、項目は有効期限切れになります。

トピック

- 項目を作成して Time to Live を設定する
- 項目を更新して Time to Live を再計算する

項目を作成して Time to Live を設定する

次の例は、新しい項目の作成時に有効期限を計算する方法を示しています。JavaScript では TTL 属性名として `'expireAt'` を使用し、Python では `'expirationDate'` を使用しています。代入ステートメントは、現在の時刻を変数として取得します。この例では、有効期限は現在の時刻から 90 日後として計算されます。その後、時刻はエポック形式に変換され、TTL 属性に整数データ型として保存されます。

Python

```
import boto3
from datetime import datetime, timedelta


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource('dynamodb', region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            'primaryKey': primary_key,
            'sortKey': sort_key,
            'creationDate': current_time,
            'expirationDate': expiration_time
        }

        table.put_item(Item=item)

        print("Item created successfully.")
    except Exception as e:
        print(f"Error creating item: {e}")
        raise


# Use your own values
create_dynamodb_item('your-table-name', 'us-west-2', 'your-partition-key-value',
 'your-sort-key-value')
```

JavaScript

このリクエストでは、新しく作成されたアイテムの有効期限を計算するロジックを追加します。

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 * 1000) /
 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, ex);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// use your own values
createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

項目を更新して Time to Live を再計算する

この例は、前のセクションの例の続きです。項目が更新された場合、有効期限を再計算できます。次の例では、`expireAt` タイムスタンプを現在の時刻から 90 日後として再計算します。

Python

```
import boto3
from datetime import datetime, timedelta


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource('dynamodb', region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={
                'partitionKey': primary_key,
                'sortKey': sort_key
            },
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={
                ':c': current_time,
                ':e': expire_at
            },
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item('your-table-name', 'us-west-2', 'your-partition-key-value',
 'your-sort-key-value')
```

更新操作の出力から、createdAt 時刻は変更されていないが、updatedAt 時刻と expireAt 時刻が更新されていることがわかります。expireAt 時刻は、最終更新日から 90 日後である 2023 年 10 月 19 日 (木) 午後 1 時 27 分 15 秒に設定されました。

partition_key	createdAt	updatedAt	expireAt	attribute_1	attribute_2
some_value	2023-07-17 14:11:05.322323	2023-07-19 13:27:15.213423	1697722035	new_value	some_value

JavaScript

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function updateDynamoDBItem(tableName, region, partitionKey, sortKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) /
 1000); ////is there a better way to do this?

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}

//enter your values here
updateDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

この概要で説明した TTL の例は、最近更新された項目だけをテーブルに保持する方法を示しています。更新された項目は有効期限が延びますが、作成後に更新されなかった項目は有効期限が切れて無料で削除されるため、ストレージが削減され、テーブルが整理されます。

期限切れ項目の操作

削除が保留になっている期限切れのアイテムは、読み取り操作と書き込み操作の対象から除外できます。これは、期限切れのデータが無効になり、使用すべきではなくなった場合に役に立ちます。フィルターが適用されていないと、バックグラウンドプロセスによって削除されるまで、読み取り/書き込み操作で引き続き表示されます。

> **Note**
>
> これらのアイテムは、削除されるまではストレージコストと読み取りコストにカウントされます。

TTL 削除は DynamoDB ストリームで確認できますが、削除が行われたリージョンでのみ確認できます。グローバルテーブルリージョンにレプリケートされた TTL 削除は、削除がレプリケートされたリージョンの DynamoDB ストリームでは確認できません。

期限切れの項目を読み取り操作から除外する

Scan や Query などの読み取り操作では、フィルター式を使用して削除待ちの期限切れアイテムを除外できます。以下のコードスニペットに示すように、フィルター式は TTL 時間が現在の時刻と同じかそれ以下の項目を除外できます。そのためには、現在の時刻を変数 (now) として取得する代入ステートメントを使用し、これをエポックタイム形式のために int に変換します。

Python

```
import boto3
from datetime import datetime
```

```


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource('dynamodb',
                                  region_name='us-east-1')

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key('partitionKey').eq(partition_key),
            FilterExpression=dynamodb.conditions.Attr('expireAt').gt(current_time)
        )

        # Print the items that are not expired
        for item in response['Items']:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
```

```
query_dynamodb_items('Music', 'your-partition-key-value')
```

更新操作の出力から、`createdAt` 時刻は変更されていないが、`updatedAt` 時刻と `expireAt` 時刻が更新されていることがわかります。`expireAt` 時刻は、最終更新日から 90 日後である 2023 年 10 月 19 日 (木) 午後 1 時 27 分 15 秒に設定されました。

partition_key	createdAt	updatedAt	expireAt	attribute_1	attribute_2
some_value	2023-07-17 14:11:05.322323	2023-07-19 13:27:15.213423	1697722035	new_value	some_value

Javascript

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function queryDynamoDBItems(tableName, region, primaryKey) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

//enter your own values here
queryDynamoDBItems('your-table-name', 'your-partition-key-value');
```

期限切れのアイテムに条件付きで書き込む

条件式を使用すると、期限切れのアイテムに対する書き込みを回避できます。以下のコードスニペットは、有効期限が現在の時間よりも大きいかどうかをチェックする条件付き更新です。true の場合、書き込みオペレーションは続行されます。

Python

```
import boto3
from datetime import datetime, timedelta
from botocore.exceptions import ClientError



def update_dynamodb_item(table_name, region, primary_key, sort_key, ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource('dynamodb', region_name=region)
        table = dynamodb.Table(table_name)
        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'ttlExpirationDate' is not
 expired
        condition_expression = "ttlExpirationDate > :val2"

        # Define the expression attribute values
        expression_attribute_values = {
            ':val1': ttl_attribute,
            ':val2': updated_expiration_time
        }

        response = table.update_item(
            Key={
                'primaryKey': primary_key,
                'sortKey': sort_key
            },
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values
        )

        print("Item updated successfully.")
        return response['ResponseMetadata']['HTTPStatusCode']  # Ideally a 200 OK
    except ClientError as e:
        if e.response['Error']['Code'] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'ttlExpirationDate' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value',
                     'your-ttl-attribute-value')
```

更新操作の出力から、createdAt 時刻は変更されていないが、updatedAt 時刻と expireAt 時刻が更新されていることがわかります。expireAt 時刻は、最終更新日から 90 日後である 2023 年 10 月 19 日 (木) 午後 1 時 27 分 15 秒に設定されました。

partition_key	createdAt	updatedAt	expireAt	attribute_1	attribute_2
some_value	2023-07-17 14:11:05.322323	2023-07-19 13:27:15.213423	1697722035	new_value	some_value

Javascript

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";


// Example function to update an item in a DynamoDB table.
//  The function should take the table name, region, partition key, sort key, and
 new attribute as arguments.
//  The function should use the DynamoDB client to update the item.
//  The function should return the updated item.
//  The function should handle errors and exceptions.
const updateDynamoDBItem = async (tableName, region, partitionKey, sortKey,
 newAttribute) => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Enter your values here
updateDynamoDBItem('your-table-name', "us-east-1",'your-partition-key-value', 'your-sort-key-value', 'your-new-attribute-value');
```

DynamoDB ストリーム内の削除済みアイテムの特定

ストリームレコードにはユーザー ID フィールド `Records[<index>].userIdentity` が含まれます。TTL プロセスによって削除された項目には、次のフィールドが含まれています。

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

次の JSON は 1 つのストリームレコードの関連する部分を示しています。

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
        "principalId": "dynamodb.amazonaws.com"
      }
    ...
  }
]
```

項目の操作: Java

AWS SDK for Java ドキュメント API を使用して、テーブル内の Amazon DynamoDB 項目に対して、一般的な作成、読み込み、更新、削除 (CRUD) のオペレーションを実行できます。

> Note
>
> SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングすることができます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「」を参照してくださいJava 1.x: DynamoDBMapper

このセクションには、いくつかの Java ドキュメント API 項目のアクションを実行する Java サンプルといくつかの完全な操作例が含まれています。

トピック

項目の置換

putItem メソッドによって、項目をテーブルに格納します。項目が存在する場合、その項目全体が置き換えられます。項目全体を置き換える代わりに固有の属性のみを更新する場合は、updateItem メソッドを使用できます。詳細については、「」を参照してください項目の更新

以下のステップに従ってください。

1. DynamoDB クラスのインスタンスを作成します。
2. 操作対象のテーブルを表すために、Table クラスのインスタンスを作成します。
3. 新しい項目を表す Item クラスのインスタンスを作成します。新しい項目のプライマリキーと属性を指定する必要があります。
4. 前の手順で作成した putItem を使用して、Table オブジェクトの Item メソッドを呼び出します。

以下の Java コード例は、前述のタスクの例です。このコードでは、ProductCatalog テーブルに新しい項目を書き込みます。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

// Build a list of related items
List<Number> relatedItems = new ArrayList<Number>();
relatedItems.add(341);
relatedItems.add(472);
relatedItems.add(649);

//Build a map of product pictures
Map<String, String> pictures = new HashMap<String, String>();
pictures.put("FrontView", "http://example.com/products/123_front.jpg");
pictures.put("RearView", "http://example.com/products/123_rear.jpg");
pictures.put("SideView", "http://example.com/products/123_left_side.jpg");

//Build a map of product reviews
Map<String, List<String>> reviews = new HashMap<String, List<String>>();
```

```
List<String> fiveStarReviews = new ArrayList<String>();
fiveStarReviews.add("Excellent! Can't recommend it highly enough!  Buy it!");
fiveStarReviews.add("Do yourself a favor and buy this");
reviews.put("FiveStar", fiveStarReviews);

List<String> oneStarReviews = new ArrayList<String>();
oneStarReviews.add("Terrible product!  Do not buy this.");
reviews.put("OneStar", oneStarReviews);

// Build the item
Item item = new Item()
    .withPrimaryKey("Id", 123)
    .withString("Title", "Bicycle 123")
    .withString("Description", "123 description")
    .withString("BicycleType", "Hybrid")
    .withString("Brand", "Brand-Company C")
    .withNumber("Price", 500)
    .withStringSet("Color",  new HashSet<String>(Arrays.asList("Red", "Black")))
    .withString("ProductCategory", "Bicycle")
    .withBoolean("InStock", true)
    .withNull("QuantityOnHand")
    .withList("RelatedItems", relatedItems)
    .withMap("Pictures", pictures)
    .withMap("Reviews", reviews);

// Write the item to the table
PutItemOutcome outcome = table.putItem(item);
```

前の例では、項目にはスカラー (String、Number、Boolean、Null)、セット (String Set)、ドキュメントタイプ (List、Map) があります。

オプションパラメータの指定

必須のパラメータに加え、putItem メソッドにはオプションパラメータも指定できます。たとえば、以下の Java コード例では、オプションパラメータを使用して、項目のアップロード条件を指定します。指定した条件を満たさない場合は、AWS SDK for Java が ConditionalCheckFailedException をスローします。このコード例では、putItem メソッドに以下のオプションパラメータを指定します。

- リクエストの条件を定義する ConditionExpression。このコードでは、特定の値に等しい ISBN 属性が既存の項目にある場合にのみ同じプライマリキーを持つ既存の項目を置き換えるという条件を定義します。

- 条件で使用される ExpressionAttributeValues のマップ。この場合、必要な置き換えは 1 つだけです。条件式のプレースホルダー :val が、実行時に、チェックする実際の ISBN 値に置き換えられます。

以下の例では、これらのオプションパラメータを使用して新しい書籍項目を追加します。

Example

```
Item item = new Item()
    .withPrimaryKey("Id", 104)
    .withString("Title", "Book 104 Title")
    .withString("ISBN", "444-4444444444")
    .withNumber("Price", 20)
    .withStringSet("Authors",
        new HashSet<String>(Arrays.asList("Author1", "Author2")));

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val", "444-4444444444");

PutItemOutcome outcome = table.putItem(
    item,
    "ISBN = :val", // ConditionExpression parameter
    null,          // ExpressionAttributeNames parameter - we're not using it for this
 example
    expressionAttributeValues);
```

PutItem および JSON ドキュメント

JSON ドキュメントは、DynamoDB テーブルに属性として格納できます。これを行うには、withJSON 項目の Item メソッドを使用します。このメソッドは、JSON ドキュメントを解析し、各要素を DynamoDB のネイティブデータ型にマッピングします。

特定の製品の注文を処理できるベンダーを含む、次の JSON ドキュメントを格納するとします。

Example

```
{
    "V01": {
        "Name": "Acme Books",
        "Offices": [ "Seattle" ]
    },
    "V02": {
        "Name": "New Publishers, Inc.",
        "Offices": ["London", "New York"
        ]
    },
    "V03": {
        "Name": "Better Buy Books",
        "Offices": [ "Tokyo", "Los Angeles", "Sydney"
        ]
    }
}
```

withJSON メソッドを使用して、ProductCatalog という名前の Map 属性でこれを VendorInfo テーブルに格納することができます。次の Java コード例はそれを行う方法を示しています。

```
// Convert the document into a String.  Must escape all double-quotes.
String vendorDocument = "{"
    + "    \"V01\": {"
    + "        \"Name\": \"Acme Books\","
    + "        \"Offices\": [ \"Seattle\" ]"
    + "    },"
    + "    \"V02\": {"
    + "        \"Name\": \"New Publishers, Inc.\","
    + "        \"Offices\": [ \"London\", \"New York\"" + "]" + "},"
    + "    \"V03\": {"
    + "        \"Name\": \"Better Buy Books\","
    +          "\"Offices\": [ \"Tokyo\", \"Los Angeles\", \"Sydney\""
    + "            ]"
    + "        }"
    + "    }";

Item item = new Item()
    .withPrimaryKey("Id", 210)
    .withString("Title", "Book 210 Title")
    .withString("ISBN", "210-2102102102")
    .withNumber("Price", 30)
    .withJSON("VendorInfo", vendorDocument);

PutItemOutcome outcome = table.putItem(item);
```

項目の取得

単一の項目を取り出すには、getItem オブジェクトの Table メソッドを使用します。以下のステップに従ってください。

1. DynamoDB クラスのインスタンスを作成します。
2. 操作対象のテーブルを表すために、Table クラスのインスタンスを作成します。
3. getItem インスタンスの Table メソッドを呼び出します。取り出す項目のプライマリキーを指定する必要があります。

以下の Java コード例は、前述のステップの例です。このコードでは、指定したパーティションキーを持つ項目を取得します。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Item item = table.getItem("Id", 210);
```

オプションパラメータの指定

必須のパラメータに加え、getItem メソッドにはオプションパラメータも指定できます。たとえば、以下の Java コード例では、オプションメソッドを使用して、特定の属性リストのみを取り出します。また、強い整合性のある戻り値をリクエストします。読み込み整合性の詳細については、「読み込み整合性」を参照してください。

ProjectionExpression を使用すると、項目全体ではなく特定の属性または要素のみを取り出すことができます。ProjectionExpression は、ドキュメントのパスを使用して最上位の属性または入れ子になった属性を指定できます。詳細については、「」を参照してくださいプロジェクション式

getItem メソッドのパラメータでは、読み取りの一貫性を指定できません。ただし、GetItemSpec を作成して、低レベル GetItem オペレーションへの入力のすべてに対するフルアクセスを提供できます。以下のコード例では、GetItemSpec を作成し、その仕様を getItem メソッドへの入力として使用します。

Example

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 206)
    .withProjectionExpression("Id, Title, RelatedItems[0], Reviews.FiveStar")
    .withConsistentRead(true);

Item item = table.getItem(spec);
```

```
System.out.println(item.toJSONPretty());
```

Item を人間が読める形式で出力するには、toJSONPretty メソッドを使用します。前のサンプルからの出力は、次のようになります。

```
{
  "RelatedItems" : [ 341 ],
  "Reviews" : {
    "FiveStar" : [ "Excellent! Can't recommend it highly enough! Buy it!", "Do yourself
 a favor and buy this" ]
  },
  "Id" : 123,
  "Title" : "20-Bicycle 123"
}
```

GetItem および JSON ドキュメント

PutItem および JSON ドキュメント セクションでは、Map という名前の VendorInfo 属性に JSON ドキュメントを格納します。getItem メソッドを使用して、ドキュメント全体を JSON 形式で取得できます。または、ドキュメントパスの表記を使用して、ドキュメントの一部の要素のみを取得することもできます。次の Java コード例は、この手法を示しています。

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210);

System.out.println("All vendor info:");
spec.withProjectionExpression("VendorInfo");
System.out.println(table.getItem(spec).toJSON());

System.out.println("A single vendor:");
spec.withProjectionExpression("VendorInfo.V03");
System.out.println(table.getItem(spec).toJSON());

System.out.println("First office location for this vendor:");
spec.withProjectionExpression("VendorInfo.V03.Offices[0]");
System.out.println(table.getItem(spec).toJSON());
```

前のサンプルからの出力は、次のようになります。

```
All vendor info:
```

```
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]},"V02":{"Name":"New Publishers, Inc.","Offices":["London","New
 York"]},"V01":{"Name":"Acme Books","Offices":["Seattle"]}}}
A single vendor:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]}}}
First office location for a single vendor:
{"VendorInfo":{"V03":{"Offices":["Tokyo"]}}}
```

Note

toJSON メソッドを使用して、任意の項目 (またはその属性) を JSON 形式文字列に変換できます。次のコード例は、最上位の属性と入れ子になった属性を複数取り出し、JSON として結果を出力します。

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210)
    .withProjectionExpression("VendorInfo.V01, Title, Price");

Item item = table.getItem(spec);
System.out.println(item.toJSON());
```

出力は次のようになります。

```
{"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
```

バッチ書き込み: 複数の項目の置換および削除

バッチ書き込みは、複数の項目の書き込みと削除をバッチで行うことを意味します。batchWriteItem メソッドによって、単一の コール内にある 1 つまたは複数のテーブルから複数の項目を置換および削除できます。以下に、AWS SDK for Java ドキュメント API を使用して複数の項目を入力および削除する手順を示します。

1. DynamoDB クラスのインスタンスを作成します。
2. テーブルのすべての入力および削除オペレーションを記述する TableWriteItems クラスのインスタンスを作成します。1 つのバッチ書き込みオペレーションで複数のテーブルに書き込む場合、テーブルごとに TableWriteItems インスタンスを 1 つずつ作成する必要があります。

3. 前の手順で作成した batchWriteItem オブジェクトを指定して、TableWriteItems メソッドを呼び出します。
4. 応答を処理します。返された未処理のリクエスト項目が応答内に存在していたかどうかをチェックする必要があります。これは、プロビジョニングされたスループットクォータまたは他の何らかの一時的エラーに達する場合に、発生する可能性があります。また、DynamoDB によって、リクエストのサイズ、およびリクエスト内で指定できるオペレーションの数が制限されます。これらの制限を超えると、DynamoDB によってリクエストが却下されます。詳細については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。

以下の Java コード例は、前述のステップの例です。この例では、2 つのテーブル (batchWriteItem と Forum) で Thread オペレーションを実行します。対応する TableWriteItems オブジェクトは、次のアクションを定義します。

- Forum テーブル内で項目を配置します。
- Thread テーブル内で項目を入力および削除します。

その後、コードは batchWriteItem を呼び出してオペレーションを実行します。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableWriteItems forumTableWriteItems = new TableWriteItems("Forum")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("Name", "Amazon RDS")
            .withNumber("Threads", 0));

TableWriteItems threadTableWriteItems = new TableWriteItems("Thread")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("ForumName","Amazon RDS","Subject","Amazon RDS Thread 1")
    .withHashAndRangeKeysToDelete("ForumName","Some partition key value", "Amazon S3",
 "Some sort key value");

BatchWriteItemOutcome outcome = dynamoDB.batchWriteItem(forumTableWriteItems,
 threadTableWriteItems);

// Code for checking unprocessed items is omitted in this example
```

実例については、「例: AWS SDK for Java ドキュメント API を使用したバッチ書き込みオペレーション」を参照してください。

バッチ取得: 複数の項目の取得

batchGetItem メソッドによって、1 つまたは複数のテーブルから複数の項目を取得できます。単一の項目を取り出すために、getItem メソッドを使用できます。

以下のステップに従ってください。

1. DynamoDB クラスのインスタンスを作成します。
2. テーブルから取り出すプライマリキーのリストを記述する TableKeysAndAttributes クラスのインスタンスを作成します。1 つのバッチ取得オペレーションで複数のテーブルから読み込む場合、テーブルごとに TableKeysAndAttributes インスタンスを 1 つずつ作成する必要があります。
3. 前の手順で作成した batchGetItem オブジェクトを指定して、TableKeysAndAttributes メソッドを呼び出します。

以下の Java コード例は、前述のステップの例です。この例では、Forum テーブル内の 2 つの項目、および Thread テーブル内の 3 つの項目を取り出します。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

    TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
    forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName", "Subject",
    "Amazon DynamoDB","DynamoDB Thread 1",
    "Amazon DynamoDB","DynamoDB Thread 2",
    "Amazon S3","S3 Thread 1");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(
    forumTableKeysAndAttributes, threadTableKeysAndAttributes);
```

```
for (String tableName : outcome.getTableItems().keySet()) {
    System.out.println("Items in table " + tableName);
    List<Item> items = outcome.getTableItems().get(tableName);
    for (Item item : items) {
        System.out.println(item);
    }
}
```

オプションパラメータの指定

必須のパラメータに加え、batchGetItem を使用する場合はオプションパラメータも指定できます。たとえば、定義した ProjectionExpression ごとに TableKeysAndAttributes を指定できます。これにより、テーブルから取り出す属性を指定することができます。

次のサンプルコードでは、Forum から 2 つの項目を取得します。withProjectionExpression パラメータは、Threads 属性のみを取得することを指定します。

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

項目の更新

updateItem オブジェクトの Table メソッドは、既存の属性値の更新、新しい属性の追加、または既存の項目からの属性の削除を実行できます。

updateItem メソッドは、以下のように動作します。

- 項目が存在しない (指定されたプライマリキーを持つ項目がテーブル内にない) 場合、updateItem はテーブルに新しい項目を追加します
- 項目が存在する場合、updateItem は UpdateExpression パラメータで指定されたとおりに更新を実行します。

> **Note**
>
> putItem を使用して項目を「更新」することもできます。たとえば、putItem を呼び出して項目をテーブルに追加したが、指定されたプライマリキーを持つ項目がすでに存在する場合、putItem は項目全体を置き換えます。入力で指定されていない属性が既存の項目内にある場合、putItem は項目からそれらの属性を削除します。
> 一般に、項目属性を変更するときは必ず updateItem を使用することをお勧めします。updateItem メソッドは、入力で指定した項目属性のみを変更し、項目内の他の属性は変更されません。

以下のステップに従ってください。

1. 操作対象のテーブルを表すために、Table クラスのインスタンスを作成します。
2. updateTable インスタンスの Table メソッドを呼び出します。変更する属性とその変更方法を記述する UpdateExpression と同時に、取り出す項目のプライマリーキーを指定する必要があります。

以下の Java コード例は、前述のタスクの例です。このコードでは、ProductCatalog テーブルの書籍項目を更新します。この例では、Authors のセットに著者を追加し、既存の ISBN 属性を削除します。また、価格を 1 引き下げます。

ExpressionAttributeValues マップは UpdateExpression で使用されます。プレースホルダー :val1 および :val2 は、実行時に、Authors と Price の実際の値に置き換えられます。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#A", "Authors");
expressionAttributeNames.put("#P", "Price");
expressionAttributeNames.put("#I", "ISBN");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1",
    new HashSet<String>(Arrays.asList("Author YY","Author ZZ")));
expressionAttributeValues.put(":val2", 1);   //Price
```

```
UpdateItemOutcome outcome =  table.updateItem(
    "Id",          // key attribute name
    101,           // key attribute value
    "add #A :val1 set #P = #P - :val2 remove #I", // UpdateExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

オプションパラメータの指定

必須のパラメータに加えて、更新を実行するために満たす必要がある条件も含めて、updateItem メソッドのオプションパラメータを指定することもできます。指定した条件を満たさない場合は、AWS SDK for Java が ConditionalCheckFailedException をスローします。たとえば、以下の Java コード例では、書籍項目の価格を条件付きで 25 に更新します。現在の価格が 20 である場合にのみ価格を更新する必要があることを示す ConditionExpression を指定します。

Example

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#P", "Price");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1", 25);  // update Price to 25...
expressionAttributeValues.put(":val2", 20);  //...but only if existing Price is 20

UpdateItemOutcome outcome = table.updateItem(
    new PrimaryKey("Id",101),
    "set #P = :val1", // UpdateExpression
    "#P = :val2",     // ConditionExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

アトミックカウンタ

updateItem を使用してアトミックカウンターを実装できます。アトミックカウンターでは、他の書き込みリクエストを妨げることなく、既存の属性の値をインクリメントまたはデクリメントします。アトミックカウンターをインクリメントするには、UpdateExpression を使用して、set アクションで既存の Number 型の属性に数値を加算します。

次のサンプルはこのアトミックカウンターを示しており、Quantity 属性を 1 ずつインクリメントさせています。さらに、ExpressionAttributeNames での UpdateExpression パラメータの使用方法も示します。

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String,String> expressionAttributeNames = new HashMap<String,String>();
expressionAttributeNames.put("#p", "PageCount");

Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", 1);

UpdateItemOutcome outcome = table.updateItem(
    "Id", 121,
    "set #p = #p + :val",
    expressionAttributeNames,
    expressionAttributeValues);
```

項目の削除

deleteItem メソッドによって、テーブルから項目を削除します。削除する項目のプライマリキーを入力する必要があります。

以下のステップに従ってください。

1. DynamoDB クライアントのインスタンスを作成します。
2. 削除する項目のキーを指定して、deleteItem メソッドを呼び出します。

次の Java サンプルは、このタスクを示しています。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

オプションパラメータの指定

deleteItem のオプションパラメータを指定できます。たとえば、次の Java コード例は、ConditionExpression 内の書籍項目は書籍が絶版になった (ProductCatalog 属性が false) 場合のみ削除可能であることを示す InPublication を指定します。

Example

```
Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", false);

DeleteItemOutcome outcome = table.deleteItem("Id",103,
    "InPublication = :val",
    null, // ExpressionAttributeNames - not used in this example
    expressionAttributeValues);
```

例: AWS SDK for Java ドキュメント API を使用した CRUD オペレーション

以下のコード例は、Amazon DynamoDB 項目に対する CRUD オペレーションの例です。この例では、項目の作成、項目の取得、さまざまな更新の実行、さらに項目の削除を行います。

> Note
>
> SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングできます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「Java 1.x: DynamoDBMapper」を参照してください。

> Note
>
> このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
> 以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

```
package com.amazonaws.codesamples.document;
```

```
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DeleteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.UpdateItemOutcome;
import com.amazonaws.services.dynamodbv2.document.spec.DeleteItemSpec;
import com.amazonaws.services.dynamodbv2.document.spec.UpdateItemSpec;
import com.amazonaws.services.dynamodbv2.document.utils.NameMap;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.ReturnValue;

public class DocumentAPIItemCRUDExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ProductCatalog";

    public static void main(String[] args) throws IOException {

        createItems();

        retrieveItem();

        // Perform various updates.
        updateMultipleAttributes();
        updateAddNewAttribute();
        updateExistingAttributeConditionally();

        // Delete the item.
        deleteItem();

    }

    private static void createItems() {
        Table table = dynamoDB.getTable(tableName);
        try {

            Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book
 120 Title")
                    .withString("ISBN", "120-1111111111")
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author12", "Author22")))
                    .withNumber("Price", 20).withString("Dimensions",
 "8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", false).withString("ProductCategory",
 "Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 121).withString("Title", "Book 121
 Title")
                    .withString("ISBN", "121-1111111111")
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author21", "Author 22")))
                    .withNumber("Price", 20).withString("Dimensions",
 "8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
 "Book");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Create items failed.");
            System.err.println(e.getMessage());

        }
    }

    private static void retrieveItem() {
        Table table = dynamoDB.getTable(tableName);

        try {

            Item item = table.getItem("Id", 120, "Id, ISBN, Title, Authors", null);

            System.out.println("Printing item after retrieving it....");
            System.out.println(item.toJSONPretty());

        } catch (Exception e) {
            System.err.println("GetItem failed.");
            System.err.println(e.getMessage());
        }

    }

    private static void updateAddNewAttribute() {
        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
 121)
                    .withUpdateExpression("set #na = :val1").withNameMap(new
 NameMap().with("#na", "NewAttribute"))
                    .withValueMap(new ValueMap().withString(":val1", "Some value"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after adding new attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to add new attribute in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void updateMultipleAttributes() {

        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
 120)
                    .withUpdateExpression("add #a :val1 set #na=:val2")
                    .withNameMap(new NameMap().with("#a", "Authors").with("#na",
 "NewAttribute"))
                    .withValueMap(
                            new ValueMap().withStringSet(":val1", "Author YY", "Author
 ZZ").withString(":val2",
                                    "someValue"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after multiple attribute update...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to update multiple attributes in " + tableName);
            System.err.println(e.getMessage());

        }
    }

    private static void updateExistingAttributeConditionally() {

        Table table = dynamoDB.getTable(tableName);

        try {

            // Specify the desired price (25.00) and also the condition (price =
            // 20.00)

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
 120)
                    .withReturnValues(ReturnValue.ALL_NEW).withUpdateExpression("set #p
 = :val1")
                    .withConditionExpression("#p = :val2").withNameMap(new
 NameMap().with("#p", "Price"))
                    .withValueMap(new ValueMap().withNumber(":val1",
 25).withNumber(":val2", 20));

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after conditional update to new
 attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error updating item in " + tableName);
            System.err.println(e.getMessage());
```

```
        }
    }

    private static void deleteItem() {

        Table table = dynamoDB.getTable(tableName);

        try {

            DeleteItemSpec deleteItemSpec = new DeleteItemSpec().withPrimaryKey("Id",
 120)
                    .withConditionExpression("#ip = :val").withNameMap(new
 NameMap().with("#ip", "InPublication"))
                    .withValueMap(new ValueMap().withBoolean(":val",
 false)).withReturnValues(ReturnValue.ALL_OLD);

            DeleteItemOutcome outcome = table.deleteItem(deleteItemSpec);

            // Check the response.
            System.out.println("Printing item that was deleted...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error deleting item in " + tableName);
            System.err.println(e.getMessage());
        }
    }
}
```

例: AWS SDK for Java ドキュメント API を使用したバッチオペレーション

このセクションでは、AWS SDK for Java ドキュメント API を使用した Amazon DynamoDB でのバッチ書き込みおよびバッチ取得オペレーションの例を示します。

> **Note**
>
> SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングできます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「」を参照してくださいJava 1.x: DynamoDBMapper

トピック

- 例: AWS SDK for Java ドキュメント API を使用したバッチ書き込みオペレーション
- 例: AWS SDK for Java ドキュメント API を使用したバッチ取得オペレーション

例: AWS SDK for Java ドキュメント API を使用したバッチ書き込みオペレーション

以下の Java コード例では、batchWriteItem メソッドを使用して、以下の置換および削除のオペレーションを実行します。

- Forum テーブル内で 1 つの項目を配置します。
- Thread テーブルに対して 1 つの項目を配置および削除します。

バッチの書き込みリクエストを作成すると、1 つまたは複数のテーブルに対して多数の置換リクエストと削除リクエストを指定できます。ただし、batchWriteItem では、1 回のバッチ書き込みオペレーションで可能なバッチ書き込みリクエストのサイズ、置換および削除のオペレーションの数を制限しています。これらの制限を超えるリクエストは却下されます。プロビジョニングされたスループットがテーブルに不足しているためにこのリクエストを処理できない場合は、応答時に未処理のリクエスト項目が返されます。

以下の例では、未処理のリクエスト項目がないか、応答を確認します。未処理のリクエスト項目がある場合は、batchWriteItem リクエストをループバックして再送信します。DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従っていれば、Forum テーブルおよび Thread テーブルは作成済みです。プログラムで、これらのテーブルを作成し、サンプルデータをアップロードすることもできます。詳細については、「」を参照してくださいAWS SDK for Java を使用してサンプルテーブルを作成してデータをアップロードする

以下の例をテストするための詳細な手順については、「Java コードの例」を参照してください。

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchWriteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableWriteItems;
import com.amazonaws.services.dynamodbv2.model.WriteRequest;

public class DocumentAPIBatchWrite {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {

        writeMultipleItemsBatchWrite();

    }

    private static void writeMultipleItemsBatchWrite() {
        try {

            // Add a new item to Forum
            TableWriteItems forumTableWriteItems = new
 TableWriteItems(forumTableName) // Forum
                    .withItemsToPut(new Item().withPrimaryKey("Name", "Amazon
 RDS").withNumber("Threads", 0));

            // Add a new item, and delete an existing item, from Thread
            // This table has a partition key and range key, so need to specify
            // both of them
            TableWriteItems threadTableWriteItems = new
 TableWriteItems(threadTableName)
                    .withItemsToPut(
                            new Item().withPrimaryKey("ForumName", "Amazon RDS",
 "Subject", "Amazon RDS Thread 1")
                                    .withString("Message", "ElastiCache Thread 1
 message")
                                    .withStringSet("Tags", new
 HashSet<String>(Arrays.asList("cache", "in-memory"))))
```

```
                    .withHashAndRangeKeysToDelete("ForumName", "Subject", "Amazon S3",
 "S3 Thread 100");

            System.out.println("Making the request.");
            BatchWriteItemOutcome outcome =
 dynamoDB.batchWriteItem(forumTableWriteItems, threadTableWriteItems);

            do {

                // Check for unprocessed keys which could happen if you exceed
                // provisioned throughput

                Map<String, List<WriteRequest>> unprocessedItems =
 outcome.getUnprocessedItems();

                if (outcome.getUnprocessedItems().size() == 0) {
                    System.out.println("No unprocessed items found");
                } else {
                    System.out.println("Retrieving the unprocessed items");
                    outcome = dynamoDB.batchWriteItemUnprocessed(unprocessedItems);
                }

            } while (outcome.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    }

}
```

例: AWS SDK for Java ドキュメント API を使用したバッチ取得オペレーション

以下の Java コード例では、batchGetItem メソッドを使用して、Forum テーブルおよび Thread テーブルから複数の項目を取り出します。BatchGetItemRequest は、取得する各項目のテーブル名とキーのリストを指定します。この例では、取得した項目を印刷して応答を処理します。

> **Note**
>
> このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
> 以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchGetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableKeysAndAttributes;
import com.amazonaws.services.dynamodbv2.model.KeysAndAttributes;

public class DocumentAPIBatchGet {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {
        retrieveMultipleItemsBatchGet();
    }

    private static void retrieveMultipleItemsBatchGet() {

        try {
            TableKeysAndAttributes forumTableKeysAndAttributes = new
TableKeysAndAttributes(forumTableName);
            // Add a partition key
            forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name", "Amazon S3",
"Amazon DynamoDB");

            TableKeysAndAttributes threadTableKeysAndAttributes = new
TableKeysAndAttributes(threadTableName);
            // Add a partition key and a sort key
            threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName",
"Subject", "Amazon DynamoDB",
                    "DynamoDB Thread 1", "Amazon DynamoDB", "DynamoDB Thread 2",
"Amazon S3", "S3 Thread 1");

            System.out.println("Making the request.");

            BatchGetItemOutcome outcome =
dynamoDB.batchGetItem(forumTableKeysAndAttributes,
                    threadTableKeysAndAttributes);

            Map<String, KeysAndAttributes> unprocessed = null;

            do {
                for (String tableName : outcome.getTableItems().keySet()) {
                    System.out.println("Items in table " + tableName);
                    List<Item> items = outcome.getTableItems().get(tableName);
                    for (Item item : items) {
                        System.out.println(item.toJSONPretty());
                    }
                }

                // Check for unprocessed keys which could happen if you exceed
                // provisioned
                // throughput or reach the limit on response size.
                unprocessed = outcome.getUnprocessedKeys();

                if (unprocessed.isEmpty()) {
                    System.out.println("No unprocessed keys found");
                } else {
                    System.out.println("Retrieving the unprocessed keys");
                    outcome = dynamoDB.batchGetItemUnprocessed(unprocessed);
                }

            } while (!unprocessed.isEmpty());
```

```
        } catch (Exception e) {
            System.err.println("Failed to retrieve items.");
            System.err.println(e.getMessage());
        }

    }

}
```

例: AWS SDK for Java ドキュメント API を使用したバイナリタイプ属性の処理

以下の Java コード例は、バイナリタイプ属性の処理の例です。この例では、Reply テーブルに項目を追加します。この項目には、圧縮データを格納するバイナリタイプ属性 (ExtendedMessage) などがあります。また、この例では、項目を取得し、すべての属性値を印刷します。説明のため、この例では GZIPOutputStream クラスを使用して、サンプルストリームを圧縮し、圧縮したデータを ExtendedMessage 属性に割り当てます。バイナリ属性が取得されると、GZIPInputStream クラスを使用して展開されます。

Note

SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングできます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「」を参照してくださいJava 1.x: DynamoDBMapper

DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従っていれば、Reply テーブルは作成済みです。このテーブルは、プログラムで作成することもできます。詳細については、「」を参照してくださいAWS SDK for Java を使用してサンプルテーブルを作成してデータをアップロードする

以下の例をテストするための詳細な手順については、「Java コードの例」を参照してください。

Example

```
package com.amazonaws.codesamples.document;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.GetItemSpec;

public class DocumentAPIItemBinaryExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";
    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    public static void main(String[] args) throws IOException {
        try {

            // Format the primary key values
            String threadId = "Amazon DynamoDB#DynamoDB Thread 2";

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
            String replyDateTime = dateFormatter.format(new Date());

            // Add a new reply with a binary attribute type
            createItem(threadId, replyDateTime);

            // Retrieve the reply with a binary attribute type
            retrieveItem(threadId, replyDateTime);

            // clean up by deleting the item
            deleteItem(threadId, replyDateTime);
        } catch (Exception e) {
            System.err.println("Error running the binary attribute type example: " + 
e);
            e.printStackTrace(System.err);
        }
    }

    public static void createItem(String threadId, String replyDateTime) throws 
IOException {

        Table table = dynamoDB.getTable(tableName);

        // Craft a long message
        String messageInput = "Long message to be compressed in a lengthy forum reply";

        // Compress the long message
        ByteBuffer compressedMessage = compressString(messageInput.toString());

        table.putItem(new Item().withPrimaryKey("Id", 
threadId).withString("ReplyDateTime", replyDateTime)
                .withString("Message", "Long message 
follows").withBinary("ExtendedMessage", compressedMessage)
                .withString("PostedBy", "User A"));
    }

    public static void retrieveItem(String threadId, String replyDateTime) throws 
IOException {

        Table table = dynamoDB.getTable(tableName);

        GetItemSpec spec = new GetItemSpec().withPrimaryKey("Id", threadId, 
"ReplyDateTime", replyDateTime)
                .withConsistentRead(true);

        Item item = table.getItem(spec);

        // Uncompress the reply message and print
        String uncompressed = 
uncompressString(ByteBuffer.wrap(item.getBinary("ExtendedMessage")));

        System.out.println("Reply message:\n" + " Id: " + item.getString("Id") + "\n" + 
" ReplyDateTime: "
                + item.getString("ReplyDateTime") + "\n" + " PostedBy: " + 
item.getString("PostedBy") + "\n"
                + " Message: "
                + item.getString("Message") + "\n" + " ExtendedMessage (uncompressed):
" + uncompressed + "\n");
    }

    public static void deleteItem(String threadId, String replyDateTime) {

        Table table = dynamoDB.getTable(tableName);
        table.deleteItem("Id", threadId, "ReplyDateTime", replyDateTime);
    }

    private static ByteBuffer compressString(String input) throws IOException {
        // Compress the UTF-8 encoded String into a byte[]
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPOutputStream os = new GZIPOutputStream(baos);
        os.write(input.getBytes("UTF-8"));
        os.close();
        baos.close();
        byte[] compressedBytes = baos.toByteArray();

        // The following code writes the compressed bytes to a ByteBuffer.
        // A simpler way to do this is by simply calling
        // ByteBuffer.wrap(compressedBytes);
        // However, the longer form below shows the importance of resetting the
        // position of the buffer
        // back to the beginning of the buffer if you are writing bytes directly
        // to it, since the SDK
        // will consider only the bytes after the current position when sending
        // data to DynamoDB.
        // Using the "wrap" method automatically resets the position to zero.
        ByteBuffer buffer = ByteBuffer.allocate(compressedBytes.length);
        buffer.put(compressedBytes, 0, compressedBytes.length);
        buffer.position(0); // Important: reset the position of the ByteBuffer
                            // to the beginning
        return buffer;
    }

    private static String uncompressString(ByteBuffer input) throws IOException {
        byte[] bytes = input.array();
        ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPInputStream is = new GZIPInputStream(bais);

        int chunkSize = 1024;
        byte[] buffer = new byte[chunkSize];
        int length = 0;
        while ((length = is.read(buffer, 0, chunkSize)) != -1) {
            baos.write(buffer, 0, length);
        }

        String result = new String(baos.toByteArray(), "UTF-8");

        is.close();
        baos.close();
        bais.close();

        return result;
    }
}
```

項目の操作: .NET

AWS SDK for .NET 低レベル API を使用して、テーブル内の項目に対して、一般的な作成、読み込み、更新、削除 (CRUD) のオペレーションを実行できます。以下に、.NET の低レベル API を使用してデータ CRUD オペレーションを実行するための一般的なステップを示します。

1. AmazonDynamoDBClient クラスのインスタンス（クライアント）を作成します。
2. 対応するリクエストオブジェクトで、オペレーション固有の必須パラメータを指定します。

   たとえば、項目をアップロードするときは PutItemRequest リクエストオブジェクトを使用し、既存の項目を取得するときは GetItemRequest リクエストオブジェクトを使用します。

   リクエストオブジェクトを使用して、必須パラメータとオプションパラメータの両方を指定できます。
3. 前述のステップで作成したリクエストオブジェクトを渡して、クライアントから提供された適切なメソッドを実行します。

   AmazonDynamoDBClient クライアントは、CRUD オペレーションに、PutItem、GetItem、UpdateItem および DeleteItem メソッドを提供します。

トピック

- 項目の置換
- 項目の取得

- 項目の更新
- アトミックカウンタ
- 項目の削除
- バッチ書き込み: 複数の項目の置換および削除
- バッチ取得: 複数の項目の取得
- 例: AWS SDK for .NET 低レベル API を使用した CRUD オペレーション
- 例: AWS SDK for .NET 低レベル API を使用したバッチオペレーション
- 例: AWS SDK for .NET 低レベル API を使用したバイナリタイプ属性の処理

項目の置換

PutItem メソッドは、項目をテーブルにアップロードします。項目が存在する場合、その項目全体が置き換えられます。

> **Note**
>
> 項目全体を置き換える代わりに固有の属性のみを更新する場合は、UpdateItem メソッドを使用できます。詳細については、「」を参照してください項目の更新

以下に、低レベル .NET SDK API を使用して項目をアップロードするステップを示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. PutItemRequest クラスのインスタンスを作成して、必要なパラメータを指定します。

   項目を配置するには、テーブル名と項目を指定する必要があります。
3. 前のステップで作成した PutItemRequest オブジェクトを指定して、PutItem メソッドを実行します。

以下の C# サンプルは、前述のステップの例です。この例では、項目を ProductCatalog テーブルにアップロードします。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";
```

```
var request = new PutItemRequest
{
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
      {
          { "Id", new AttributeValue { N = "201" }},
          { "Title", new AttributeValue { S = "Book 201 Title" }},
          { "ISBN", new AttributeValue { S = "11-11-11-11" }},
          { "Price", new AttributeValue { S = "20.00" }},
          {
            "Authors",
            new AttributeValue
            { SS = new List<string>{"Author1", "Author2"}   }
          }
      }
};
client.PutItem(request);
```

前の例では、Id、Title、ISBN および Authors 属性を持つブック項目をアップロードします。Id は数値型の属性であり、他のすべての属性は文字列型であることに注意してください。作成者は String セットです。

オプションパラメータの指定

次の C# 例に示すように、PutItemRequest オブジェクトを使用してオプションのパラメータを指定することもできます。この例では、次のオプションパラメータが指定されています。

- ExpressionAttributeNames、ExpressionAttributeValues および ConditionExpression は、既存の項目に特定の値を持つ ISBN 属性がある場合にのみ、項目を置き換えるように指定します。
- レスポンスで古い項目をリクエストする ReturnValues パラメータ。

Example

```
var request = new PutItemRequest
 {
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
               {
                   { "Id", new AttributeValue { N = "104" }},
                   { "Title", new AttributeValue { S = "Book 104  Title" }},
                    { "ISBN", new AttributeValue { S = "444-4444444444" }},
                    { "Authors",
                      new AttributeValue { SS = new List<string>{"Author3"}}}
                },
    // Optional parameters.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":isbn",new AttributeValue {S = "444-4444444444"}}
    },
    ConditionExpression = "#I = :isbn"

};
var response = client.PutItem(request);
```

詳細については、「PutItem」を参照してください。

項目の取得

GetItem メソッドは項目を取得します。

> Note
>
> 複数の項目を取り出すために、BatchGetItem メソッドを使用できます。詳細については、「」を参照してくださいバッチ取得: 複数の項目の取得

以下に、低レベル AWS SDK for .NET API を使用して既存の項目を取得するステップを示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. GetItemRequest クラスのインスタンスを作成して、必要なパラメータを指定します。

   項目を吸いtpィするには、テーブル名と項目のプライマリキーを指定する必要があります。
3. 前のステップで作成した GetItemRequest オブジェクトを指定して、GetItem メソッドを実行します。

以下の C# サンプルは、前述のステップの例です。例では、ProductCatalog テーブルから項目を取得します。

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
 };
 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item; // Attribute list in the response.
```

オプションパラメータの指定

次の C# 例に示すように、`GetItemRequest` オブジェクトを使用してオプションのパラメータを指定することもできます。このサンプルでは、次のオプションパラメータが指定されています。

- 取得する属性を指定する `ProjectionExpression` パラメータ。
- 強力な整合性のある読み込みを実行する `ConsistentRead` パラメータ。読み込み整合性の詳細については、「読み込み整合性」を参照してください。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
   // Optional parameters.
   ProjectionExpression = "Id, ISBN, Title, Authors",
   ConsistentRead = true
 };

 var response = client.GetItem(request);
```

```
// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item;
```

詳細については、「GetItem」を参照してください。

項目の更新

UpdateItem メソッドは、既存の項目があればそれを更新します。UpdateItem オペレーションを使用して、既存の属性値を更新するか、新しい属性を追加するか、既存のコレクションから属性を削除することができます。プライマリキーが指定されている項目がない場合は、新しい項目が追加されます。

UpdateItem オペレーションは、以下のガイドラインに従います。

- 項目が存在しない場合、UpdateItem は入力で指定されたプライマリキーを使用して、新しい項目を追加します。
- 項目が存在する場合、UpdateItem は次のように更新を適用します。
  - 既存の属性値を更新による値に置き換えます。
  - 入力で指定された属性が存在しない場合は、新しい属性を項目に追加します。
  - 入力された属性が null である場合は、属性を削除します。
  - ADD に Action を使用する場合は、既存のセット (文字列または数値セット) に値を追加するか、既存の数値属性値から数学的に加算 (正の数を使用) または減算 (負の数を使用) することができます。

> **Note**
>
> PutItem オペレーションにより、更新を実行できます。詳細については、「」を参照してください項目の置換 たとえば、PutItem をコールして項目をアップロードするときにプライマリキーが存在する場合は、PutItem オペレーションによって項目全体が置き換わります。既存の項目内に属性があり、入力でそれらの属性が指定されていない場合、それらの属性は、PutItem オペレーションによって削除されます。ただし、UpdateItem が更新するのは指定された入力属性だけです。その項目では、その他の既存の属性は変更されません。

以下に、低レベル .NET SDK API を使用して既存の項目を更新するステップを示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。

2. UpdateItemRequest クラスのインスタンスを作成して、必要なパラメータを指定します。

   これは、属性の追加、既存の属性の更新、属性の削除など、すべての更新を記述するリクエストオブジェクトです。既存の属性を削除するには、その属性名に null 値を指定します。

3. 前のステップで作成した UpdateItemRequest オブジェクトを指定して、UpdateItem メソッドを実行します。

以下の C# サンプルコードは、前述のステップの例です。例では、ProductCatalog テーブルのブック項目を更新します。この例では、Authors のコレクションに著者を追加し、既存の ISBN 属性を削除します。また、価格を 1 引き下げます。

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#A", "Authors"},
        {"#P", "Price"},
        {"#NA", "NewAttribute"},
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":auth",new AttributeValue { SS = {"Author YY","Author ZZ"}}},
        {":p",new AttributeValue {N = "1"}},
        {":newattr",new AttributeValue {S = "someValue"}},
    },

    // This expression does the following:
    // 1) Adds two new authors to the list
    // 2) Reduces the price
    // 3) Adds a new attribute to the item
    // 4) Removes the ISBN attribute from the item
    UpdateExpression = "ADD #A :auth SET #P = #P - :p, #NA = :newattr REMOVE #I"
};
```

```
var response = client.UpdateItem(request);
```

オプションパラメータの指定

次の C# 例に示すように、UpdateItemRequest オブジェクトを使用してオプションのパラメータを指定することもできます。次のオプションパラメータが指定されています。

- ExpressionAttributeValues および ConditionExpression は、既存の料金が 20.00 である場合にのみ料金を更新できるように指定します。
- レスポンスで更新された項目をリクエストする ReturnValues パラメータ。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },

    // Update price only if the current price is 20.00.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#P", "Price"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":newprice",new AttributeValue {N = "22"}},
        {":currprice",new AttributeValue {N = "20"}}
    },
    UpdateExpression = "SET #P = :newprice",
    ConditionExpression = "#P = :currprice",
    TableName = tableName,
    ReturnValues = "ALL_NEW" // Return all the attributes of the updated item.
};

var response = client.UpdateItem(request);
```

詳細については、「UpdateItem」を参照してください。

アトミックカウンタ

updateItem を使用してアトミックカウンターを実装できます。アトミックカウンターでは、他の書き込みリクエストを妨げることなく、既存の属性の値をインクリメントまたはデクリメントします。アトミックカウンタを更新するには、UpdateExpression パラメータにタイプに Number の属性を持つ updateItem および ADD を、Action として使用します。

次のサンプルはこのアトミックカウンターを示しており、Quantity 属性を 1 ずつインクリメントさせています。

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string, AttributeValue>() { { "Id", new AttributeValue { N =
 "121" } } },
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#Q", "Quantity"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":incr",new AttributeValue {N = "1"}}
    },
    UpdateExpression = "SET #Q = #Q + :incr",
    TableName = tableName
};

var response = client.UpdateItem(request);
```

項目の削除

DeleteItem メソッドによって、テーブルから項目を削除します。

以下では、低レベル .NET SDK API を使用して項目を削除するステップを示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. DeleteItemRequest クラスのインスタンスを作成して、必要なパラメータを指定します。

   項目を削除するには、テーブル名と項目のプライマリキーが必要です。

3. 前のステップで作成した DeleteItemRequest オブジェクトを指定して、DeleteItem メソッドを実行します。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },
};

var response = client.DeleteItem(request);
```

オプションパラメータの指定

次の C# コード例に示すように、DeleteItemRequest オブジェクトを使用して任意のパラメータを指定することもできます。次のオプションパラメータが指定されています。

- ExpressionAttributeValues および ConditionExpression は、ブック項目が公開されていない場合にのみ削除できるよう指定します (InPublication 属性値は false です)。
- レスポンスで削除された項目をリクエストする ReturnValues パラメータ。

Example

```
var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },

    // Optional parameters.
    ReturnValues = "ALL_OLD",
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#IP", "InPublication"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":inpub",new AttributeValue {BOOL = false}}
    },
    ConditionExpression = "#IP = :inpub"
};

var response = client.DeleteItem(request);
```

詳細については、「DeleteItem」を参照してください。

バッチ書き込み: 複数の項目の置換および削除

バッチ書き込みは、複数の項目の書き込みと削除をバッチで行うことを意味します。BatchWriteItem メソッドによって、単一の コール内にある 1 つまたは複数のテーブルから複数の項目を置換および削除できます。以下に、低レベル .NET SDK API を使用して複数の項目を取得するステップを示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. BatchWriteItemRequest クラスのインスタンスを作成して、すべての入力および削除オペレーションを記述します。
3. 前のステップで作成した BatchWriteItemRequest オブジェクトを指定して、BatchWriteItem メソッドを実行します。
4. 応答を処理します。返された未処理のリクエスト項目が応答内に存在していたかどうかをチェックする必要があります。これは、プロビジョニングされたスループットクォータまたは他の何らかの一時的エラーに達する場合に、発生する可能性があります。また、DynamoDB によって、リクエストのサイズ、およびリクエスト内で指定できるオペレーションの数が制限されます。これらの制限を超えると、DynamoDB によってリクエストが却下されます。詳細については、「BatchWriteItem」を参照してください。

以下の C# サンプルコードは、前述のステップの例です。この例では、BatchWriteItemRequest を作成して次の書き込みオペレーションを実行します。

- Forum テーブル内で項目を配置します。
- Thread テーブルから項目を入力および削除します。

このコードは BatchWriteItem を実行してバッチ操作を実行します。

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchWriteItemRequest
 {
   RequestItems = new Dictionary<string, List<WriteRequest>>
    {
      {
        table1Name, new List<WriteRequest>
        {
          new WriteRequest
          {
             PutRequest = new PutRequest
             {
                Item = new Dictionary<string,AttributeValue>
                {
                  { "Name", new AttributeValue { S = "Amazon S3 forum" } },
                  { "Threads", new AttributeValue { N = "0" }}
                }
             }
          }
        }
      } ,
      {
        table2Name, new List<WriteRequest>
        {
          new WriteRequest
          {
            PutRequest = new PutRequest
            {
               Item = new Dictionary<string,AttributeValue>
               {
                 { "ForumName", new AttributeValue { S = "Amazon S3 forum" } },
                 { "Subject", new AttributeValue { S = "My sample question" } },
                 { "Message", new AttributeValue { S = "Message Text." } },
                 { "KeywordTags", new AttributeValue { SS = new List<string> { "Amazon
 S3", "Bucket" }  } }
               }
            }
          },
          new WriteRequest
          {
             DeleteRequest = new DeleteRequest
```

```
                {
                   Key = new Dictionary<string,AttributeValue>()
                   {
                      { "ForumName", new AttributeValue { S = "Some forum name" } },
                      { "Subject", new AttributeValue { S = "Some subject" } }
                   }
                }
             }
           }
         }
      }
 };
response = client.BatchWriteItem(request);
```

実例については、「例: AWS SDK for .NET 低レベル API を使用したバッチオペレーション」を参照してください。

バッチ取得: 複数の項目の取得

BatchGetItem メソッドによって、1 つまたは複数のテーブルから複数の項目を取得できます。

> Note
>
> 単一の項目を取り出すために、GetItem メソッドを使用できます。

以下に、低レベル AWS SDK for .NET API を使用して複数の項目を取得するステップを示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. BatchGetItemRequest クラスのインスタンスを作成して、必要なパラメータを指定します。

   複数の項目を取得するには、テーブル名とプライマリキーバリューのリストが必要です。
3. 前のステップで作成した BatchGetItemRequest オブジェクトを指定して、BatchGetItem メソッドを実行します。
4. 応答を処理します。未処理キーがなかったかどうかを確認する必要があります。これは、プロビジョニングされたスループットクォータまたは他の何らかの一時的エラーに達する場合に発生する可能性があります。

以下の C# サンプルコードは、前述のステップの例です。この例では、Forum および Thread の 2 つのテーブルから取得します。リクエストでは、Forum テーブルに 2 つの項目、Thread テーブル

に 3 つの項目を指定します。応答には、両方のテーブルの項目が含まれます。このコードは、応答を処理する方法を示しています。

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      }
    },
    {
      table2Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 1" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 2" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "Amazon S3" } },
            { "Subject", new AttributeValue { S = "Amazon S3 Thread 1" } }
          }
        }
      }
    }
  }
};

var response = client.BatchGetItem(request);

// Check the response.
var result = response.BatchGetItemResult;
var responses = result.Responses; // The attribute list in the response.

var table1Results = responses[table1Name];
Console.WriteLine("Items in table {0}" + table1Name);
foreach (var item1 in table1Results.Items)
{
  PrintItem(item1);
}

var table2Results = responses[table2Name];
Console.WriteLine("Items in table {1}" + table2Name);
foreach (var item2 in table2Results.Items)
{
  PrintItem(item2);
}
// Any unprocessed keys? could happen if you exceed ProvisionedThroughput or some other
 error.
Dictionary<string, KeysAndAttributes> unprocessedKeys = result.UnprocessedKeys;
foreach (KeyValuePair<string, KeysAndAttributes> pair in unprocessedKeys)
{
    Console.WriteLine(pair.Key, pair.Value);
}
```

オプションパラメータの指定

次の C# コード例に示すように、BatchGetItemRequest オブジェクトを使用して任意のパラメータを指定することもできます。例では、Forum テーブルから 2 つの項目を取得します。次のオプションパラメータが指定されています。

- 取得する属性を指定する ProjectionExpression パラメータ。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      },
      // Optional - name of an attribute to retrieve.
      ProjectionExpression = "Title"
    }
  }
};

var response = client.BatchGetItem(request);
```

詳細については、BatchGetItem を参照してください。

例: AWS SDK for .NET 低レベル API を使用した CRUD オペレーション

以下の C# コード例は、Amazon DynamoDB 項目に対する CRUD オペレーションの例です。この例では、ProductCatalog テーブルへの項目の追加、項目の取得、さまざまな更新の実行、さらに項目の削除を行います。DynamoDB でのコード例用のテーブルの作成とデータのロード のステップに従っていれば、ProductCatalog テーブルは作成済みです。これらのサンプルテーブルは、プログラムで作成することもできます。詳細については、「」を参照してくださいAWS SDK for .NET を使用してサンプルテーブルを作成してデータをアップロードする

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelItemCRUDExample
    {
        private static string tableName = "ProductCatalog";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                CreateItem();
                RetrieveItem();

                // Perform various updates.
                UpdateMultipleAttributes();
                UpdateExistingAttributeConditionally();

                // Delete item.
                DeleteItem();
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void CreateItem()
        {
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  }},
                { "Title", new AttributeValue {
                      S = "Book 201 Title"
                  }},
                { "ISBN", new AttributeValue {
                      S = "11-11-11-11"
                  }},
                { "Authors", new AttributeValue {
                      SS = new List<string>{"Author1", "Author2" }
                  }},
                { "Price", new AttributeValue {
                      N = "20.00"
                  }},
                { "Dimensions", new AttributeValue {
                      S = "8.5x11.0x.75"
                  }},
                { "InPublication", new AttributeValue {
                      BOOL = false
                  } }
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem()
        {
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ProjectionExpression = "Id, ISBN, Title, Authors",
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");
            PrintItem(attributeList);
        }

        private static void UpdateMultipleAttributes()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                // Perform the following updates:
                // 1) Add two new authors to the list
                // 1) Set a new attribute
                // 2) Remove the ISBN attribute
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#A","Authors"},
                {"#NA","NewAttribute"},
                {"#I","ISBN"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":auth",new AttributeValue {
                     SS = {"Author YY", "Author ZZ"}
                 }},
                {":new",new AttributeValue {
                     S = "New Value"
                 }}
            },

                UpdateExpression = "ADD #A :auth SET #NA = :new REMOVE #I",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
                                                     // print attributeList.
            Console.WriteLine("\nPrinting item after multiple attribute
 update ............");
            PrintItem(attributeList);
        }

        private static void UpdateExistingAttributeConditionally()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#P", "Price"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":newprice",new AttributeValue {
                     N = "22.00"
                 }},
                {":currprice",new AttributeValue {
                     N = "20.00"
                 }}
            },
                // This updates price only if current price is 20.00.
```

```
                UpdateExpression = "SET #P = :newprice",
                ConditionExpression = "#P = :currprice",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
            Console.WriteLine("\nPrinting item after updating price value
 conditionally ...........");
            PrintItem(attributeList);
        }

        private static void DeleteItem()
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },

                // Return the entire item as it appeared before the update.
                ReturnValues = "ALL_OLD",
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#IP", "InPublication"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":inpub",new AttributeValue {
                     BOOL = false
                 }}
            },
                ConditionExpression = "#IP = :inpub"
            };

            var response = client.DeleteItem(request);
            // Check the response.
            var attributeList = response.Attributes; // Attribute list in the response.
                                                     // Print item.
            Console.WriteLine("\nPrinting item that was just deleted ...........");
            PrintItem(attributeList);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

例: AWS SDK for .NET 低レベル API を使用したバッチオペレーション

トピック

- 例: AWS SDK for .NET 低レベル API を使用したバッチ書き込みオペレーション
- 例: AWS SDK for .NET 低レベル API を使用したバッチ取得オペレーション

このセクションでは、Amazon DynamoDB がサポートするバッチ操作、バッチ書き込み、およびバッチ取得の例を示します。

例: AWS SDK for .NET 低レベル API を使用したバッチ書き込みオペレーション

以下の C# コード例では、BatchWriteItem メソッドを使用して、以下の置換および削除のオペレーションを実行します。

- Forum テーブル内で 1 つの項目を配置します。
- Thread テーブルに対して 1 つの項目を配置および削除します。

バッチの書き込みリクエストを作成すると、1 つまたは複数のテーブルに対して多数の置換リクエストと削除リクエストを指定できます。ただし、DynamoDB BatchWriteItem では、1 回のバッチ書き込みオペレーションで可能なバッチ書き込みリクエストのサイズ、置換および削除のオペレーションの数を制限しています。詳細については、「BatchWriteItem」を参照してください。これらの制限を超えるリクエストは却下されます。プロビジョニングされたスループットがテーブルに不足しているためにこのリクエストを処理できない場合は、応答時に未処理のリクエスト項目が返されます。

以下の例では、未処理のリクエスト項目がないか、応答を確認します。未処理のリクエスト項目がある場合は、BatchWriteItem リクエストをループバックして再送信します。DynamoDB でのコード例用のテーブルの作成とデータのロード のステップに従っていれば、Forum および Thread テーブルは作成済みです。プログラムで、これらのサンプルテーブルを作成し、サンプルデータをアップロードすることもできます。詳細については、「」を参照してくださいAWS SDK for .NET を使用してサンプルテーブルを作成してデータをアップロードする

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchWrite
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                TestBatchWrite();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void TestBatchWrite()
        {
            var request = new BatchWriteItemRequest
            {
                ReturnConsumedCapacity = "TOTAL",
                RequestItems = new Dictionary<string, List<WriteRequest>>
            {
                {
                    table1Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "Name", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Threads", new AttributeValue {
                                          N = "0"
                                      }}
                                }
                            }
                        }
                    }
                },
                {
                    table2Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "ForumName", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "My sample question"
                                      } },
                                    { "Message", new AttributeValue {
                                          S = "Message Text."
                                      } },
                                    { "KeywordTags", new AttributeValue {
                                          SS = new List<string> { "S3", "Bucket" }
                                      } }
                                }
                            }
                        },
                        new WriteRequest
                        {
                            // For the operation to delete an item, if you provide a
primary key value
                            // that does not exist in the table, there is no error, it
is just a no-op.
                            DeleteRequest = new DeleteRequest
                            {
                                Key = new Dictionary<string, AttributeValue>()
                                {
                                    { "ForumName",  new AttributeValue {
                                          S = "Some partition key value"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "Some sort key value"
                                      } }
                                }
                            }
                        }
                    }
                }
            }
            };

            CallBatchWriteTillCompletion(request);
        }

        private static void CallBatchWriteTillCompletion(BatchWriteItemRequest request)
        {
            BatchWriteItemResponse response;
            int callCount = 0;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchWriteItem(request);
                callCount++;

                // Check the response.

                var tableConsumedCapacities = response.ConsumedCapacity;
                var unprocessed = response.UnprocessedItems;

                Console.WriteLine("Per-table consumed capacity");
                foreach (var tableConsumedCapacity in tableConsumedCapacities)
                {
                    Console.WriteLine("{0} - {1}", tableConsumedCapacity.TableName,
 tableConsumedCapacity.CapacityUnits);
                }

                Console.WriteLine("Unprocessed");
                foreach (var unp in unprocessed)
                {
                    Console.WriteLine("{0} - {1}", unp.Key, unp.Value.Count);
                }
                Console.WriteLine();

                // For the next iteration, the request will have unprocessed items.
                request.RequestItems = unprocessed;
            } while (response.UnprocessedItems.Count > 0);

            Console.WriteLine("Total # of batch write API calls made: {0}", callCount);
        }
    }
}
```

例: AWS SDK for .NET 低レベル API を使用したバッチ取得オペレーション

次の C# コード例では、`BatchGetItem` メソッドを使用して、Amazon DynamoDB の `Forum` テーブルおよび `Thread` テーブルから複数の項目を取り出します。`BatchGetItemRequest` は、各テーブルのテーブル名とプライマリキーのリストを指定します。この例では、取得した項目を印刷して応答を処理します。

DynamoDB でのコード例用のテーブルの作成とデータのロード のステップに従っていれば、サンプルデータを使用してすでにこれらのテーブルが作成済みです。プログラムで、これらのサンプルテーブルを作成し、サンプルデータをアップロードすることもできます。詳細については、「」を参照してくださいAWS SDK for .NET を使用してサンプルテーブルを作成してデータをアップロードする

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchGet
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                RetrieveMultipleItemsBatchGet();

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void RetrieveMultipleItemsBatchGet()
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                        } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
                        } }
                          }
                      }
                  }},
                {
                    table2Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
                        {
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 1"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchGetItem(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
 tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
 ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
 response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

例: AWS SDK for .NET 低レベル API を使用したバイナリタイプ属性の処理

以下の C# コード例は、バイナリタイプ属性の処理の例です。この例では、Reply テーブルに項目を追加します。この項目には、圧縮データを格納するバイナリタイプ属性 (ExtendedMessage) などがあります。また、この例では、項目を取得し、すべての属性値を印刷します。説明のため、この例では GZipStream クラスを使用して、サンプルストリームを圧縮し、圧縮したデータを ExtendedMessage 属性に割り当て、属性値を出力するときに解凍します。

DynamoDB でのコード例用のテーブルの作成とデータのロード のステップに従っていれば、Reply テーブルは作成済みです。これらのサンプルテーブルは、プログラムで作成することもできます。詳細については、「」を参照してくださいAWS SDK for .NET を使用してサンプルテーブルを作成してデータをアップロードする

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

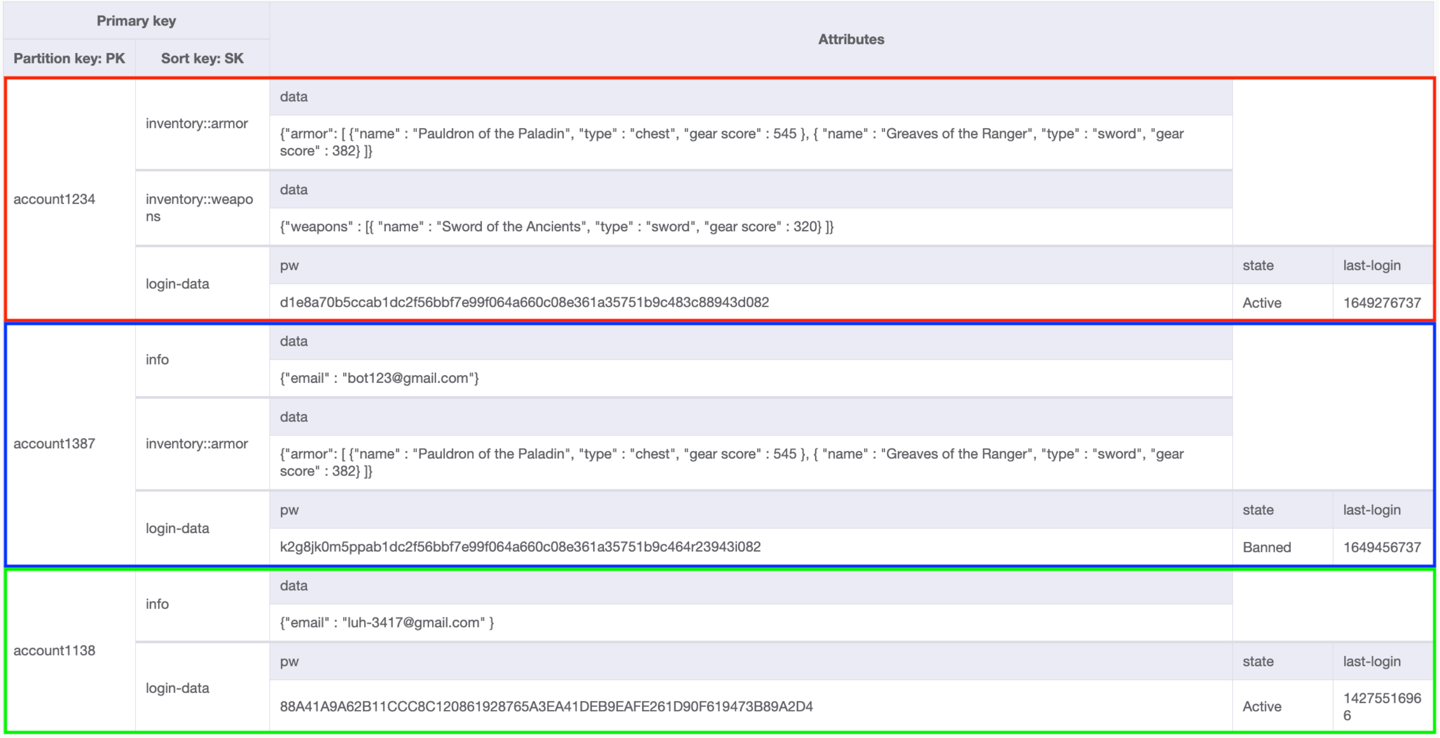
```
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelItemBinaryExample
    {
        private static string tableName = "Reply";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            // Reply table primary key.
            string replyIdPartitionKey = "Amazon DynamoDB#DynamoDB Thread 1";
            string replyDateTimeSortKey = Convert.ToString(DateTime.UtcNow);

            try
            {
                CreateItem(replyIdPartitionKey, replyDateTimeSortKey);
                RetrieveItem(replyIdPartitionKey, replyDateTimeSortKey);
                // Delete item.
                DeleteItem(replyIdPartitionKey, replyDateTimeSortKey);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateItem(string partitionKey, string sortKey)
        {
            MemoryStream compressedMessage = ToGzipMemoryStream("Some long extended
 message to compress.");
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  }},
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  }},
                { "Subject", new AttributeValue {
                      S = "Binary type "
                  }},
                { "Message", new AttributeValue {
                      S = "Some message about the binary type"
                  }},
                { "ExtendedMessage", new AttributeValue {
                      B = compressedMessage
                  }}
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem(string partitionKey, string sortKey)
        {
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            },
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");

            PrintItem(attributeList);
        }

        private static void DeleteItem(string partitionKey, string sortKey)
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            }
            };
            var response = client.DeleteItem(request);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]") +
                    (value.B == null ? "" : "B=[" + FromGzipMemoryStream(value.B) +
 "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
        private static MemoryStream ToGzipMemoryStream(string value)
        {
            MemoryStream output = new MemoryStream();
            using (GZipStream zipStream = new GZipStream(output,
 CompressionMode.Compress, true))
            using (StreamWriter writer = new StreamWriter(zipStream))
            {
                writer.Write(value);
            }
            return output;
        }

        private static string FromGzipMemoryStream(MemoryStream stream)
        {
            using (GZipStream zipStream = new GZipStream(stream,
 CompressionMode.Decompress))
            using (StreamReader reader = new StreamReader(zipStream))
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

項目コレクション - DynamoDB で一対多リレーションシップをモデル化する方法

DynamoDB の項目コレクションは、同じパーティションキー値を共有する項目のグループです。これは、これらの項目が関連していることを意味します。項目コレクションは、DynamoDB で 1 対多リレーションシップをモデル化する主要なメカニズムです。項目コレクションは、複合プライマリキーを使用するように設定されたテーブルまたはインデックスにのみ存在できます。

> ⓘ Note
>
> 項目コレクションは、ベーステーブルまたはセカンダリインデックスのいずれかに存在できます。項目コレクションが特にインデックスとやり取りする方法の詳細については、「ローカルセカンダリインデックス内の項目コレクション」を参照してください。

3 つの異なるユーザーとそのゲーム内インベントリを示す次のテーブルを考えてみます。



Primary key		Attributes		
Partition key: PK	Sort key: SK			
account1234	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	inventory::weapons	data		
		{"weapons" : [{ "name" : "Sword of the Ancients", "type" : "sword", "gear score" : 320}]}		
	login-data	pw	state	last-login
		d1e8a70b5ccab1dc2f56bbf7e99f064a660c08e361a35751b9c483c88943d082	Active	1649276737
account1387	info	data		
		{"email" : "bot123@gmail.com"}		
	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	login-data	pw	state	last-login
		k2g8jk0m5ppab1dc2f56bbf7e99f064a660c08e361a35751b9c464r23943i082	Banned	1649456737
account1138	info	data		
		{"email" : "luh-3417@gmail.com" }		
	login-data	pw	state	last-login
		88A41A9A62B11CCC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4	Active	14275516966

各コレクションの一部の項目において、ソートキーは、`inventory::armor`、`inventory::weapon`、`info` など、データをグループ化するための情報を連結したものです。項目コレクションごとに、これらの属性の異なる組み合わせをソートキーとして使用できます。ユーザー `account1234` には `inventory::weapons` 項目がありますが、ユーザー `account1387` にはありません (まだ見つけていないため)。ユーザー `account1138` は、ソートキーとして 2 つの項目のみ使用しています (まだインベントリがないため)。他のユーザーは 3 つを使用しています。

DynamoDB では、これらの項目コレクションから項目を選択的に取得して、次の操作を実行できます。

- 特定のユーザーからすべての項目を取得する。
- 特定のユーザーから 1 つの項目のみを取得する。
- 特定のユーザーに属する特定タイプのすべての項目を取得する。

項目コレクションでデータを整理してクエリを高速化する

この例では、これらの 3 つの項目コレクションの各項目は、プレイヤーと選択したデータモデルを表し、ゲームとプレイヤーのアクセスパターンを反映しています。どのようなデータがゲームに必要であるか。いつ必要であるか。どのくらいの頻度で必要であるか。この方法で実行する場合のコスト

はどれくらいか。これらの質問に対する回答に基づいて、これらのデータモデリングを決定しています。

このゲームでは、武器のインベントリのページと鎧のページが別個にプレイヤーに提示されます。プレイヤーがインベントリを開くと、最初に武器が表示されます。これは、このページを高速にロードしたいためであり、以降のインベントリページはその後にロードできます。これらの各項目タイプは、プレイヤーが獲得するゲーム内項目が増えるに従って肥大化する可能性があるため、データベースでは各インベントリページをプレイヤーの項目コレクションの独立した項目にしています。

次のセクションでは、Query オペレーションを使用して項目コレクションとやり取りする方法についてより詳しく説明します。

トピック

DynamoDB のクエリオペレーション

Amazon DynamoDB の Query オペレーションを使用して、プライマリキーの値に基づいて項目を探すことができます。

パーティションキーの属性名、および属性の単一値を指定する必要があります。Query はそのパーティションキー値を持つすべての項目を返します。必要に応じて、ソートキーの属性を指定し、比較演算子を使用して、検索結果をさらに絞り込むことができます。

リクエストの構文、レスポンスパラメータ、その他の例など、Query の使用方法の詳細については、Amazon DynamoDB API リファレンスの「クエリ」を参照してください。

トピック

クエリオペレーションのキー条件式

検索条件を指定するには、キー条件式 (テーブルまたはインデックスから読み取る項目を決定する文字列) を使用します。

等価条件としてパーティションキーの名前と値を指定する必要があります。キー条件式では、非キー属性を使用することはできません。

オプションで、ソートキーに 2 番目の条件を指定できます (存在する場合)。ソートキーの条件では、次の比較演算子を 1 つ使用する必要があります。

- *a* `=` *b* — 属性 *a* が値 *b* と等しい場合、true
- *a* `<` *b* — *a* が *b* 未満の場合、true
- *a* `<=` *b* — *a* が *b* 以下である場合、true
- *a* `>` *b* — *a* が *b* より大きい場合、true
- *a* `>=` *b* — *a* が *b* 以上である場合、true
- *a* `BETWEEN` *b* `AND` *c* — *a* が *b* 以上で、*c* 以下である場合、true。

次の関数もサポートされます。

- `begins_with (`*a*`,` *substr*`)` — 属性の値 *a* が特定のサブ文字列から始まる場合、true。

キー条件式の使用方法を示す AWS Command Line Interface (AWS CLI) の例を次に示します。これらの式では、実際の値の代わりにプレースホルダー (`:name` や `:sub` など) を使用しています。詳細については、「DynamoDB の式の属性名」および「式の属性値」を参照してください。

Example

`Thread` テーブルに対して、特定の `ForumName` (パーティションキー) についてのクエリを実行します。その `ForumName` の値を持つすべての項目はクエリによって読み込まれます。これはソートキー (`Subject`) が `KeyConditionExpression` に含まれないためです。

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Thread テーブルに対して、特定の ForumName (パーティションキー) についてのクエリを実行しますが、今回は指定の Subject (ソートキー) を持つ項目のみを返します。

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

--expression-attribute-values の引数は、values.json のファイルに保存されます。

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

Reply テーブルに対して、特定の Id (パーティションキー) についてのクエリを実行しますが、ReplyDateTime (ソートキー) が特定の文字で始まる項目のみを返します。

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

--expression-attribute-values の引数は、values.json のファイルに保存されます。

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
}
```

最初の文字が a-z または A-Z であり、残りの文字 (ある場合、2 番目以降の文字) が a-z、A-Z、または 0-9 である場合は、キー条件式で任意の属性値を使用できます。さらに、属性名は DynamoDB の予約語であってはいけません。（詳細リストについては、「DynamoDB の予約語」を参照してください）属性名がこれらの要件を満たさない場合は、式の属性名をプレースホルダーとして定義する必要があります。詳細については、「DynamoDB の式の属性名」を参照してください。

特定のパーティションキー値を持つ項目は、DynamoDB によってソートキーの値で並べ替えられた順序で近くに配置されて保存されます。Query オペレーションでは、並べ替えられた順序でDynamoDB によって項目が取得され、KeyConditionExpression や存在する任意の FilterExpression を使用して処理されます。その時初めて、Query の結果がクライアントに返されます。

Query オペレーションは常に結果セットを返します。一致する項目がない場合、結果セットは空になります。

Query の結果は常にソートキーの値によってソートされます。ソートキーのデータ型が Number である場合は、結果が番号順で返されます。その他の場合は、UTF-8 バイトの順序で結果が返されます。デフォルトの並べ替え順序は昇順です。順序を反転させるには、ScanIndexForward パラメータを false に設定します。

1 回の Query オペレーションで、最大 1 MB のデータを取得できます。この制限は、FilterExpression または ProjectionExpression が結果に適用される前に適用されます。レスポンスに LastEvaluatedKey が存在し、Null 以外の場合、結果セットをページ分割する必要があります (テーブルクエリ結果をページ分割する を参照)。

クエリオペレーションのフィルター式

Query 結果の絞り込みが必要な場合は、オプションでフィルタ式を指定できます。フィルタ式によって、Query 結果の内、どの項目を返すべきか確定します。他のすべての結果は破棄されます。

フィルタ式は、Query の完了後、結果が返される前に適用されます。そのため、Query は、フィルタ式があるかどうかにかかわらず、同じ量の読み込みキャパシティーを消費します。

1 回の Query オペレーションで、最大 1 MB のデータを取得できます。この制限は、フィルタ式を評価する前に適用されます。

フィルタ式には、パーティションキーまたはソートキーの属性を含めることはできません。フィルタ式ではなく、キー条件式でこれらの属性を指定する必要があります。

フィルター式の構文は、キー条件式の構文と似ています。フィルター式は、キー条件式と同じコンパレータ、関数および論理演算子を使用できます。さらに、フィルター式では、非等号演算子 (<>)、OR 演算子、CONTAINS 演算子、IN 演算子、BEGINS_WITH 演算子、BETWEEN 演算子、EXISTS 演算子、および SIZE 演算子を使用できます。詳細については、「クエリオペレーションのキー条件式」および「フィルター式と条件式の構文」を参照してください。

Example

以下の AWS CLI の例では、Thread テーブルに対して、特定の ForumName (パーティションキー) および Subject (ソートキー) についてのクエリを実行します。見つかった項目のうち、最も一般的なディスカッションスレッド (つまり、一定数以上の Views があるスレッド) のみが返されます。

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject = :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

--expression-attribute-values の引数は、values.json のファイルに保存されます。

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Views は DynamoDB で予約語であるため (DynamoDB の予約語 を参照)、この例では #v をプレースホルダーとして使用することにご注意ください。詳細については、「DynamoDB の式の属性名」を参照してください。

Note

フィルタ式は、Query 結果セットから項目を削除します。可能であれば、大量の項目を取得してもそのほとんどを破棄する必要がある場合、 Query の使用は避けてください。

テーブルクエリ結果をページ分割する

DynamoDB では、Query オペレーションの結果をページ割りします。ページ分割を行うことで Query 結果が 1 MB サイズ (またはそれ以下) のデータの「ページ」に分割されます。アプリケーションは結果の最初のページ、次に 2 ページと処理できます。

1 つの Query は、サイズの制限である1 MB 以内の結果セットだけを返します。さらに結果があるかどうかを確認して、一度に 1 ページずつ結果を取り出すには、アプリケーションで次の操作を行う必要があります。

1. 低レベルの `Query` 結果を確認します。
   - 結果に `LastEvaluatedKey` 要素が含まれており、それが Null 以外の場合、ステップ 2 に進みます。
   - 結果に `LastEvaluatedKey` がない場合、これ以上取得する項目はありません。
2. 以前のものと同じパラメータを使用して新しい `Query` リクエストを作成します。ただし、今回は、ステップ 1 から `LastEvaluatedKey` 値を取得して、新しい `Query` リクエストの `ExclusiveStartKey` パラメータとして使用します。
3. 新しい `Query` リクエストを実行します。
4. ステップ 1 に進んでください。

言い換えると、`Query` レスポンスからの `LastEvaluatedKey` を次の `Query` リクエストの `ExclusiveStartKey` として使用する必要があります。`Query` レスポンスに `LastEvaluatedKey` の要素がない場合、結果の最後のページを取得します。`LastEvaluatedKey` が空ではない場合でも、必ずしも結果セットにまだ値があることを意味するわけではありません。結果セットの最後まで到達したことを確認できるのは、`LastEvaluatedKey` が空になったときだけです。

AWS CLI を使用してこの動作を表示できます。AWS CLI は、`LastEvaluatedKey` が結果に表示されなくなるまで、低レベルの `Query` リクエストを DynamoDB に繰り返し送信します。特定の年の映画タイトルを取得する次の AWS CLI の例を考えてみます。

```
aws dynamodb query --table-name Movies \
    --projection-expression "title" \
    --key-condition-expression "#y = :yyyy" \
    --expression-attribute-names '{"#y":"year"}' \
    --expression-attribute-values '{":yyyy":{"N":"1993"}}' \
    --page-size 5 \
    --debug
```

通常、AWS CLI はページ分割を自動的に処理します。ただし、この例では、AWS CLI `--page-size` パラメータによりページごとの項目数が制限されています。`--debug` パラメータは、リクエストとレスポンスについての低レベルの情報を表示します。

この例を実行した場合、DynamoDB からの最初のレスポンスは次のようになります。

```
2017-07-07 11:13:15,603 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":5,"Items":[{"title":{"S":"A Bronx Tale"}},
{"title":{"S":"A Perfect World"}},{"title":{"S":"Addams Family Values"}},
{"title":{"S":"Alive"}},{"title":{"S":"Benny & Joon"}}],
"LastEvaluatedKey":{"year":{"N":"1993"},"title":{"S":"Benny & Joon"}},
"ScannedCount":5}'
```

レスポンスの LastEvaluatedKey は、すべての項目が取得されたわけではないことを示します。その後、AWS CLI は DynamoDB に別の Query リクエストを送信します。このリクエストとレスポンスのパターンが、最終レスポンスまで繰り返されます。

```
2017-07-07 11:13:16,291 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"What\'s Eating Gilbert
 Grape"}}],"ScannedCount":1}'
```

LastEvaluatedKey がない場合、これ以上取得する項目がないことを示します。

> Note
>
> AWS SDK は低レベルの DynamoDB レスポンス (LastEvaluatedKey の有無を含む) を処理し、ページ割りした Query 結果のさまざまな抽象化を提供します。たとえば、SDK for Java のドキュメントインターフェイスでは、java.util.Iterator サポートが利用可能なため、結果を一度に 1 つずつ確認できます。
> 各種のプログラミング言語のコード例については、「Amazon DynamoDB 利用開始ガイド」と、該当言語の「AWS SDK ドキュメント」を参照してください。

Query オペレーションの Limit パラメータを使用して結果セット内の項目数を制限することで、ページサイズを小さくすることもできます。

DynamoDB でのクエリ実行の詳細については、「DynamoDB のクエリオペレーション」を参照してください。

クエリオペレーションを使用する際のその他の側面

結果セットの項目数の制限

Query オペレーションを使用すると、読み取られる項目数を制限することができます。これを行うには、Limit パラメータに項目の最大数を設定します。

たとえば、フィルタ式を使用せず、Limit 値を 6 として、テーブルを Query するとします。Query 結果には、リクエストのキー条件式に一致するテーブルからの最初 6 つの項目が含まれます。

ここで、Query にフィルタ式を追加するとします。この場合、DynamoDB は最大 6 つの項目を読み込み、フィルタ式と一致する項目だけを返します。DynamoDB が項目の読み込みを続けた場合、さ

らに多くの項目がフィルタ式にマッチしても、最終的な Query 結果に含まれる項目数は 6 つ以下です。

結果での項目のカウント

Query レスポンスには、条件に一致する項目に加えて次の要素が含まれます。

- ScannedCount — フィルター式 (存在する場合) が適用される前に、キー条件式に一致した項目数。
- Count — フィルター式 (存在する場合) が適用された後に残っている項目数。

> Note
>
> フィルタ式を使用しない場合、ScannedCount と Count は同じ値を持ちます。

Query 結果セットのサイズが 1 MB より大きい場合、ScannedCount および Count では、合計項目数の一部のみが示されます。すべての結果を取得するためには、複数の Query オペレーションを実行する必要があります (テーブルクエリ結果をページ分割する を参照してください)。

各 Query レスポンスには、特定の Query リクエストによって処理された項目の ScannedCount および Count が含まれます。すべての Query リクエストの合計を取得するには、ScannedCount および Count の実行中の集計を維持することができます。

クエリで消費されるキャパシティユニット

パーティションキーの属性名、およびその属性の単一値を指定する限り、いずれのテーブルまたはセカンダリインデックスにも、Query を実行できます。Query により、そのパーティションのキー値を持つすべての項目が返されます。必要に応じて、ソートキーの属性を指定し、比較演算子を使用して、検索結果をさらに絞り込むことができます。QueryAPI オペレーションでは、次のように読み込みキャパシティーユニットを消費します。

...を **Query** する場合	DynamoDB は ... からの読み込み容量ユニットを消費します。
テーブル	テーブルのプロビジョニングされた読み込みキャパシティー。

...を **Query** する場合	DynamoDB は ... からの読み込み容量ユニットを消費します。
グローバルセカンダリインデックス	インデックスのプロビジョニングされた読み込みキャパシティー。
ローカルセカンダリインデックス	ベーステーブルのプロビジョニングされた読み込みキャパシティー。

デフォルトでは、Queryオペレーションはどのくらいの読み込みキャパシティーを消費するかについてのデータを返しません。ただし、この情報を取得するために Query リクエストで ReturnConsumedCapacity パラメータを指定できます。ReturnConsumedCapacity の有効な設定を以下に示します。

- NONE — 消費された容量データは返されません。(これがデフォルトです)
- TOTAL — レスポンスには消費された読み込み容量単位の合計値が含まれます。
- INDEXES — レスポンスは、アクセスする各テーブルとインデックスの消費される容量とともに、消費される読み込み容量単位の合計値を示します。

DynamoDB はアプリケーションに返されるデータ量ではなく、項目の数と項目のサイズに基づいて、消費される読み込みキャパシティユニットの数を計算します。このため、消費される容量ユニットの数は、(デフォルトの動作で) 属性のすべてをリクエストしても、(プロジェクション式を使用して) 一部をリクエストしても、同じになります。この数は、フィルタ式を使用していてもいなくても同じです。Query は 最小読み込みキャパシティユニットを消費して、最大 4 KB の項目について強力な整合性のある読み込みを 1 秒あたり 1 回、または結果整合性のある読み込みを 1 秒あたり 2 回実行します。4 KB より大きい項目を読み込む必要がある場合、DynamoDB には追加の読み込みリクエストユニットが必要です。空のテーブルや、パーティションキーの数が少ない非常に大きなテーブルでは、クエリされたデータの量を超えて追加の RCU が課金される場合があります。これにより、データが存在しない場合でも、Query リクエストを処理するためのコストがカバーされます。

クエリの読み込み整合性

Query オペレーションは、結果的に整合性のある読み込みをデフォルトで行います。つまり、Query 結果が、最近完了した PutItem または UpdateItem オペレーションによる変更を反映しない場合があります。詳細については、「読み込み整合性」を参照してください。

強力な整合性のある読み込みが必要な場合は、ConsistentRead リクエストで true パラメータを Query に設定します。

テーブルおよびインデックスにクエリを実行: Java

Query オペレーションを使用すると、Amazon DynamoDB でテーブルまたはセカンダリインデックスのクエリを実行できます。この関数ではパーティションキー値と等価条件を指定する必要があります。テーブルまたはインデックスにソートキーがある場合は、ソートキー値と条件を指定することで結果を絞り込むことができます。

> **Note**
>
> AWS SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングすることができます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「Java 1.x: DynamoDBMapper」を参照してください。

次に、AWS SDK for Java ドキュメント API を使用して項目を取り出すステップを示します。

1. DynamoDB クラスのインスタンスを作成します。
2. 操作対象のテーブルを表すために、Table クラスのインスタンスを作成します。
3. query インスタンスの Table メソッドを呼び出します。任意のオプションクエリパラメータとともに、取得する項目のパーティションキー値を指定する必要があります。

応答には、クエリによって返されたすべての項目を示す ItemCollection オブジェクトが含まれています。

以下の Java コード例は、前述のタスクの例です。この例では、フォーラムスレッドへの返信を格納する Reply テーブルがあることを前提としています。詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

```
Reply ( Id, ReplyDateTime, ... )
```

各フォーラムスレッドには一意の ID があり、0 またはそれ以上の返信を受け取ることができます。つまり、Id テーブルの Reply 属性は、フォーラム名とフォーラムの件名の両方で構成されています。 Id (パーティションキー) と ReplyDateTime (ソートキー) が、テーブルの複合プライマリキーを構成しています。

次のクエリでは、特定のスレッド件名に対するすべての返信を取り出します。このクエリでは、テーブル名と Subject の両方の値が必要になります。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2).build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("Reply");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Id = :v_id")
    .withValueMap(new ValueMap()
        .withString(":v_id", "Amazon DynamoDB#DynamoDB Thread 1"));

ItemCollection<QueryOutcome> items = table.query(spec);

Iterator<Item> iterator = items.iterator();
Item item = null;
while (iterator.hasNext()) {
    item = iterator.next();
    System.out.println(item.toJSONPretty());
}
```

オプションパラメータの指定

query メソッドでは、複数のオプションパラメータがサポートされています。たとえば、必要に応じて条件を指定して前述のクエリの結果を絞り込み、過去 2 週間の返信が返されるようにできます。この条件はソートキー条件と呼ばれます。指定したクエリ条件が DynamoDB によってプライマリキーのソートキーに対して評価されるためです。その他のオプションパラメータを指定して、クエリ結果の項目から特定の属性のリストだけを取り出すこともできます。

次の Java コード例では、過去 15 日間に投稿されたフォーラムスレッドの返信が取り出されます。この例では、次のものを使用してオプションパラメータを指定しています。

- KeyConditionExpression - 特定のディスカッションフォーラムからの返信を取得し (パーティションキー)、その項目のセット内では、過去 15 日の間に投稿された返信を取得します (ソートキー)。
- FilterExpression - 特定のユーザーからの返信だけを返します。フィルタは、クエリの処理の終了後、ユーザーに結果が返される前に適用されます。

- ValueMap - KeyConditionExpression プレースホルダーの実際の値を定義します。
- ConsistentRead - true に設定すると、強力な整合性のある読み込みをリクエストします。

この例では、すべての低レベル QuerySpec 入力パラメータにアクセスできる Query オブジェクトを使用します。

Example

```
Table table = dynamoDB.getTable("Reply");

long twoWeeksAgoMilli = (new Date()).getTime() - (15L*24L*60L*60L*1000L);
Date twoWeeksAgo = new Date();
twoWeeksAgo.setTime(twoWeeksAgoMilli);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
String twoWeeksAgoStr = df.format(twoWeeksAgo);

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Id = :v_id and ReplyDateTime > :v_reply_dt_tm")
    .withFilterExpression("PostedBy = :v_posted_by")
    .withValueMap(new ValueMap()
        .withString(":v_id", "Amazon DynamoDB#DynamoDB Thread 1")
        .withString(":v_reply_dt_tm", twoWeeksAgoStr)
        .withString(":v_posted_by", "User B"))
    .withConsistentRead(true);

ItemCollection<QueryOutcome> items = table.query(spec);

Iterator<Item> iterator = items.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next().toJSONPretty());
}
```

また、withMaxPageSize メソッドを使用して、ページあたりの項目数を制限することもできます。query メソッドを呼び出すと、結果の項目が含まれている ItemCollection が返されます。その後、結果を 1 ページずつ、最後のページまで処理していくことができます。

次の Java コード例では、前に示したクエリの仕様を変更します。今回、クエリの仕様は withMaxPageSize メソッドを使用します。Page クラスには、コードが各ページの項目を処理できるようにするイテレータがあります。

Example

```
spec.withMaxPageSize(10);

ItemCollection<QueryOutcome> items = table.query(spec);

// Process each page of results
int pageNum = 0;
for (Page<Item, QueryOutcome> page : items.pages()) {

    System.out.println("\nPage: " + ++pageNum);

    // Process each item on the current page
    Iterator<Item> item = page.iterator();
    while (item.hasNext()) {
        System.out.println(item.next().toJSONPretty());
    }
}
```

例 – Java を使用したクエリ

以下のテーブルには、フォーラムのコレクションに関する情報が格納されています。詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

> **Note**
>
> SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングできます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「Java 1.x: DynamoDBMapper」を参照してください。

Example

```
Forum ( Name, ... )
Thread ( ForumName, Subject, Message, LastPostedBy, LastPostDateTime, ...)
Reply ( Id, ReplyDateTime, Message, PostedBy, ...)
```

この Java コードの例では、「DynamoDB」フォーラムの「DynamoDB Thread 1」スレッドに対するさまざまな返信の検索を実行します。

- スレッドに対する返信を検索します。
- 結果のページあたりの項目数に対する制限を指定して、スレッドへの返信を探します。結果セットの項目数がページサイズを超えた場合は、結果の最初のページだけが得られます。このコーディングパターンによって、確実にクエリ結果の全ページがコードで処理されます。
- 過去 15 日間の返信を検索します。
- 特定の日付範囲の返信を検索します。

  前述の 2 つのクエリはどちらも、ソートキー条件を指定してクエリ結果を絞り込む方法、必要に応じてその他のクエリパラメータを使用する方法を示しています。

> Note
>
> このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
> 以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

```
package com.amazonaws.codesamples.document;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.Page;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;

public class DocumentAPIQuery {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";

    public static void main(String[] args) throws Exception {

        String forumName = "Amazon DynamoDB";
        String threadSubject = "DynamoDB Thread 1";

        findRepliesForAThread(forumName, threadSubject);
        findRepliesForAThreadSpecifyOptionalLimit(forumName, threadSubject);
        findRepliesInLast15DaysWithConfig(forumName, threadSubject);
        findRepliesPostedWithinTimePeriod(forumName, threadSubject);
        findRepliesUsingAFilterExpression(forumName, threadSubject);
    }

    private static void findRepliesForAThread(String forumName, String threadSubject) {

        Table table = dynamoDB.getTable(tableName);

        String replyId = forumName + "#" + threadSubject;

        QuerySpec spec = new QuerySpec().withKeyConditionExpression("Id = :v_id")
                .withValueMap(new ValueMap().withString(":v_id", replyId));

        ItemCollection<QueryOutcome> items = table.query(spec);

        System.out.println("\nfindRepliesForAThread results:");

        Iterator<Item> iterator = items.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    private static void findRepliesForAThreadSpecifyOptionalLimit(String forumName,
 String threadSubject) {

        Table table = dynamoDB.getTable(tableName);

        String replyId = forumName + "#" + threadSubject;

```

```
       QuerySpec spec = new QuerySpec().withKeyConditionExpression("Id = :v_id")
               .withValueMap(new ValueMap().withString(":v_id",
replyId)).withMaxPageSize(1);

       ItemCollection<QueryOutcome> items = table.query(spec);

       System.out.println("\nfindRepliesForAThreadSpecifyOptionalLimit results:");

       // Process each page of results
       int pageNum = 0;
       for (Page<Item, QueryOutcome> page : items.pages()) {

           System.out.println("\nPage: " + ++pageNum);

           // Process each item on the current page
           Iterator<Item> item = page.iterator();
           while (item.hasNext()) {
               System.out.println(item.next().toJSONPretty());
           }
       }
   }

   private static void findRepliesInLast15DaysWithConfig(String forumName, String
threadSubject) {

       Table table = dynamoDB.getTable(tableName);

       long twoWeeksAgoMilli = (new Date()).getTime() - (15L * 24L * 60L * 60L *
1000L);
       Date twoWeeksAgo = new Date();
       twoWeeksAgo.setTime(twoWeeksAgoMilli);
       SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
       String twoWeeksAgoStr = df.format(twoWeeksAgo);

       String replyId = forumName + "#" + threadSubject;

       QuerySpec spec = new QuerySpec().withProjectionExpression("Message,
ReplyDateTime, PostedBy")
               .withKeyConditionExpression("Id = :v_id and ReplyDateTime
<= :v_reply_dt_tm")
               .withValueMap(new ValueMap().withString(":v_id",
replyId).withString(":v_reply_dt_tm", twoWeeksAgoStr));

       ItemCollection<QueryOutcome> items = table.query(spec);
```

```
        System.out.println("\nfindRepliesInLast15DaysWithConfig results:");
        Iterator<Item> iterator = items.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    private static void findRepliesPostedWithinTimePeriod(String forumName, String
 threadSubject) {

        Table table = dynamoDB.getTable(tableName);

        long startDateMilli = (new Date()).getTime() - (15L * 24L * 60L * 60L * 1000L);
        long endDateMilli = (new Date()).getTime() - (5L * 24L * 60L * 60L * 1000L);
        java.text.SimpleDateFormat df = new java.text.SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");
        String startDate = df.format(startDateMilli);
        String endDate = df.format(endDateMilli);

        String replyId = forumName + "#" + threadSubject;

        QuerySpec spec = new QuerySpec().withProjectionExpression("Message,
 ReplyDateTime, PostedBy")
                .withKeyConditionExpression("Id = :v_id and ReplyDateTime
 between :v_start_dt and :v_end_dt")
                .withValueMap(new ValueMap().withString(":v_id",
 replyId).withString(":v_start_dt", startDate)
                        .withString(":v_end_dt", endDate));

        ItemCollection<QueryOutcome> items = table.query(spec);

        System.out.println("\nfindRepliesPostedWithinTimePeriod results:");
        Iterator<Item> iterator = items.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }
    }

    private static void findRepliesUsingAFilterExpression(String forumName, String
 threadSubject) {

        Table table = dynamoDB.getTable(tableName);
        String replyId = forumName + "#" + threadSubject;

        QuerySpec spec = new QuerySpec().withProjectionExpression("Message,
 ReplyDateTime, PostedBy")
                .withKeyConditionExpression("Id
 = :v_id").withFilterExpression("PostedBy = :v_postedby")
                .withValueMap(new ValueMap().withString(":v_id",
 replyId).withString(":v_postedby", "User B"));

        ItemCollection<QueryOutcome> items = table.query(spec);

        System.out.println("\nfindRepliesUsingAFilterExpression results:");
        Iterator<Item> iterator = items.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }
    }

}
```

テーブルおよびインデックスにクエリを実行: .NET

`Query` オペレーションを使用すると、Amazon DynamoDB でテーブルまたはセカンダリインデックスのクエリを実行できます。この関数ではパーティションキー値と等価条件を指定する必要があります。テーブルまたはインデックスにソートキーがある場合は、ソートキー値と条件を指定することで結果を絞り込むことができます。

次に、AWS SDK for .NET API を使用してテーブルのクエリを行うステップを示します。

1. `AmazonDynamoDBClient` クラスのインスタンスを作成します。
2. `QueryRequest` クラスのインスタンスを作成して、クエリオペレーションパラメータを指定します。
3. `Query` メソッドを実行し、前述のステップで作成した `QueryRequest` オブジェクトを指定します。

   応答には、クエリによって返されたすべての項目を示す `QueryResult` オブジェクトが含まれています。

以下の C# コード例は、前述のタスクの例です。このコードでは、フォーラムスレッドへの返信を格納する Reply テーブルがあることを前提としています。詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

Example

```
Reply Id, ReplyDateTime, ... )
```

各フォーラムスレッドには一意の ID があり、0 またはそれ以上の返信を受け取ることができます。したがってプライマリキーは、Id (パーティションキー) と ReplyDateTime (ソートキー) の両方で構成されます。

次のクエリでは、特定のスレッド件名に対するすべての返信を取り出します。このクエリでは、テーブル名と Subject の両方の値が必要になります。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

var request = new QueryRequest
{
    TableName = "Reply",
    KeyConditionExpression = "Id = :v_Id",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_Id", new AttributeValue { S =  "Amazon DynamoDB#DynamoDB Thread 1" }}}
};

var response = client.Query(request);

foreach (Dictionary<string, AttributeValue> item in response.Items)
{
    // Process the result.
    PrintItem(item);
}
```

オプションパラメータの指定

Query メソッドでは、複数のオプションパラメータがサポートされています。たとえば、必要に応じて条件を指定して前述のクエリの結果を絞り込み、過去 2 週間の返信が返されるようにできます。この条件はソートキー条件と呼ばれます。指定したクエリ条件が DynamoDB によってプライマリキーのソートキーに対して評価されるためです。その他のオプションパラメータを指定して、クエ

リ結果の項目から特定の属性のリストだけを取り出すこともできます。詳細については、「クエリ」を参照してください。

次の C# コード例では、過去 15 日間に投稿されたフォーラムスレッドの返信が取り出されます。この例では、次のオプションパラメータが指定されています。

- 過去 15 日間の返信だけを取り出す KeyConditionExpression。
- クエリ結果内の項目について取得する属性のリストを指定する ProjectionExpression パラメータ。
- 強力な整合性のある読み込みを実行する ConsistentRead パラメータ。

Example

```
DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
string twoWeeksAgoString = twoWeeksAgoDate.ToString(AWSSDKUtils.ISO8601DateFormat);

var request = new QueryRequest
{
    TableName = "Reply",
    KeyConditionExpression = "Id = :v_Id and ReplyDateTime > :v_twoWeeksAgo",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_Id", new AttributeValue { S =  "Amazon DynamoDB#DynamoDB Thread 2" }},
        {":v_twoWeeksAgo", new AttributeValue { S =  twoWeeksAgoString }}
    },
    ProjectionExpression = "Subject, ReplyDateTime, PostedBy",
    ConsistentRead = true
};

var response = client.Query(request);

foreach (Dictionary<string, AttributeValue> item in response.Items)
{
    // Process the result.
    PrintItem(item);
}
```

また、オプションの Limit パラメータを追加することで、ページサイズ、またはページあたりの項目数を制限することもできます。Query メソッドを実行するたびに、指定された数の項目を含む結果が 1 ページ取得されます。次のページを取得するには、前ページの最後の項目のプライマリキーの値を入力して、Query メソッドを再度実行します。そうすることで、次の項目のセットを返すこ

とができます。この情報は、`ExclusiveStartKey` プロパティを設定することでリクエストに含めます。このプロパティは最初は null である場合があります。以降のページを取り出すには、このプロパティ値を更新して、前ページの最後の項目のプライマリキーにする必要があります。

次の C# 例では、`Reply` テーブルのクエリを実行しています。リクエストでは、オプションの `Limit` および `ExclusiveStartKey` パラメータを指定しています。`do/while` ループは、`LastEvaluatedKey` から null 値が返されるまで、一度に 1 ページのスキャンを継続します。

Example

```
Dictionary<string,AttributeValue> lastKeyEvaluated = null;

do
{
    var request = new QueryRequest
    {
        TableName = "Reply",
        KeyConditionExpression = "Id = :v_Id",
        ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
            {":v_Id", new AttributeValue { S =  "Amazon DynamoDB#DynamoDB Thread 2" }}
        },

        // Optional parameters.
        Limit = 1,
        ExclusiveStartKey = lastKeyEvaluated
    };

    var response = client.Query(request);

    // Process the query result.
    foreach (Dictionary<string, AttributeValue> item in response.Items)
    {
         PrintItem(item);
    }

    lastKeyEvaluated = response.LastEvaluatedKey;

} while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);
```

例 - AWS SDK for .NET を使用したクエリ

以下のテーブルには、フォーラムのコレクションに関する情報が格納されています。詳細については、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

Example

```
Forum ( Name, ... )
Thread ( ForumName, Subject, Message, LastPostedBy, LastPostDateTime, ...)
Reply ( Id, ReplyDateTime, Message, PostedBy, ...)
```

この例では、「DynamoDB」フォーラムの「DynamoDB Thread 1」スレッドに対するさまざまな返信の「検索」を実行します。

- スレッドに対する返信を検索します。
- スレッドに対する返信を検索します。`Limit` クエリパラメータを指定してページサイズを設定します。

  この関数は、複数のページの結果を処理するページ分割の使用を示しています。DynamoDB にはページサイズの制限があり、結果がページサイズを超えた場合には、結果の最初のページだけが得られます。このコーディングパターンによって、確実にクエリ結果の全ページがコードで処理されます。
- 過去 15 日間の返信を検索します。
- 特定の日付範囲の返信を検索します。

  前述の 2 つのクエリはどちらも、ソートキー条件を指定してクエリ結果を絞り込む方法、必要に応じてその他のクエリパラメータを使用する方法を示しています。

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.Util;

namespace com.amazonaws.codesamples
{
    class LowLevelQuery
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                // Query a specific forum and thread.
                string forumName = "Amazon DynamoDB";
                string threadSubject = "DynamoDB Thread 1";

                FindRepliesForAThread(forumName, threadSubject);
                FindRepliesForAThreadSpecifyOptionalLimit(forumName, threadSubject);
                FindRepliesInLast15DaysWithConfig(forumName, threadSubject);
                FindRepliesPostedWithinTimePeriod(forumName, threadSubject);

                Console.WriteLine("Example complete. To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message);
 Console.ReadLine(); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message);
 Console.ReadLine(); }
            catch (Exception e) { Console.WriteLine(e.Message); Console.ReadLine(); }
        }

        private static void FindRepliesPostedWithinTimePeriod(string forumName, string
 threadSubject)
        {
            Console.WriteLine("*** Executing FindRepliesPostedWithinTimePeriod() ***");
            string replyId = forumName + "#" + threadSubject;
            // You must provide date value based on your test data.
            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(21);
            string start = startDate.ToString(AWSSDKUtils.ISO8601DateFormat);

            // You provide date value based on your test data.
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(5);
            string end = endDate.ToString(AWSSDKUtils.ISO8601DateFormat);

            var request = new QueryRequest
            {
                TableName = "Reply",
                ReturnConsumedCapacity = "TOTAL",
                KeyConditionExpression = "Id = :v_replyId and ReplyDateTime
 between :v_start and :v_end",
                ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
                {":v_replyId", new AttributeValue {
                     S = replyId
                 }},
                {":v_start", new AttributeValue {
                     S = start
                 }},
                {":v_end", new AttributeValue {
                     S = end
                 }}
            }
            };

            var response = client.Query(request);

            Console.WriteLine("\nNo. of reads used (by query in
 FindRepliesPostedWithinTimePeriod) {0}",
                      response.ConsumedCapacity.CapacityUnits);
            foreach (Dictionary<string, AttributeValue> item
                 in response.Items)
            {
                PrintItem(item);
            }
            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void FindRepliesInLast15DaysWithConfig(string forumName, string
 threadSubject)
        {
            Console.WriteLine("*** Executing FindRepliesInLast15DaysWithConfig() ***");
            string replyId = forumName + "#" + threadSubject;

            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            string twoWeeksAgoString =
                twoWeeksAgoDate.ToString(AWSSDKUtils.ISO8601DateFormat);

            var request = new QueryRequest
            {
                TableName = "Reply",
                ReturnConsumedCapacity = "TOTAL",
                KeyConditionExpression = "Id = :v_replyId and ReplyDateTime
 > :v_interval",
                ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
                {":v_replyId", new AttributeValue {
                     S = replyId
                 }},
                {":v_interval", new AttributeValue {
                     S = twoWeeksAgoString
                 }}
            },

                // Optional parameter.
                ProjectionExpression = "Id, ReplyDateTime, PostedBy",
                // Optional parameter.
                ConsistentRead = true
            };

            var response = client.Query(request);

            Console.WriteLine("No. of reads used (by query in
 FindRepliesInLast15DaysWithConfig) {0}",
                      response.ConsumedCapacity.CapacityUnits);
            foreach (Dictionary<string, AttributeValue> item
                 in response.Items)
            {
                PrintItem(item);
            }
            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void FindRepliesForAThreadSpecifyOptionalLimit(string forumName,
 string threadSubject)
        {
            Console.WriteLine("*** Executing
 FindRepliesForAThreadSpecifyOptionalLimit() ***");
            string replyId = forumName + "#" + threadSubject;

            Dictionary<string, AttributeValue> lastKeyEvaluated = null;
            do
            {
                var request = new QueryRequest
                {
                    TableName = "Reply",
                    ReturnConsumedCapacity = "TOTAL",
                    KeyConditionExpression = "Id = :v_replyId",
                    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
{
                    {":v_replyId", new AttributeValue {
                         S = replyId
                     }}
                },
                    Limit = 2, // The Reply table has only a few sample items. So the
page size is smaller.
                    ExclusiveStartKey = lastKeyEvaluated
                };

                var response = client.Query(request);

                Console.WriteLine("No. of reads used (by query in
FindRepliesForAThreadSpecifyLimit) {0}\n",
                          response.ConsumedCapacity.CapacityUnits);
                foreach (Dictionary<string, AttributeValue> item
                     in response.Items)
                {
                    PrintItem(item);
                }
                lastKeyEvaluated = response.LastEvaluatedKey;
            } while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);

            Console.WriteLine("To continue, press Enter");


            Console.ReadLine();
        }

        private static void FindRepliesForAThread(string forumName, string
threadSubject)
        {
            Console.WriteLine("*** Executing FindRepliesForAThread() ***");
            string replyId = forumName + "#" + threadSubject;

            var request = new QueryRequest
            {
                TableName = "Reply",
                ReturnConsumedCapacity = "TOTAL",
                KeyConditionExpression = "Id = :v_replyId",
                ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
```

```
                {":v_replyId", new AttributeValue {
                     S = replyId
                 }}
            }
            };

            var response = client.Query(request);
            Console.WriteLine("No. of reads used (by query in FindRepliesForAThread)
 {0}\n",
                      response.ConsumedCapacity.CapacityUnits);
            foreach (Dictionary<string, AttributeValue> item in response.Items)
            {
                PrintItem(item);
            }
            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void PrintItem(
            Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

DynamoDB でのスキャンの使用

Amazon DynamoDB の Scan オペレーションでは、テーブルまたはセカンダリインデックスのすべての項目を読み込みます。デフォルトでは、Scan オペレーションはテーブルまたはインデックスのすべての項目のデータ属性を返します。ProjectionExpression パラメータを使用し、Scan がすべての属性ではなく一部のみを返すようにできます。

Scan は常に結果セットを返します。一致する項目がない場合、結果セットは空になります。

1 回の Scan リクエストで、最大 1 MB のデータを取得できます。DynamoDB では、必要に応じてこのデータにフィルター式を適用して、結果をユーザーに返す前に絞り込むことができます。

トピック

Scan のフィルタ式

Scan 結果の絞り込みが必要な場合は、オプションでフィルタ式を指定できます。フィルタ式によって、Scan 結果の内、どの項目を返すべきか確定します。他のすべての結果は破棄されます。

フィルタ式は、Scan の完了後、結果が返される前に適用されます。そのため、Scan は、フィルタ式があるかどうかにかかわらず、同じ量の読み込みキャパシティーを消費します。

1 回の Scan オペレーションで、最大 1 MB のデータを取得できます。この制限は、フィルタ式を評価する前に適用されます。

Scan では、パーティションキー属性やソートキー属性など、フィルター式で任意の属性を指定できます。

フィルタ式の構文は、条件式の構文と同じです。フィルタ式は、条件式と同じコンパレータ、関数および論理演算子を使用できます。論理演算子の詳細については、「比較演算子および関数リファレンス」を参照してください。

Example

次の AWS Command Line Interface (AWS CLI) の例では Thread テーブルをスキャンして、特定のユーザーによって最後に投稿された項目のみを返します。

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

結果セットの項目数の制限

Scan オペレーションは、結果で返される項目数を制限することができます。これを行うには、フィルタ式を評価する前に、Limit パラメータに、Scan オペレーションが返す項目の最大数を設定します。

たとえば、フィルタ式を使用せず、Scan 値を Limit として、テーブルを 6 するとします。Scan 結果には、テーブルの最初の 6 つの項目が含まれます。

ここで、Scan にフィルタ式を追加するとします。この場合、DynamoDB は返される 6 つの項目にフィルター式を適用し、一致しない項目を廃棄します。最終的な Scan 結果はフィルタリングされる項目の数に応じて、6 つ以下の項目を含みます。

結果のページ分割

DynamoDB では、Scan オペレーションの結果をページ割りします。ページ分割を行うことで Scan 結果が 1 MB サイズ (またはそれ以下) のデータの「ページ」に分割されます。アプリケーションは結果の最初のページ、次に 2 ページと処理できます。

1 つの Scan は、サイズの制限である1 MB 以内の結果セットだけを返します。

さらに結果があるかどうかを確認して、一度に 1 ページずつ結果を取り出すには、アプリケーションで次の操作を行う必要があります。

1. 低レベルの Scan 結果を確認します。

    - 結果に LastEvaluatedKey 要素が含まれる場合、ステップ 2 に進みます。

   - 結果に LastEvaluatedKey がない場合、これ以上取得する項目はありません。

2. 以前のものと同じパラメータを使用して新しい Scan リクエストを作成します。ただし、今回は、ステップ 1 から LastEvaluatedKey 値を取得して、新しい Scan リクエストの ExclusiveStartKey パラメータとして使用します。
3. 新しい Scan リクエストを実行します。
4. ステップ 1 に進んでください。

言い換えると、LastEvaluatedKey レスポンスからの Scan を次の ExclusiveStartKey リクエストの Scan として使用する必要があります。LastEvaluatedKey レスポンスに Scan 要素がない場合、結果の最後のページを取得します。(結果セットの最後まで到達したことを確認できるのは、LastEvaluatedKey がないときだけです)

AWS CLI を使用してこの動作を表示できます。AWS CLI は低レベル Scan リクエストを DynamoDB に送信し、LastEvaluatedKey が結果に表示されなくなるまで送信を繰り返します。次の AWS CLI の例を見てください。この例は、Movies テーブル全体をスキャンしますが、特定のジャンルの映画のみを返します。

```
aws dynamodb scan \
    --table-name Movies \
    --projection-expression "title" \
    --filter-expression 'contains(info.genres,:gen)' \
    --expression-attribute-values '{":gen":{"S":"Sci-Fi"}}' \
    --page-size 100  \
    --debug
```

通常、AWS CLI はページ分割を自動的に処理します。ただし、この例では、AWS CLI --page-size パラメータによりページごとの項目数が制限されています。--debug パラメータは、リクエストとレスポンスについての低レベルの情報を表示します。

Note

ページ分割の結果は、渡した入力パラメータによっても異なります。

- aws dynamodb scan --table-name Prices --max-items 1 を使用すると NextToken が返されます
- aws dynamodb scan --table-name Prices --limit 1 を使用すると LastEvaluatedKey が返されます。

また、特に --starting-token を使用するには、NextToken 値が必要であることに注意してください。

この例を実行した場合、DynamoDB からの最初のレスポンスは次のようになります。

```
2017-07-07 12:19:14,389 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":7,"Items":[{"title":{"S":"Monster on the Campus"}},{"title":{"S":"+1"}},
{"title":{"S":"100 Degrees Below Zero"}},{"title":{"S":"About Time"}},{"title":
{"S":"After Earth"}},
{"title":{"S":"Age of Dinosaurs"}},{"title":{"S":"Cloudy with a Chance of Meatballs
 2"}}],
"LastEvaluatedKey":{"year":{"N":"2013"},"title":{"S":"Curse of
 Chucky"}},"ScannedCount":100}'
```

レスポンスの LastEvaluatedKey は、すべての項目が取得されたわけではないことを示します。その後、AWS CLI は DynamoDB に別の Scan リクエストを送信します。このリクエストとレスポンスのパターンが、最終レスポンスまで繰り返されます。

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

LastEvaluatedKey がない場合、これ以上取得する項目がないことを示します。

Note

AWS SDK は低レベルの DynamoDB レスポンス (LastEvaluatedKey の有無を含む) を処理し、ページ割りした Scan 結果のさまざまな抽象化を提供します。たとえば、SDK for Java のドキュメントインターフェイスでは、java.util.Iterator サポートが利用可能なため、結果を一度に 1 つずつ確認できます。

各種のプログラミング言語のコード例については、Amazon DynamoDB 利用開始ガイドと、該当言語の AWS SDK ドキュメントを参照してください。

結果での項目のカウント

Scan レスポンスには、条件に一致する項目に加えて次の要素が含まれます。

- ScannedCount — ScanFilter が適用される前に評価される項目数。ScannedCount 値が大きく、Count 結果が小さいまたはない場合は、Scan オペレーションが不十分であることを示しています。リクエストでフィルタを使用していない場合、ScannedCount は Count と同じです。
- Count — フィルター式 (存在する場合) が適用された後に残っている項目数。

> Note
>
> フィルタ式を使用しない場合、ScannedCount と Count は同じ値を持ちます。

Scan 結果セットのサイズが 1 MB より大きい場合、ScannedCount および Count では、合計項目数の一部のみが示されます。すべての結果を取得するためには、複数の Scan オペレーションを実行する必要があります (結果のページ分割 を参照してください)。

各 Scan レスポンスには、特定の ScannedCount によって処理された項目の Count および Scan が含まれます。すべての Scan リクエストの合計を取得するには、ScannedCount および Count の実行中の集計を維持することができます。

Scan で消費されるキャパシティユニット

任意のテーブルまたはセカンダリインデックスで Scan できます。Scan オペレーションでは、次のように読み込み容量単位を消費します。

...を **Scan** する場合	DynamoDB は ... からの読み込み容量ユニットを消費します。
テーブル	テーブルのプロビジョニングされた読み込みキャパシティー。
グローバルセカンダリインデックス	インデックスのプロビジョニングされた読み込みキャパシティー。
ローカルセカンダリインデックス	ベーステーブルのプロビジョニングされた読み込みキャパシティー。

デフォルトでは、Scanオペレーションはどのくらいの読み込みキャパシティーを消費するかについてのデータを返しません。ただし、この情報を取得するために Scan リクエストで

ReturnConsumedCapacity パラメータを指定できます。ReturnConsumedCapacity の有効な設定を以下に示します。

- NONE — 消費された容量データは返されません。(これがデフォルトです)
- TOTAL — レスポンスには消費された読み込み容量単位の合計値が含まれます。
- INDEXES — レスポンスは、アクセスする各テーブルとインデックスの消費される容量とともに、消費される読み込み容量単位の合計値を示します。

DynamoDB はアプリケーションに返されるデータ量ではなく、項目の数と項目のサイズに基づいて、消費される読み込みキャパシティユニットの数を計算します。このため、消費される容量ユニットの数は、(デフォルトの動作で) 属性のすべてをリクエストしても、(プロジェクション式を使用して) 一部をリクエストしても、同じになります。数は、フィルタ式を使用していてもいなくても同じです。Scan は、最小読み込み容量ユニットを消費して、1 秒あたり 1 回の強力な整合性のある読み込み、あるいは最大 4 KB の項目について 1 秒あたり 2 回の結果整合性のある読み込みを行います。4 KB より大きい項目を読み込む必要がある場合、DynamoDB には追加の読み込みリクエストユニットが必要です。空のテーブルや、パーティションキーの数が少ない非常に大きなテーブルでは、スキャンされたデータ量を超える追加の RCU が課金される場合があります。これにより、データが存在しない場合でも、Scan リクエストを処理するためのコストがカバーされます。

スキャンの読み込み整合性

Scan オペレーションは、結果的に整合性のある読み込みをデフォルトで行います。つまり、Scan 結果が、最近完了した PutItem または UpdateItem オペレーションによる変更を反映しない場合があります。詳細については、「読み込み整合性」を参照してください。

強力な整合性のある読み込みが必要な場合は、Scan が開始する時に ConsistentRead パラメータを true リクエストで Scan に設定できます。これにより、Scan が開始する前に完了した書き込みオペレーションがすべて Scan 応答に含められます。

ConsistentRead を true に設定し、DynamoDB Streams と同時に使用すると、テーブルのバックアップまたはレプリケーションシナリオで役立ちます。最初に、テーブル内のデータの整合性のあるコピーを取得するため、Scan を true に設定して ConsistentRead を使用します。Scan の実行中、DynamoDB Streams はテーブルで発生した追加の書き込みアクティビティをすべて記録します。Scan が完了したら、ストリームからテーブルへの書き込みアクティビティを適用できます。

> **Note**
>
> Scan を ConsistentRead に設定した true オペレーションでは、ConsistentRead をデフォルト値 (false) のままにした場合と比較して、2 倍の読み込みキャパシティーユニットが使用されます。

並列スキャン

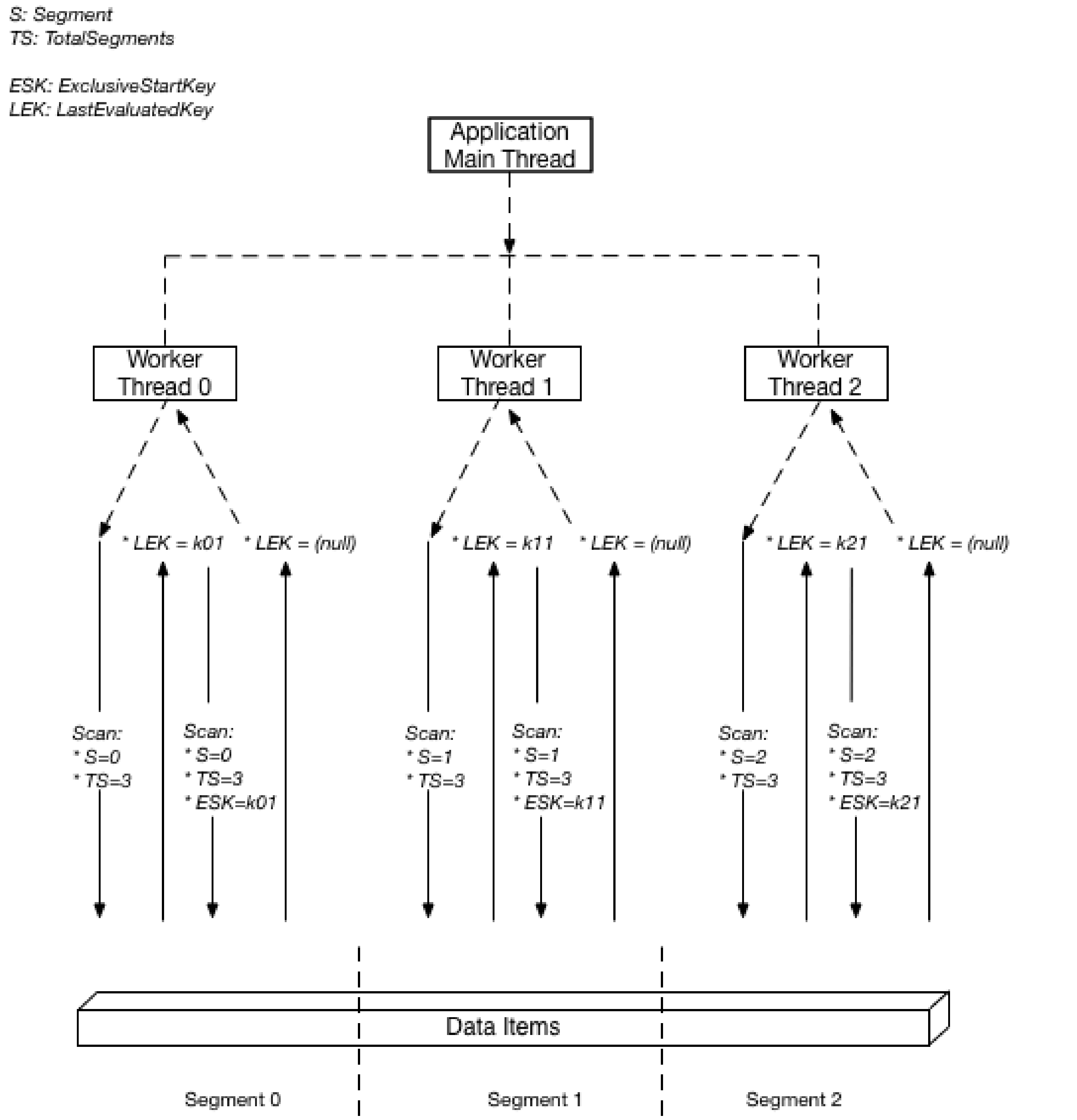
デフォルトでは、Scan オペレーションは、データを順次処理します。Amazon DynamoDB はアプリケーションに 1 MB 単位でデータを返し、アプリケーションは追加の Scan オペレーションを使用して、次の 1 MB のデータを取得できます。

スキャンするテーブルまたはインデックスが大きいほど、Scan を完了するのに時間がかかります。さらに、シーケンシャル Scan は、プロビジョンされた読み込みスループットキャパシティを常に完全に使用できるとは限りません。DynamoDB は大きなテーブルのデータを複数の物理パーティションに分散しますが、Scan オペレーションでは、一度に 1 つのパーティションしか読み込むことができません。このため、Scan のスループットは、単一のパーティションの最大スループットによって制約されます。

これらの問題に対処するために、Scan オペレーションでは、テーブルまたはセカンダリインデックスを論理的に複数のセグメントに分割し、複数のアプリケーションワーカーがセグメントを並行してスキャンします。各ワーカーは、スレッド (マルチスレッドをサポートするプログラミング言語) またはオペレーティングシステムプロセスにすることができます。並列スキャンを実行するには、各ワーカーが独自の Scan リクエストを以下のパラメータで送信します。

- Segment — 特定のワーカーによってスキャンされるセグメント。各ワーカーは、Segment に異なる値を使用する必要があります。
- TotalSegments — 並列スキャンのセグメントの合計数。この値は、アプリケーションが使用するワーカーの数と同じでなければなりません。

次の図表は、マルチスレッドアプリケーションが 3 度の並列処理で並列 Scan を実行する方法を示しています。



この図では、アプリケーションは 3 つのスレッドを生成し、各スレッドに番号を割り当てます。(セグメントはゼロベースであるため、最初の数値は常に 0 です。) 各スレッドは Scan リクエストを公開し、設定 Segment をその指定された番号に設定し、TotalSegments を 3 に設定します。各ス

レッドは、指定されたセグメントをスキャンし、一度に 1 MB のデータを取得し、アプリケーションのメインスレッドにデータを返します。

`Segment` および `TotalSegments` の値は、個々の `Scan` リクエストに適用されるため、いつでも異なる値を使用できます。アプリケーションが最高のパフォーマンスを達成するまで、これらの値および使用するワーカーの数を試さなければならない場合があります。

> **Note**
>
> 多数のワーカーを使用した並列スキャンでは、スキャン対象のテーブルまたはインデックスに対してプロビジョンされたスループットをすべて簡単に使用できます。テーブルまたはインデックスが他のアプリケーションから大量の読み込みまたは書き込みアクティビティが発生している場合は、このようなスキャンを避けることをお勧めします。
> リクエストごとに返されるデータの量を制御するには、`Limit` パラメータを使用します。これにより、1 人のワーカーがプロビジョンされたスループットをすべて消費し、他のすべてのワーカーが犠牲になる状況を防ぐことができます。

テーブルおよびインデックスのスキャン: Java

`Scan` オペレーションは、Amazon DynamoDB のテーブルまたはインデックス内のすべての項目を読み込みます。

次に、AWS SDK for Java ドキュメント API を使用してテーブルをスキャンするステップを示します。

1. `AmazonDynamoDB` クラスのインスタンスを作成します。
2. `ScanRequest` クラスのインスタンスを作成して、スキャンパラメータを指定します。

   必須パラメータは、テーブル名のみです。
3. `scan` メソッドを実行し、前述のステップで作成した `ScanRequest` オブジェクトを指定します。

以下の `Reply` テーブルは、フォーラムスレッドの返信を保存します。

Example

```
Reply ( Id, ReplyDateTime, Message, PostedBy )
```

テーブルには、さまざまなフォーラムスレッドに対するすべての返信が保持されます。したがってプライマリキーは、Id (パーティションキー) と ReplyDateTime (ソートキー) の両方で構成されます。以下の Java コード例は、テーブル全体をスキャンします。ScanRequest インスタンスは、スキャンするテーブルの名前を指定します。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

ScanRequest scanRequest = new ScanRequest()
    .withTableName("Reply");

ScanResponse result = client.scan(scanRequest);
for (Map<String, AttributeValue> item : result.getItems()){
    printItem(item);
}
```

オプションパラメータの指定

scan メソッドでは、複数のオプションパラメータがサポートされています。たとえば、オプションでフィルター式を使用して、スキャン結果をフィルタリングできます。フィルター式では、条件を評価する条件および属性の名前と値を指定できます。詳細については、「スキャン」を参照してください。

以下の Java サンプルは、ProductCatalog テーブルをスキャンして、料金が 0 未満の項目を検索します。この例では、次のオプションパラメータが指定されています。

- 料金が 0 (エラー状態) 未満の項目のみを取得するフィルター式。
- クエリ結果内の項目について取得する属性のリスト。

Example

```
Map<String, AttributeValue> expressionAttributeValues =
    new HashMap<String, AttributeValue>();
expressionAttributeValues.put(":val", new AttributeValue().withN("0"));

ScanRequest scanRequest = new ScanRequest()
    .withTableName("ProductCatalog")
    .withFilterExpression("Price < :val")
    .withProjectionExpression("Id")
    .withExpressionAttributeValues(expressionAttributeValues);
```

```
ScanResponse result = client.scan(scanRequest);
for (Map<String, AttributeValue> item : result.getItems()) {
    printItem(item);
}
```

また、スキャンリクエストの withLimit メソッドを使用して、ページのサイズ、または 1 ページあたりの項目数を制限することもできます。scan メソッドを実行するたびに、指定された数の項目を含む結果が 1 ページ取得されます。次のページを取得するには、前ページの最後の項目のプライマリキーの値を入力して、scan メソッドを再度実行します。そうすることで、scan メソッドが、次の項目のセットを返すことができます。この情報は、withExclusiveStartKey メソッドを使用することでリクエストに含めます。最初は、このメソッドのパラメータを null にすることができます。以降のページを取り出すには、このプロパティ値を更新して、前ページの最後の項目のプライマリキーにする必要があります。

次の Java コード例では、ProductCatalog テーブルをスキャンします。リクエストでは、withLimit および withExclusiveStartKey メソッドが使用されます。do/while ループは、結果の getLastEvaluatedKey メソッドから null 値が返されるまで、一度に 1 ページのスキャンを継続します。

Example

```
Map<String, AttributeValue> lastKeyEvaluated = null;
do {
    ScanRequest scanRequest = new ScanRequest()
        .withTableName("ProductCatalog")
        .withLimit(10)
        .withExclusiveStartKey(lastKeyEvaluated);

    ScanResponse result = client.scan(scanRequest);
    for (Map<String, AttributeValue> item : result.getItems()){
        printItem(item);
    }
    lastKeyEvaluated = result.getLastEvaluatedKey();
} while (lastKeyEvaluated != null);
```

例 - Java を使用したスキャン

次の Java コード例では、動作サンプルを提供し、ProductCatalog テーブルをスキャンして、料金が 100 未満の項目を検索します。

Note

SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングできます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「Java 1.x: DynamoDBMapper」を参照してください。

Note

このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

```
package com.amazonaws.codesamples.document;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class DocumentAPIScan {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);
    static String tableName = "ProductCatalog";

    public static void main(String[] args) throws Exception {
        findProductsForPriceLessThanOneHundred();
    }

    private static void findProductsForPriceLessThanOneHundred() {

        Table table = dynamoDB.getTable(tableName);

        Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
        expressionAttributeValues.put(":pr", 100);

        ItemCollection<ScanOutcome> items = table.scan("Price < :pr", //
 FilterExpression
                "Id, Title, ProductCategory, Price", // ProjectionExpression
                null, // ExpressionAttributeNames - not used in this example
                expressionAttributeValues);

        System.out.println("Scan of " + tableName + " for items with a price less than
 100.");
        Iterator<Item> iterator = items.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }
    }

}
```

例 - Java を使用した並列スキャン

次の Java コードの例は並列スキャン方法を示します。プログラムは、`ParallelScanTest` という名前のテーブルを削除してから再作成します。その後、テーブルにデータをロードします。データのロードが完了すると、プログラムは複数のスレッドを生成し、`Scan` リクエストを並列に発行します。プログラムは、各並列リクエストのランタイム統計を出力します。

> **Note**
>
> SDK for Java には、オブジェクト永続性モデルも用意されています。このモデルにより、クライアント側のクラスを DynamoDB テーブルにマッピングできます。この方法により、記述する必要のあるコードの量を減らすことができます。詳細については、「Java 1.x: DynamoDBMapper」を参照してください。

Note

このコード例では、アカウントの DynamoDB に対し、DynamoDB でのコード例用のテーブルの作成とデータのロード セクションの手順に従ってデータが既にロードされていることを前提としています。
以下の例を実行するための詳しい手順については、「Java コードの例」を参照してください。

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.ScanSpec;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class DocumentAPIParallelScan {

    // total number of sample items
    static int scanItemCount = 300;

    // number of items each scan request should return
    static int scanItemLimit = 10;
```

```
   // number of logical segments for parallel scan
   static int parallelScanThreads = 16;

   // table that will be used for scanning
   static String parallelScanTestTableName = "ParallelScanTest";

   static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
   static DynamoDB dynamoDB = new DynamoDB(client);

   public static void main(String[] args) throws Exception {
       try {

           // Clean up the table
           deleteTable(parallelScanTestTableName);
           createTable(parallelScanTestTableName, 10L, 5L, "Id", "N");

           // Upload sample data for scan
           uploadSampleProducts(parallelScanTestTableName, scanItemCount);

           // Scan the table using multiple threads
           parallelScan(parallelScanTestTableName, scanItemLimit,
parallelScanThreads);
       } catch (AmazonServiceException ase) {
           System.err.println(ase.getMessage());
       }
   }

   private static void parallelScan(String tableName, int itemLimit, int
numberOfThreads) {
       System.out.println(
               "Scanning " + tableName + " using " + numberOfThreads + " threads " +
itemLimit + " items at a time");
       ExecutorService executor = Executors.newFixedThreadPool(numberOfThreads);

       // Divide DynamoDB table into logical segments
       // Create one task for scanning each segment
       // Each thread will be scanning one segment
       int totalSegments = numberOfThreads;
       for (int segment = 0; segment < totalSegments; segment++) {
           // Runnable task that will only scan one segment
           ScanSegmentTask task = new ScanSegmentTask(tableName, itemLimit,
totalSegments, segment);
```

```
            // Execute the task
            executor.execute(task);
        }

        shutDownExecutorService(executor);
    }

    // Runnable task for scanning a single segment of a DynamoDB table
    private static class ScanSegmentTask implements Runnable {

        // DynamoDB table to scan
        private String tableName;

        // number of items each scan request should return
        private int itemLimit;

        // Total number of segments
        // Equals to total number of threads scanning the table in parallel
        private int totalSegments;

        // Segment that will be scanned with by this task
        private int segment;

        public ScanSegmentTask(String tableName, int itemLimit, int totalSegments, int
 segment) {
            this.tableName = tableName;
            this.itemLimit = itemLimit;
            this.totalSegments = totalSegments;
            this.segment = segment;
        }

        @Override
        public void run() {
            System.out.println("Scanning " + tableName + " segment " + segment + " out
 of " + totalSegments
                    + " segments " + itemLimit + " items at a time...");
            int totalScannedItemCount = 0;

            Table table = dynamoDB.getTable(tableName);

            try {
                ScanSpec spec = new
 ScanSpec().withMaxResultSize(itemLimit).withTotalSegments(totalSegments)
                        .withSegment(segment);
                ItemCollection<ScanOutcome> items = table.scan(spec);
                Iterator<Item> iterator = items.iterator();

                Item currentItem = null;
                while (iterator.hasNext()) {
                    totalScannedItemCount++;
                    currentItem = iterator.next();
                    System.out.println(currentItem.toString());
                }

            } catch (Exception e) {
                System.err.println(e.getMessage());
            } finally {
                System.out.println("Scanned " + totalScannedItemCount + " items from 
segment " + segment + " out of "
                        + totalSegments + " of " + tableName);
            }
        }
    }

    private static void uploadSampleProducts(String tableName, int itemCount) {
        System.out.println("Adding " + itemCount + " sample items to " + tableName);
        for (int productIndex = 0; productIndex < itemCount; productIndex++) {
            uploadProduct(tableName, productIndex);
        }
    }

    private static void uploadProduct(String tableName, int productIndex) {

        Table table = dynamoDB.getTable(tableName);

        try {
            System.out.println("Processing record #" + productIndex);

            Item item = new Item().withPrimaryKey("Id", productIndex)
                    .withString("Title", "Book " + productIndex + " 
Title").withString("ISBN", "111-1111111111")
                    .withStringSet("Authors", new 
HashSet<String>(Arrays.asList("Author1"))).withNumber("Price", 2)
                    .withString("Dimensions", "8.5 x 11.0 x 
0.5").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory", 
"Book");
```

```
           table.putItem(item);

       } catch (Exception e) {
           System.err.println("Failed to create item " + productIndex + " in " +
tableName);
           System.err.println(e.getMessage());
       }
   }

   private static void deleteTable(String tableName) {
       try {

           Table table = dynamoDB.getTable(tableName);
           table.delete();
           System.out.println("Waiting for " + tableName + " to be deleted...this may
take a while...");
           table.waitForDelete();

       } catch (Exception e) {
           System.err.println("Failed to delete table " + tableName);
           e.printStackTrace(System.err);
       }
   }

   private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
           String partitionKeyName, String partitionKeyType) {

       createTable(tableName, readCapacityUnits, writeCapacityUnits, partitionKeyName,
partitionKeyType, null, null);
   }

   private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
           String partitionKeyName, String partitionKeyType, String sortKeyName,
String sortKeyType) {

       try {
           System.out.println("Creating table " + tableName);

           List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
           keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition
```

```
                         // key

           List<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
           attributeDefinitions
                   .add(new AttributeDefinition().withAttributeName(partitionKeyName)
                           .withAttributeType(partitionKeyType));

           if (sortKeyName != null) {
               keySchema.add(new
KeySchemaElement().withAttributeName(sortKeyName).withKeyType(KeyType.RANGE)); // Sort

                         // key
               attributeDefinitions
                       .add(new
AttributeDefinition().withAttributeName(sortKeyName).withAttributeType(sortKeyType));
           }

           Table table = dynamoDB.createTable(tableName, keySchema,
attributeDefinitions, new ProvisionedThroughput()

.withReadCapacityUnits(readCapacityUnits).withWriteCapacityUnits(writeCapacityUnits));
           System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
           table.waitForActive();

       } catch (Exception e) {
           System.err.println("Failed to create table " + tableName);
           e.printStackTrace(System.err);
       }
   }

   private static void shutDownExecutorService(ExecutorService executor) {
       executor.shutdown();
       try {
           if (!executor.awaitTermination(10, TimeUnit.SECONDS)) {
               executor.shutdownNow();
           }
       } catch (InterruptedException e) {
           executor.shutdownNow();

           // Preserve interrupt status
           Thread.currentThread().interrupt();
```

```
        }
    }
}
```

テーブルおよびインデックスのスキャン: .NET

Scan オペレーションは、Amazon DynamoDB のテーブルまたはインデックス内のすべての項目を読み込みます。

次に、低レベル AWS SDK for .NET API を使用してテーブルのスキャンを行うステップを示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. ScanRequest クラスのインスタンスを作成して、スキャンオペレーションパラメータを指定します。

   必須パラメータは、テーブル名のみです。
3. Scan メソッドを実行し、前述のステップで作成した ScanRequest オブジェクトを指定します。

以下の Reply テーブルは、フォーラムスレッドの返信を保存します。

Example

```
>Reply ( <emphasis role="underline">Id</emphasis>, <emphasis
 role="underline">ReplyDateTime</emphasis>, Message, PostedBy )
```

テーブルには、さまざまなフォーラムスレッドに対するすべての返信が保持されます。したがってプライマリキーは、Id (パーティションキー) と ReplyDateTime (ソートキー) の両方で構成されます。次の C# のサンプルコードでは、テーブル全体をスキャンします。ScanRequest インスタンスは、スキャンするテーブルの名前を指定します。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

var request = new ScanRequest
{
    TableName = "Reply",
};

var response = client.Scan(request);
var result = response.ScanResult;

foreach (Dictionary<string, AttributeValue> item in response.ScanResult.Items)
{
  // Process the result.
  PrintItem(item);
}
```

オプションパラメータの指定

Scan メソッドでは、複数のオプションパラメータがサポートされています。たとえば、オプションでスキャンフィルターを使用して、スキャン結果をフィルタリングできます。スキャンフィルターでは、条件を評価する条件および属性の名前を指定できます。詳細については、「スキャン」を参照してください。

以下の C# コードは、ProductCatalog テーブルをスキャンして、料金が 0 未満の項目を検索します。このサンプルでは、次のオプションパラメータが指定されています。

- 料金が 0 (エラー状態) 未満の項目のみを取得する FilterExpression パラメータ。
- クエリ結果内の項目について取得する属性を指定する ProjectionExpression パラメータ。

以下の C# サンプルは、ProductCatalog テーブルをスキャンして、料金が 0 未満のすべての項目を検索できます。

Example

```
var forumScanRequest = new ScanRequest
 {
   TableName = "ProductCatalog",
   // Optional parameters.
   ExpressionAttributeValues = new Dictionary<string,AttributeValue> {
        {":val", new AttributeValue { N = "0" }}
   },
   FilterExpression = "Price < :val",
   ProjectionExpression = "Id"
 };
```

また、オプションの Limit パラメータを追加することで、ページサイズ、または 1 ページあたりの項目数を制限することもできます。Scan メソッドを実行するたびに、指定された数の項目を含む結果が 1 ページ取得されます。次のページを取得するには、前ページの最後の項目のプライマリキーの値を入力して、Scan メソッドを再度実行します。そうすることで、Scan メソッドが、次の項目のセットを返すことができます。この情報は、ExclusiveStartKey プロパティを設定することでリクエストに含めます。このプロパティは最初は null である場合があります。以降のページを取り出すには、このプロパティ値を更新して、前ページの最後の項目のプライマリキーにする必要があります。

次の C# コード例では、ProductCatalog テーブルをスキャンします。リクエストでは、オプションの Limit および ExclusiveStartKey パラメータを指定しています。do/while ループは、LastEvaluatedKey から null 値が返されるまで、一度に 1 ページのスキャンを継続します。

Example

```
Dictionary<string, AttributeValue> lastKeyEvaluated = null;
do
{
    var request = new ScanRequest
    {
        TableName = "ProductCatalog",
        Limit = 10,
        ExclusiveStartKey = lastKeyEvaluated
    };

    var response = client.Scan(request);

    foreach (Dictionary<string, AttributeValue> item
      in response.Items)
    {
        PrintItem(item);
    }
    lastKeyEvaluated = response.LastEvaluatedKey;

} while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);
```

例 - .NET を使用したスキャン

次の C# コードは、実行例を提供し、ProductCatalog テーブルをスキャンして、料金が 0 未満の項目を検索します。

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelScan
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                FindProductsForPriceLessThanZero();

                Console.WriteLine("Example complete. To continue, press Enter");
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void FindProductsForPriceLessThanZero()
        {
            Dictionary<string, AttributeValue> lastKeyEvaluated = null;
            do
            {
                var request = new ScanRequest
                {
                    TableName = "ProductCatalog",
                    Limit = 2,
                    ExclusiveStartKey = lastKeyEvaluated,
                    ExpressionAttributeValues = new Dictionary<string, AttributeValue>
 {
                    {":val", new AttributeValue {
                         N = "0"
```

```
                     }}
                },
                    FilterExpression = "Price < :val",

                    ProjectionExpression = "Id, Title, Price"
                };

                var response = client.Scan(request);

                foreach (Dictionary<string, AttributeValue> item
                     in response.Items)
                {
                    Console.WriteLine("\nScanThreadTableUsePaging - printing.....");
                    PrintItem(item);
                }
                lastKeyEvaluated = response.LastEvaluatedKey;
            } while (lastKeyEvaluated != null && lastKeyEvaluated.Count != 0);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void PrintItem(
            Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
```

```
}
```

例 - .NET を使用した並列スキャン

次の C# コードの例は並列スキャン方法を示します。プログラムは、ProductCatalog テーブルを削除してから再作成します。その後、テーブルにデータをロードします。データのロードが完了すると、プログラムは複数のスレッドを生成し、Scan リクエストを並列に発行します。最後に、プログラムはランタイム統計の要約を出力します。

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelParallelScan
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "ProductCatalog";
        private static int exampleItemCount = 100;
        private static int scanItemLimit = 10;
        private static int totalSegments = 5;


        static void Main(string[] args)
        {
            try
            {
                DeleteExampleTable();
                CreateExampleTable();
                UploadExampleData();
                ParallelScanExampleTable();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void ParallelScanExampleTable()
        {
            Console.WriteLine("\n*** Creating {0} Parallel Scan Tasks to scan {1}",
 totalSegments, tableName);
            Task[] tasks = new Task[totalSegments];
            for (int segment = 0; segment < totalSegments; segment++)
            {
                int tmpSegment = segment;
                Task task = Task.Factory.StartNew(() =>
                                  {
                                      ScanSegment(totalSegments, tmpSegment);
                                  });

                tasks[segment] = task;
            }

            Console.WriteLine("All scan tasks are created, waiting for them to
 complete.");
            Task.WaitAll(tasks);

            Console.WriteLine("All scan tasks are completed.");
        }

        private static void ScanSegment(int totalSegments, int segment)
        {
            Console.WriteLine("*** Starting to Scan Segment {0} of {1} out of {2} total
 segments ***", segment, tableName, totalSegments);
            Dictionary<string, AttributeValue> lastEvaluatedKey = null;
            int totalScannedItemCount = 0;
            int totalScanRequestCount = 0;
            do
            {
                var request = new ScanRequest
                {
                    TableName = tableName,
                    Limit = scanItemLimit,
                    ExclusiveStartKey = lastEvaluatedKey,
                    Segment = segment,
                    TotalSegments = totalSegments
                };
```

```
               var response = client.Scan(request);
               lastEvaluatedKey = response.LastEvaluatedKey;
               totalScanRequestCount++;
               totalScannedItemCount += response.ScannedCount;
               foreach (var item in response.Items)
               {
                   Console.WriteLine("Segment: {0}, Scanned Item with Title: {1}",
segment, item["Title"].S);
               }
           } while (lastEvaluatedKey.Count != 0);

           Console.WriteLine("*** Completed Scan Segment {0} of {1}.
TotalScanRequestCount: {2}, TotalScannedItemCount: {3} ***", segment, tableName,
totalScanRequestCount, totalScannedItemCount);
       }

       private static void UploadExampleData()
       {
           Console.WriteLine("\n*** Uploading {0} Example Items to {1} Table***",
exampleItemCount, tableName);
           Console.Write("Uploading Items: ");
           for (int itemIndex = 0; itemIndex < exampleItemCount; itemIndex++)
           {
               Console.Write("{0}, ", itemIndex);
               CreateItem(itemIndex.ToString());
           }
           Console.WriteLine();
       }

       private static void CreateItem(string itemIndex)
       {
           var request = new PutItemRequest
           {
               TableName = tableName,
               Item = new Dictionary<string, AttributeValue>()
           {
               { "Id", new AttributeValue {
                     N = itemIndex
                 }},
               { "Title", new AttributeValue {
                     S = "Book " + itemIndex + " Title"
                 }},
               { "ISBN", new AttributeValue {
                     S = "11-11-11-11"
                 }},
               { "Authors", new AttributeValue {
                     SS = new List<string>{"Author1", "Author2" }
                 }},
               { "Price", new AttributeValue {
                     N = "20.00"
                 }},
               { "Dimensions", new AttributeValue {
                     S = "8.5x11.0x.75"
                 }},
               { "InPublication", new AttributeValue {
                     BOOL = false
                 } }
           }
           };
           client.PutItem(request);
       }

       private static void CreateExampleTable()
       {
           Console.WriteLine("\n*** Creating {0} Table ***", tableName);
           var request = new CreateTableRequest
           {
               AttributeDefinitions = new List<AttributeDefinition>()
           {
               new AttributeDefinition
               {
                   AttributeName = "Id",
                   AttributeType = "N"
               }
           },
               KeySchema = new List<KeySchemaElement>
           {
               new KeySchemaElement
               {
                   AttributeName = "Id",
                   KeyType = "HASH" //Partition key
               }
           },
               ProvisionedThroughput = new ProvisionedThroughput
               {
                   ReadCapacityUnits = 5,
                   WriteCapacityUnits = 6
               },
               TableName = tableName
           };

           var response = client.CreateTable(request);

           var result = response;
           var tableDescription = result.TableDescription;
           Console.WriteLine("{1}: {0} \t ReadsPerSec: {2} \t WritesPerSec: {3}",
                     tableDescription.TableStatus,
                     tableDescription.TableName,
                     tableDescription.ProvisionedThroughput.ReadCapacityUnits,
                     tableDescription.ProvisionedThroughput.WriteCapacityUnits);

           string status = tableDescription.TableStatus;
           Console.WriteLine(tableName + " - " + status);

           WaitUntilTableReady(tableName);
       }

       private static void DeleteExampleTable()
       {
           try
           {
               Console.WriteLine("\n*** Deleting {0} Table ***", tableName);
               var request = new DeleteTableRequest
               {
                   TableName = tableName
               };

               var response = client.DeleteTable(request);
               var result = response;
               Console.WriteLine("{0} is being deleted...", tableName);
               WaitUntilTableDeleted(tableName);
           }
           catch (ResourceNotFoundException)
           {
               Console.WriteLine("{0} Table delete failed: Table does not exist",
tableName);
           }
       }

       private static void WaitUntilTableReady(string tableName)
       {
           string status = null;
           // Let us wait until table is created. Call DescribeTable.
           do
           {
               System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
               try
               {
                   var res = client.DescribeTable(new DescribeTableRequest
                   {
                       TableName = tableName
                   });

                   Console.WriteLine("Table name: {0}, status: {1}",
                             res.Table.TableName,
                             res.Table.TableStatus);
                   status = res.Table.TableStatus;
               }
               catch (ResourceNotFoundException)
               {
                   // DescribeTable is eventually consistent. So you might
                   // get resource not found. So we handle the potential exception.
               }
           } while (status != "ACTIVE");
       }

       private static void WaitUntilTableDeleted(string tableName)
       {
           string status = null;
           // Let us wait until table is deleted. Call DescribeTable.
           do
           {
               System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
               try
               {
                   var res = client.DescribeTable(new DescribeTableRequest
                   {
                       TableName = tableName
                   });

                   Console.WriteLine("Table name: {0}, status: {1}",
                             res.Table.TableName,
                             res.Table.TableStatus);
                   status = res.Table.TableStatus;
               }
               catch (ResourceNotFoundException)
               {
                   Console.WriteLine("Table name: {0} is not found. It is deleted",
tableName);
                   return;
               }
           } while (status == "DELETING");
       }
   }
}
```

PartiQL: Amazon DynamoDB 用の SQL 互換クエリ言語

Amazon DynamoDB は SQL 互換のクエリ言語である PartiQL をサポートしており、Amazon DynamoDB でデータの選択、挿入、更新、および削除を行うことができます。PartiQL を使用すると、AWS Management Console、NoSQL Workbench、AWS Command Line Interface、および PartiQL 用の DynamoDB API を使用して、DynamoDB テーブルと簡単にやり取りを行い、アドホッククエリを実行できます。

PartiQL オペレーションは、他の DynamoDB データプレーンオペレーションと同様の可用性、レイテンシー、パフォーマンスを提供します。

次のセクションでは、PartiQL の DynamoDB 実装について説明します。

トピック

PartiQL とは何ですか?

PartiQL は、構造化データ、半構造化データ、ネストされたデータを含む複数のデータストア間で、SQL 互換のクエリアクセスを提供します。PartiQL は、Amazon 内で広く使用されており、現在、DynamoDB を含む多くの AWS のサービスの一部として利用できます。

PartiQL の仕様とコアクエリ言語のチュートリアルについては、PartiQL ドキュメントを参照してください。

> **Note**
>
> - Amazon DynamoDB は、PartiQL クエリ言語のサブセットをサポートしています。
> - Amazon DynamoDB では、Amazon ion データ形式または Amazon Ion リテラルはサポートしていません。

Amazon DynamoDB での PartiQL

DynamoDB で PartiQL クエリを実行するには、次を使用します。

- DynamoDB コンソール
- NoSQL Workbench
- AWS Command Line Interface (AWS CLI)
- DynamoDB API

これらの方法を使用して DynamoDB にアクセスする方法については、「DynamoDB へのアクセス」を参照してください。

DynamoDB 用の PartiQL の開始方法

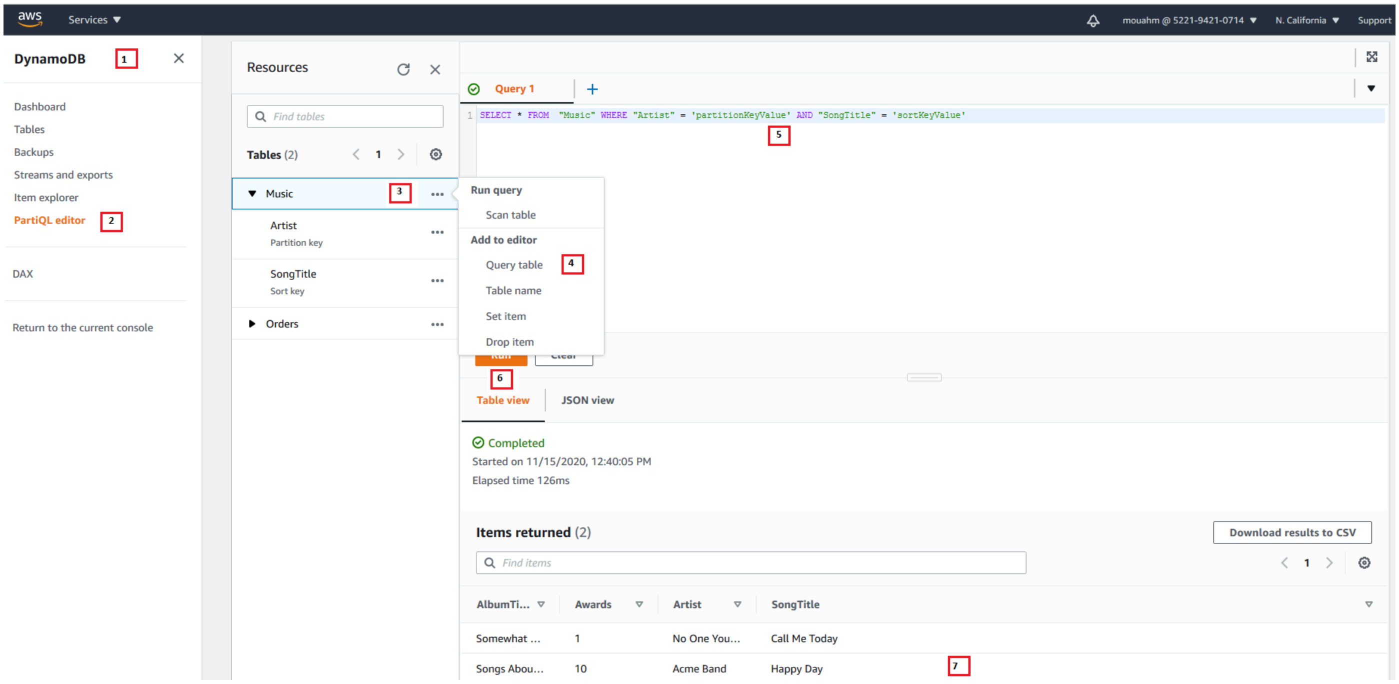
このセクションでは、Amazon DynamoDB コンソール、AWS Command Line Interface (AWS CLI)、および DynamoDB API から DynamoDB 用の PartiQL を使用する方法について説明します。

次の例では、「DynamoDB の開始方法」チュートリアルで、DynamoDB テーブルを定義していることが前提です。

DynamoDB コンソール、AWS Command Line Interface、または DynamoDB API を使用して DynamoDB にアクセスする方法については、「DynamoDB へのアクセス」を参照してください。

NoSQL Workbench をダウンロードして、DynamoDB 用の PartiQL ステートメントを構築するには、DynamoDB Operation Builder 用の NoSQL Workbench の右上にある [PartiQL operations] (PartiQL オペレーション) を選択します。

Console



```
SELECT * FROM  "Music" WHERE "Artist" = 'partitionKeyValue' AND "SongTitle" = 'sortKeyValue'
```

Completed
Started on 11/15/2020, 12:40:05 PM
Elapsed time 126ms

Items returned (2)

AlbumTi...	Awards	Artist	SongTitle
Somewhat ...	1	No One You...	Call Me Today
Songs Abou...	10	Acme Band	Happy Day

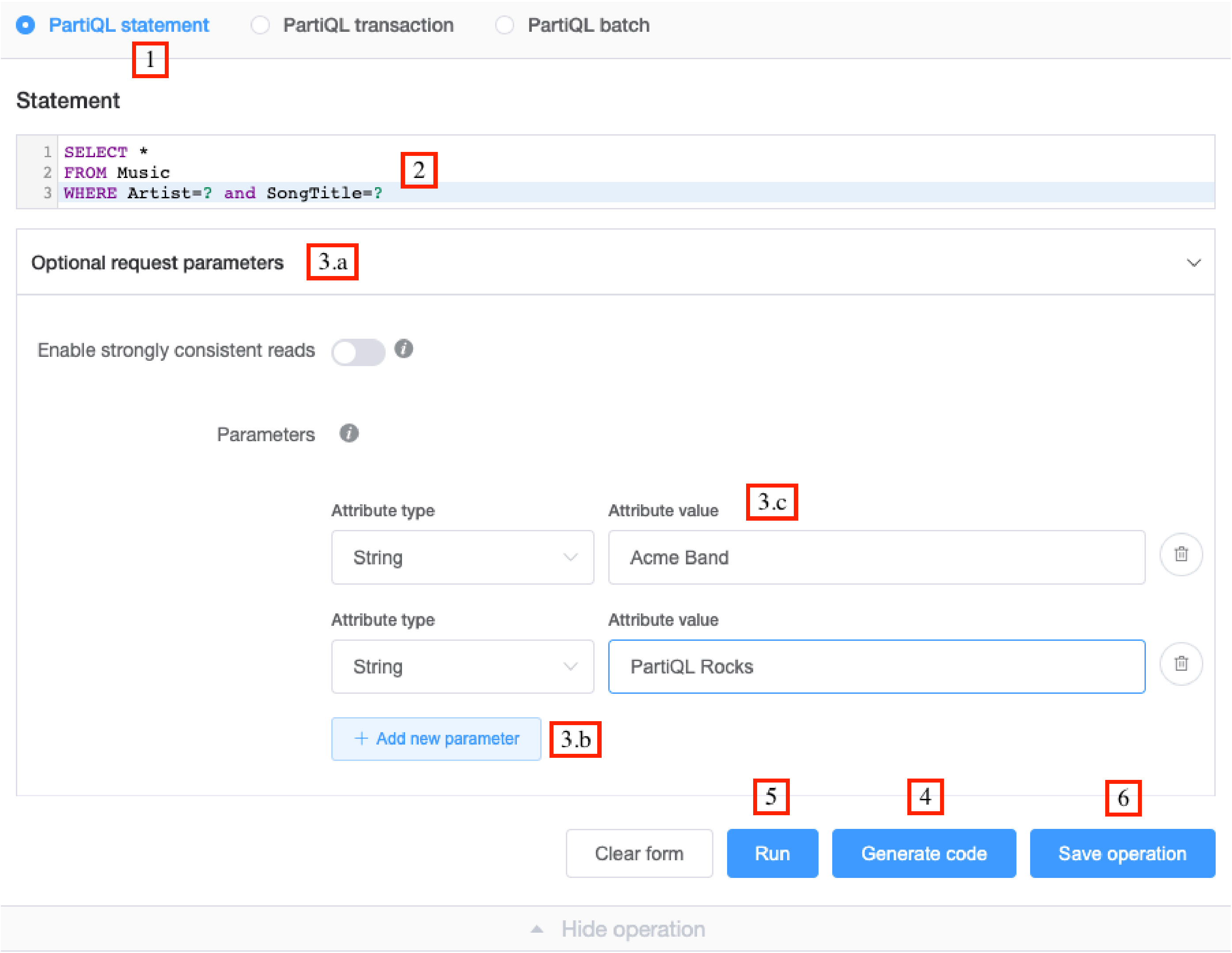
Note

DynamoDB 用 PartiQL は、新しい DynamoDB コンソールでのみ使用できます。新しい DynamoDB コンソールを使用するには、コンソール左側のナビゲーションペインで、[新しいコンソールのプレビューを試す] を選択します。

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソール左側のナビゲーションペインで、[PartiQL エディター] を選択します。
3. [Music] テーブルを選択します。
4. [Query table] (クエリテーブル) を選択します。このアクションで生成したクエリは、完全なテーブルスキャンを実行しません。
5. `partitionKeyValue` を文字列型の値である `Acme Band` に置換します。`sortKeyValue` を文字列型の値である `Happy Day` に置換します。
6. [Run (実行)] ボタンを選択します。

7. [Table view] (テーブルビュー) または [JSON view] (JSON ビュー) ボタンを選択すると、クエリの結果を確認できます。

NoSQL workbench



```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

Attribute type: String　Attribute value: Acme Band

Attribute type: String　Attribute value: PartiQL Rocks

1. [PartiQL statement] (PartiQL ステートメント) を選択します。

2. 次の PartiQL [SELECT statement] (SELECT ステートメント) を入力します。

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. `Artist` および `SongTitle` パラメータの値を指定するには

a. [Optional request parameters] (オプションのリクエストパラメータ) を選択します。

b. [Add new parameters] (パラメータの追加) を選択します。

c. 属性タイプとして [string] (文字列型) を選択し、値に `Acme Band` を選択します。

d. ステップ b とステップ c を繰り返し、[string] (文字列型) のタイプと `PartiQL Rocks` の値を選択します。

4. コードを生成する場合は、[Generate code (コードの生成)] を選択します。

   表示されたタブから目的の言語を選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。

5. オペレーションをすぐに実行する場合は、[Run] (実行) をクリックします。

AWS CLI

1. INSERT PartiQL ステートメントを使用して、`Music` テーブルに項目を作成します。

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. SELECT PartiQL ステートメントを使用して、Music テーブルから項目を取得します。

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

3. UPDATE PartiQL ステートメントを使用して、`Music` テーブルの項目を更新します。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

`Music` テーブルに、項目のリスト値を追加します。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Music テーブルから項目のリスト値を削除します。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Music テーブルに、項目の新しいマップメンバーを追加します。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Music テーブルに、項目の新しい文字列セットの属性を追加します。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Music テーブルで、項目の文字列セットの属性を更新します。

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. DELETE PartiQL ステートメントを使用して、Music テーブルから項目を削除します。

```
aws dynamodb execute-statement --statement "DELETE  FROM Music  \
    WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Java

```
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ConditionalCheckFailedException;
import com.amazonaws.services.dynamodbv2.model.ExecuteStatementRequest;
import com.amazonaws.services.dynamodbv2.model.ExecuteStatementResult;
import com.amazonaws.services.dynamodbv2.model.InternalServerErrorException;
import
 com.amazonaws.services.dynamodbv2.model.ItemCollectionSizeLimitExceededException;
import
 com.amazonaws.services.dynamodbv2.model.ProvisionedThroughputExceededException;
import com.amazonaws.services.dynamodbv2.model.RequestLimitExceededException;
import com.amazonaws.services.dynamodbv2.model.ResourceNotFoundException;
import com.amazonaws.services.dynamodbv2.model.TransactionConflictException;

public class DynamoDBPartiQGettingStarted {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

        try {
            // Create ExecuteStatementRequest
            ExecuteStatementRequest executeStatementRequest = new
 ExecuteStatementRequest();
            List<AttributeValue> parameters= getPartiQLParameters();

            //Create an item in the Music table using the INSERT PartiQL statement
            processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
 value {'Artist':?,'SongTitle':?}", parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
 statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
 where Artist=? and SongTitle=?", parameters));

            //Update an item in the Music table using the UPDATE PartiQL statement.
```

```
           processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist=? and
SongTitle=?", parameters));

           //Add a list value for an item in the Music table.
           processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  where Artist=? and
SongTitle=?", parameters));

           //Remove a list value for an item in the Music table.
           processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2]   where Artist=? and SongTitle=?", parameters));

           //Add a new map member for an item in the Music table.
           processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

           //Add a new string set attribute for an item in the Music table.
           processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

           //update a string set attribute for an item in the Music table.
           processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

           //Retrieve an item from the Music table using the SELECT PartiQL
statement.
           processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

           //delete an item from the Music Table
           processResults(executeStatementRequest(dynamoDB, "DELETE  FROM Music
where Artist=? and SongTitle=?", parameters));
       } catch (Exception e) {
           handleExecuteStatementErrors(e);
       }
   }

   private static AmazonDynamoDB createDynamoDbClient(String region) {
       return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
   }
   private static List<AttributeValue> getPartiQLParameters() {
       List<AttributeValue> parameters = new ArrayList<AttributeValue>();
       parameters.add(new AttributeValue("Acme Band"));
       parameters.add(new AttributeValue("PartiQL Rocks"));
       return parameters;
   }

   private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters ) {
       ExecuteStatementRequest request = new ExecuteStatementRequest();
       request.setStatement(statement);
       request.setParameters(parameters);
       return client.executeStatement(request);
   }

   private static void processResults(ExecuteStatementResult
executeStatementResult) {
       System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());

   }

   // Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (ConditionalCheckFailedException ccfe) {
           System.out.println("Condition check specified in the operation failed,
review and update the condition " +
                                      "check before retrying. Error: " +
ccfe.getErrorMessage());
       } catch (TransactionConflictException tce) {
           System.out.println("Operation was rejected because there is an ongoing
transaction for the item, generally " +
                                      "safe to retry with exponential back-off.
Error: " + tce.getErrorMessage());
       } catch (ItemCollectionSizeLimitExceededException icslee) {
           System.out.println("An item collection is too large, you\'re using Local
Secondary Index and exceeded " +
                                      "size limit of items per
partition key. Consider using Global Secondary Index instead. Error: " +
icslee.getErrorMessage());
```

```
        } catch (Exception e) {
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with 
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for 
your account, increase account level throughput before " +
                                      "retrying. Error: " + 
rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom 
retry strategy make sure to retry with exponential back-off. " +
                                      "Otherwise consider reducing frequency of 
requests or increasing provisioned capacity for your table or secondary index. 
Error: " +
                                      ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists 
before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that 
the request was correctly transmitted to the DynamoDB " +
                                      "service, but for some reason, the service 
was not able to process it, and returned an error response instead. Investigate and 
" +
                                      "configure retry strategy. Error type: " + 
ase.getErrorType() + ". Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that 
the client was unable to get a response from DynamoDB " +
                                      "service, or the client was unable to parse 
the response from the service. Investigate and configure retry strategy. "+
                                      "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure 
retry strategy. Error: " + e.getMessage());
        }
    }

}
```

DynamoDB の PartiQL データ型

次の表に、DynamoDB 用の PartiQL で使用できるデータ型を、一覧で表示します。

DynamoDB データ型	PartiQL 表現	コメント
`Boolean`	TRUE、FALSE	大文字と小文字は区別されません。
`Binary`	該当なし	コード経由でのみサポートされます。
`List`	[value1, value2,...]	List 型に保存できるデータ型に制限はなく、リスト内の要素が同じ型である必要はありません。
`Map`	{ 'name' : value }	Map 型に保存できるデータ型に制限はなく、マップ内の要素が同じ型である必要はありません。
`Null`	NULL	大文字と小文字は区別されません。
`Number`	1, 1.0, 1e0	数値は、正、負、または 0 とすることができます。数値は最大 38 桁の精度を持つことができます。
`Number Set`	<<number1, number2>>	数値セット内の要素は、Number 型でなければなりません。

DynamoDB データ型	PartiQL 表現	コメント
`String Set`	<<'string1', 'string2'>>	文字列セット内の要素は、String 型でなければなりません。
`String`	'string value'	String 型 の値を指定するには、一重引用符を使用する必要があります。

例

次のデータ型を挿入する方法を、次のステートメントで示します。`String`、`Number`、`Map`、`List`、`Number Set` および `String Set`。

```
INSERT INTO TypesTable value {'primarykey':'1',
'NumberType':1,
'MapType' : {'entryname1': 'value', 'entryname2': 4},
'ListType': [1,'stringval'],
'NumberSetType':<<1,34,32,4.5>>,
'StringSetType':<<'stringval','stringval2'>>
}
```

次のステートメントでは、`Map`、`List`、`Number Set` および `String Set` の型に、新しい要素を挿入する方法を示しています。また、`Number` 型の値を変更する方法も示しています。

```
UPDATE TypesTable
SET NumberType=NumberType + 100
SET MapType.NewMapEntry=[2020, 'stringvalue', 2.4]
SET ListType = LIST_APPEND(ListType, [4, <<'string1', 'string2'>>])
SET NumberSetType= SET_ADD(NumberSetType, <<345, 48.4>>)
SET StringSetType = SET_ADD(StringSetType, <<'stringsetvalue1', 'stringsetvalue2'>>)
WHERE primarykey='1'
```

次のステートメントでは、`Map`、`List`、`Number Set` および `String Set` の型から要素を削除する方法を示しています。また、`Number` 型の値を変更する方法も示しています。

```
UPDATE TypesTable
SET NumberType=NumberType - 1
```

```
REMOVE ListType[1]
REMOVE MapType.NewMapEntry
SET NumberSetType = SET_DELETE( NumberSetType, <<345>>)
SET StringSetType = SET_DELETE( StringSetType, <<'stringsetvalue1'>>)
WHERE primarykey='1'
```

詳細については、「DynamoDB のデータ型」を参照してください。

DynamoDB 用の PartiQL ステートメント

Amazon DynamoDB は、次の PartiQL ステートメントをサポートしています。

> Note
>
> DynamoDB はすべての PartiQL ステートメントをサポートしているわけではありません。このリファレンスでは、AWS CLI シェルまたは API で手動で実行する PartiQL ステートメントについて、ベーシックな構文と使用例を示します。

データ操作言語 (DML) は、DynamoDB テーブル内のデータを管理するために使用する、PartiQL ステートメントのセットです。DML ステートメントを使用して、テーブル内のデータを追加、変更、または削除します。

次の DML ステートメントとクエリ言語ステートメントがサポートされています。

- DynamoDB 用の PartiQL select ステートメント
- DynamoDB 用の PartiQL 更新ステートメント
- DynamoDB 用の PartiQL 挿入ステートメント
- DynamoDB 用の PartiQL 削除ステートメント

DynamoDB 用の PartiQL を使用してトランザクションを実行する および DynamoDB 用の PartiQL を使用してバッチ操作を実行する は、DynamoDB 用の PartiQL でもサポートされています。

DynamoDB 用の PartiQL select ステートメント

Amazon DynamoDB では、SELECT ステートメントを使用して、テーブルからデータを取得します。

SELECT ステートメントを使用すると、パーティションキーを持つ等価条件または IN 条件が WHERE 句で指定されていない場合、完全なテーブルスキャンになることがあります。スキャンオペ

レーションは、すべての項目でリクエストされた値を調べるので、大きなテーブルまたはインデックスに対してプロビジョニングされたスループットを 1 回のオペレーションで使い果たすことがあります。

PartiQL で完全なテーブルスキャンを避けたい場合は、次のようにします。

- 完全なテーブルスキャンが行われないように、WHERE 句の条件を適切に設定していることを確かめながら、SELECT ステートメントを作成します。
- DynamoDB デベロッパーガイドの 例: DynamoDB 用の PartiQL で ステートメントの選択を許可し、テーブル全体のスキャンを行うステートメントを拒否する に記載されているように、IAM ポリシーを使用して、完全なテーブルスキャンを無効にします。

詳細については、「DynamoDB デベロッパーガイド」の「データのクエリとスキャンのベストプラクティス」を参照してください。

トピック

- 構文
- パラメータ
- 例

構文

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

パラメータ

expression

(必須) * ワイルドカードからの射影、または結果セットからの 1 つ以上の属性名かドキュメントパスの射影リスト。式には、Amazon DynamoDB での PartiQL 関数の使用 への呼び出し、または DynamoDB での PartiQL 算術演算子、比較演算子、論理演算子 によって変更されたフィールドで構成できます。

table

(必須) クエリするテーブル名。

index

(オプション) クエリを実行するインデックスの名前です。

> **Note**
>
> インデックスにクエリを実行するときは、テーブル名とインデックス名に二重引用符を追加する必要があります。
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condition

(オプション) クエリの選択条件。

> **Important**
>
> SELECT ステートメントによって、完全なテーブルスキャンが行われないようにするには、WHERE 句の条件がパーティションキーを指定する必要があります。等価演算子または IN 演算子を使用します。
> 例えば、`OrderID` パーティションキーと、`Address` を含むその他の非キー属性がある `Orders` テーブルがある場合、次のステートメントは完全なテーブルスキャンを実行しません。
>
> ```
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100
>
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100 and Address='some address'
>
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100 or pk = 200
>
> SELECT *
> FROM "Orders"
> ```

```
WHERE OrderID IN [100, 300, 234]
```

ただし、次の SELECT ステートメントを実行すると、完全なテーブルスキャンが行われます。

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

key

(任意) 返ってきた結果の並び替えに使用するハッシュキーまたはソートキー。デフォルトの順序は昇順 (ASC) です。返ってきた結果を降順に並べる場合は、DESC を指定します。

> **Note**
>
> WHERE 句を省略すると、テーブル内のすべての項目が取得されます。

例

次のクエリは、Orders テーブルでパーティションキーと OrderID を指定し、等価演算子を使用することで、存在する場合に項目を 1 つ返します。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

次のクエリは、OR 演算子を使用することで、特定のパーティションキーと OrderID、値を持つ Orders テーブルから、すべての項目を返します。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

次のクエリは、IN 演算子を使用することで、特定のパーティションキーと OrderID、値を持つ Orders テーブルから、すべての項目を返します。返ってきた結果は、OrderID キー属性の値によって、降順に並べられます。

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

次のクエリは、Total が非キー属性である場合に、Total が 500 以上である Orders テーブルからすべての項目を返す、完全なテーブルスキャンを示します。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

次のクエリは、IN 演算子と非キー属性である Total を使用して、特定の Total 範囲の順序で、Orders テーブルからすべての項目を返す、完全なテーブルスキャンを示します。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

次のクエリは、BETWEEN 演算子と非キー属性である Total を使用して、特定の Total 範囲の順序で、Orders テーブルからすべての項目を返す、完全なテーブルスキャンを示します。

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

次のクエリは、WHERE 句の条件で CustomerID パーティションキーと MovieID ソートキーを指定し、SELECT 句でドキュメントのパスを使用することで、firestick デバイスで視聴を開始した最初の日付を返します。

```
SELECT Devices.FireStick.DateWatched[0]
```

```
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

次のクエリは、テーブル全体のスキャンを示しています。このスキャンでは、WHERE 句の条件でドキュメントのパスを使用し、2019 年 12 月 24 日以降に初めて firestick デバイスを使用した項目のリストを返します。

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

DynamoDB 用の PartiQL 更新ステートメント

UPDATE ステートメントを使用して、Amazon DynamoDB テーブルの項目内にある、1 つ以上の属性の値を変更します。

> **Note**
>
> 一度に更新できる項目は 1 つだけです。1 つの DynamoDB PartiQL ステートメントを発行して、複数の項目を更新することはできません。複数項目の更新については、「DynamoDB 用の PartiQL を使用してトランザクションを実行する」または「DynamoDB 用の PartiQL を使用してバッチ操作を実行する」を参照してください。

トピック

- 構文
- パラメータ
- 戻り値
- 例

構文

```
UPDATE  table
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

パラメータ

`table`

(必須) 修正されるデータを含んでいるテーブル。

`##`

(必須) 作成または変更される属性名、またはドキュメントパス。

`###`

(必須) 属性値またはオペレーションの結果。

SET で使用するためにサポートされている操作は、次の通りです。

- LIST_APPEND: List 型に値を追加します。
- SET_ADD: 数値または文字列セットに値を追加します。
- SET_DELETE: 数値または文字列セットから値を削除します。

`condition`

(必須) 修正される項目の選択条件。この条件は、単一のプライマリキー値を解決する必要があります。

`returnvalues`

(オプション) 属性が更新される前か、更新された後に、表示された項目の属性を取得したい場合に、`returnvalues` を使用します。有効な値は以下のとおりです。

- `ALL OLD *`: 更新操作の前に表示されていた項目について、すべての属性を返します。
- `MODIFIED OLD *`: 更新操作の前に表示されていた属性について、更新された属性だけを返します。
- `ALL NEW *`: 更新操作の後に表示される項目について、すべての属性を返します。
- `MODIFIED NEW *`: `UpdateItem` 操作の後に表示される属性について、更新された属性だけを返します。

戻り値

`returnvalues` パラメータが指定されない限り、このステートメントは値を返しません。

> Note
>
> UPDATE ステートメントの WHERE 句が、DynamoDB テーブルのどの項目も true と評価しない場合、ConditionalCheckFailedException が返ります。

例

既存の項目の属性値を更新します。属性が存在しない場合は、作成されます。

次のクエリは、数値型の属性 (AwardsWon) とマップ型の属性 (AwardDetail) を追加して、"Music" テーブルの項目を更新します。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

RETURNING ALL OLD * を追加すると、Update 操作の前に表示されていた属性を返すことができます。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

以下が返されます。

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

RETURNING ALL NEW * を追加すると、Update 操作の後に表示されていた属性を返すことができます。

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

以下が返されます。

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
}
```

次のクエリは、"Music" テーブルの項目を、AwardDetail.Grammys リストに追加して更新します。

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

次のクエリは、"Music" テーブルの項目を、AwardDetail.Grammys リストから削除して更新します。

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

次のクエリは、"Music" テーブルで AwardDetail マップに BillBoard を追加して、項目を更新します。

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

次のクエリは、"Music" テーブルで、文字列セットの属性 BandMembers を追加して、項目を更新します。

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

次のクエリは、"Music" テーブルで文字列セットの属性 BandMembers に newbandmember を追加して、項目を更新します。

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

DynamoDB 用の PartiQL 削除ステートメント

DELETE ステートメントを使用して、Amazon DynamoDB テーブルにある項目を削除します。

> **Note**
>
> 一度に削除できる項目は 1 つだけです。1 つの DynamoDB PartiQL ステートメントを発行して、複数の項目を削除することはできません。複数のアイテムの削除については、「DynamoDB 用の PartiQL を使用してトランザクションを実行する」または「DynamoDB 用の PartiQL を使用してバッチ操作を実行する」を参照してください。

トピック

- 構文
- パラメータ
- 戻り値
- 例

構文

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

パラメータ

table

(必須) 削除する項目を含む DynamoDB テーブル。

condition

(必須) 削除する項目の選択基準。この条件は、単一のプライマリキー値を解決する必要があります。

returnvalues

(オプション) 削除される前に表示された、項目の属性を取得したい場合には、`returnvalues` を使用します。有効な値は以下のとおりです。

- `ALL OLD *`: 古い項目の内容が返されます。

戻り値

`returnvalues` パラメータが指定されない限り、このステートメントは値を返しません。

> Note
>
> DynamoDB テーブルに、DELETE が発行された項目と同じプライマリキーを持つ項目が無い場合、削除した項目を 0 としたうえで、SUCCESS を返します。テーブルに同じプライマリキーを持つ項目があるが、DELETE ステートメントの WHERE 句の条件で false と評価された場合、`ConditionalCheckFailedException` が返されます。

例

次のクエリは、"Music" テーブルの項目を削除します。

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

RETURNING ALL OLD * を追加すると、削除されたデータを返すことができます。

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

Delete ステートメントは、以下を返すようになりました。

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

DynamoDB 用の PartiQL 挿入ステートメント

INSERT ステートメントを使用して、Amazon DynamoDB のテーブルに項目を追加します。

> **Note**
>
> 一度に挿入できる項目は 1 つだけです。1 つの DynamoDB PartiQL ステートメントを発行して、複数の項目を挿入することはできません。複数の項目の挿入については、「DynamoDB 用の PartiQL を使用してトランザクションを実行する」または「DynamoDB 用の PartiQL を使用してバッチ操作を実行する」を参照してください。

トピック

構文

項目を 1 つ挿入します。

```
INSERT INTO table VALUE item
```

パラメータ

table

(必須) データを挿入するテーブル。このテーブルは既存であることが必要です。

item

(必須) 有効な DynamoDB 項目は PartiQL タプル。1 つの項目のみ指定する必要があります。また、項目の各属性名は大文字と小文字が区別され、PartiQL では一重引用符 (`'...'`) で示されます。

文字列値は、PartiQL では一重引用符 (`'...'`) で示されます。

戻り値

このステートメントは値を返しません。

> Note
>
> DynamoDB テーブルに、挿入される項目と同じプライマリキーを持つ項目が既にある場合、`DuplicateItemException` が返されます。

例

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

Amazon DynamoDB での PartiQL 関数の使用

Amazon DynamoDB の PartiQL は、次の SQL 標準関数の組み込みバージョンをサポートしています。

> Note
>
> このリストに含まれていない SQL 関数は、DynamoDB で現在サポートされていません。

集計関数

- Amazon DynamoDB 用の PartiQL で SIZE 関数を使用する

条件関数

- DynamoDB 用の PartiQL での EXISTS 関数の使用
- DynamoDB 用の PartiQL で ATTRIBUTE_TYPE 関数を使用する
- DynamoDB 用の PartiQL で BEGINS_WITH 関数を使用する
- DynamoDB 用の PartiQL で CONTAINS 関数を使用する
- DynamoDB 用の PartiQL で MISSING 関数を使用する

DynamoDB 用の PartiQL での EXISTS 関数の使用

EXISTS を使用して、`ConditionCheck` が TransactWriteItems API で実行するのと同じ機能を実行できます。EXISTS 関数は、トランザクションでのみ使用できます。

値が指定されると、値が空でないコレクションである場合に `TRUE` を返します。それ以外の場合は `FALSE` を返します。

> Note
>
> この関数は、トランザクション操作でのみ使用できます。

Syntax

```
EXISTS ( statement )
```

引数

statement

(必須) 関数が評価する SELECT ステートメント。

> Note
>
> SELECT 文では、完全なプライマリキーと他の条件を、1 つ指定する必要があります。

戻り型

`bool`

例

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

DynamoDB 用の PartiQL で BEGINS_WITH 関数を使用する

指定した属性が特定の部分文字列から始まる場合、`TRUE` を返します。

Syntax

```
begins_with(path, value )
```

引数

##

(必須) 使用する属性名またはドキュメントのパス。

value

(必須) 検索対象の文字列。

戻り型

`bool`

例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

DynamoDB 用の PartiQL で MISSING 関数を使用する

指定した属性が項目に含まれていない場合、TRUE を返します。この関数では、等価演算子および不等価演算子のみを使用できます。

Syntax

```
 attributename IS | IS NOT  MISSING
```

引数

attributename

(必須) 検索対象の属性名。

戻り型

bool

例

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

DynamoDB 用の PartiQL で ATTRIBUTE_TYPE 関数を使用する

指定したパスの属性が特定のデータ型のものである場合、TRUE を返します。

Syntax

```
attribute_type( attributename, type )
```

引数

attributename

(必須) 使用する属性名。

type

(必須) チェックする属性タイプ。有効な値のリストについては、DynamoDB の「attribute_type」を参照してください。

戻り型

`bool`

例

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

DynamoDB 用の PartiQL で CONTAINS 関数を使用する

パスで指定した属性が次のいずれかである場合、`TRUE` を返します。

- 特定の部分文字列を含む文字列型。
- 特定の要素を含むセット型。

詳細については、DynamoDB の「contains 関数」を参照してください。

Syntax

```
contains( path, substring )
```

引数

##

(必須) 使用する属性名またはドキュメントのパス。

substring

(必須) チェックする対象の属性の部分文字列またはセットのメンバー。詳細については、DynamoDB の「contains 関数」を参照してください。

戻り型

`bool`

例

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Amazon DynamoDB 用の PartiQL で SIZE 関数を使用する

属性のサイズを表す数値をバイト単位で返します。Size で使用できる有効なデータ型は、次の通りです。詳細については、DynamoDB の「size 関数」を参照してください。

Syntax

```
size( path)
```

引数

##

(必須) 属性名またはドキュメントのパス。

サポートされているタイプについては、DynamoDB の「size 関数」を参照してください。

戻り型

`int`

例

```
 SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

DynamoDB での PartiQL 算術演算子、比較演算子、論理演算子

Amazon DynamoDB の PartiQL は、次の SQL 標準演算子をサポートしています。

> **Note**
>
> このリストに含まれていない SQL 演算子は、DynamoDB で現在サポートされていません。

算術演算子

演算子	説明
+	Add
-	- (減算)

比較演算子

演算子	説明
=	Equal to
<>	等しくない
!=	等しくない
>	超
<	未満
>=	以上
<=	以下

論理演算子

演算子	説明
AND	TRUE AND で区切られたすべての条件が TRUE の場合に
BETWEEN	TRUE オペランドが比較の範囲内にある場合に
IN	TRUE オペランドが (最大 50 ハッシュ属性値、または 100 個の非キー属性値において) 式のリストの 1 つに等しい場合

演算子	説明
IS	TRUE オペランドが指定されていて、PartiQL のデータ型であり、NULL または MISSING を含む場合に
NOT	指定されたブール式の値を反転する
OR	TRUE OR で区切られた条件のいずれかが TRUEの場合に

DynamoDB 用の PartiQL を使用してトランザクションを実行する

このセクションでは、DynamoDB 用の PartiQL でトランザクションを使用する方法について説明します。PartiQL トランザクションは、合計 100 件のステートメント (アクション) に制限されています。

DynamoDB トランザクションの詳細については、「DynamoDB トランザクションで複雑なワークフローを管理する」を参照してください。

> **Note**
>
> トランザクション全体は、読み込みステートメントまたは 書き込みステートメントのいずれかで構成されている必要があります。1 つのトランザクションで両方を混在させることはできません。この EXISTS 関数は例外です。TransactWriteItems API オペレーションの ConditionCheck と同様に、項目の特定の属性の状態を確認するために使用できます。

トピック

- 構文
- パラメータ
- 戻り値
- 例

構文

```
[
    {
       "Statement":" statement ",
       "Parameters":[
          {
             " parametertype " : " parametervalue "
          }, ...]
    } , ...
]
```

パラメータ

statement

(必須) DynamoDB 用の PartiQL がサポートするステートメントです。

> Note
>
> トランザクション全体は、読み込みステートメントまたは 書き込みステートメントのいずれかで構成されている必要があります。1 つのトランザクションで両方を混在させることはできません。

parametertype

(オプション) PartiQL ステートメントを指定するときにパラメータが使用された場合の DynamoDB タイプ。

parametervalue

(オプション) PartiQL ステートメントを指定するときにパラメータが使用された場合のパラメータの値。

戻り値

このステートメントは、書き込み操作 (INSERT、UPDATE、DELETE) の値を返しません。ただし、WHERE 句で指定された条件に基づいて、読み取り操作 (SELECT) に対して異なる値を返します。

> **Note**
>
> シングルトンの INSERT、UPDATE、または DELETE 操作のいずれかでエラーが返された場合、`TransactionCanceledException` 例外によりトランザクションがキャンセルされます。キャンセル理由コードには、個々のシングルトン操作からのエラーが含まれます。

例

以下の例では、トランザクションとして複数のステートメントを実行します。

AWS CLI

1. 次の JSON コードを partiql.json というファイルに保存します

```
[
    {
        "Statement": "EXISTS(SELECT * FROM \"Music\" where Artist='No One You
 Know' and SongTitle='Call Me Today' and Awards is  MISSING)"
    },
    {
        "Statement": "INSERT INTO Music value {'Artist':?,'SongTitle':'?'}",
        "Parameters": [{\"S\": \"Acme Band\"}, {\"S\": \"Best Song\"}]
    },
    {
        "Statement": "UPDATE \"Music\" SET AwardsWon=1 SET
 AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
 SongTitle='PartiQL Rocks'"
    }
]
```

2. コマンドプロンプトで、次のコマンドを実行します。

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlTransaction {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
```

```
       AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

       try {
           // Create ExecuteTransactionRequest
           ExecuteTransactionRequest executeTransactionRequest =
createExecuteTransactionRequest();
           ExecuteTransactionResult executeTransactionResult =
dynamoDB.executeTransaction(executeTransactionRequest);
           System.out.println("ExecuteTransaction successful.");
           // Handle executeTransactionResult

       } catch (Exception e) {
           handleExecuteTransactionErrors(e);
       }
   }

   private static AmazonDynamoDB createDynamoDbClient(String region) {
       return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
   }

   private static ExecuteTransactionRequest createExecuteTransactionRequest() {
       ExecuteTransactionRequest request = new ExecuteTransactionRequest();

       // Create statements
       List<ParameterizedStatement> statements = getPartiQLTransactionStatements();

       request.setTransactStatements(statements);
       return request;
   }

   private static List<ParameterizedStatement> getPartiQLTransactionStatements() {
       List<ParameterizedStatement> statements = new
ArrayList<ParameterizedStatement>();

       statements.add(new ParameterizedStatement()
                              .withStatement("EXISTS(SELECT * FROM "Music" where
Artist='No One You Know' and SongTitle='Call Me Today' and Awards is  MISSING)"));

       statements.add(new ParameterizedStatement()
                              .withStatement("INSERT INTO "Music" value
{'Artist':'?','SongTitle':'?'}")
                              .withParameters(new AttributeValue("Acme Band"),new
AttributeValue("Best Song")));
```

```
        statements.add(new ParameterizedStatement()
                              .withStatement("UPDATE "Music" SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
SongTitle='PartiQL Rocks'"));

        return statements;
    }

    // Handles errors during ExecuteTransaction execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleExecuteTransactionErrors(Exception exception) {
        try {
            throw exception;
        } catch (TransactionCanceledException tce) {
            System.out.println("Transaction Cancelled, implies a client issue, fix
before retrying. Error: " + tce.getErrorMessage());
        } catch (TransactionInProgressException tipe) {
            System.out.println("The transaction with the given request token is
already in progress, consider changing " +
                "retry strategy for this type of error. Error: " +
tipe.getErrorMessage());
        } catch (IdempotentParameterMismatchException ipme) {
            System.out.println("Request rejected because it was retried with a
different payload but with a request token that was already used, " +
                "change request token for this payload to be accepted. Error: " +
ipme.getErrorMessage());
        } catch (Exception e) {
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
 provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
 it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

次の例は、DynamoDB が WHERE 句で指定された異なる条件を持つ項目を読み取るときのさまざまな戻り値を示しています。

AWS CLI

1. 次の JSON コードを partiql.json というファイルに保存します

```
[
    // Item exists and projected attribute exists
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One You Know' and
 SongTitle='Call Me Today'"
    },
    // Item exists but projected attributes do not exist
    {
        "Statement": "SELECT non_existent_projected_attribute FROM "Music" WHERE
 Artist='No One You Know' and SongTitle='Call Me Today'"
    },
    // Item does not exist
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One I Know' and
 SongTitle='Call You Today'"
    }
]
```

2. コマンドプロンプトで、次のコマンドを実行します。

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. 以下のレスポンスが返されます。

```
{
    "Responses": [
        // Item exists and projected attribute exists
        {
            "Item": {
                "Artist":{
                    "S": "No One You Know"
                },
                "SongTitle":{
                    "S": "Call Me Today"
                }
            }
        },
        // Item exists but projected attributes do not exist
        {
            "Item": {}
        },
        // Item does not exist
        {}
    ]
}
```

DynamoDB 用の PartiQL を使用してバッチ操作を実行する

このセクションでは、DynamoDB 用の PartiQL でバッチステートメントを使用する方法について説明します。

> Note
>
> - バッチ全体は、読み取りステートメントまたは書き込みステートメントのいずれかで構成する必要があります。1 つのバッチに両方を混在させることはできません。
> - `BatchExecuteStatement` と `BatchWriteItem` が実行できるステートメントの数は、バッチあたり 25 が上限です。

トピック

- 構文
- パラメータ
- 例

構文

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

パラメータ

statement

(必須) DynamoDB 用の PartiQL がサポートするステートメントです。

> **Note**
>
> - バッチ全体は、読み取りステートメントまたは書き込みステートメントのいずれかで構成する必要があります。1 つのバッチに両方を混在させることはできません。
> - `BatchExecuteStatement` と `BatchWriteItem` が実行できるステートメントの数は、バッチあたり 25 が上限です。

`parametertype`

(オプション) PartiQL ステートメントを指定するときにパラメータが使用された場合の DynamoDB タイプ。

`parametervalue`

(オプション) PartiQL ステートメントを指定するときにパラメータが使用された場合のパラメータの値。

例

AWS CLI

1. 次の json を partiql.json というファイルに保存します

```
[
   {
  "Statement": "INSERT INTO Music value {'Artist':'?','SongTitle':'?'}",
   "Parameters": [{"S": "Acme Band"}, {"S": "Best Song"}]
 },
 {
  "Statement": "UPDATE Music SET AwardsWon=1 SET AwardDetail={'Grammys':[2020,
 2018]} where Artist='Acme Band' and SongTitle='PartiQL Rocks'"
    }
]
```

2. コマンドプロンプトで、次のコマンドを実行します。

```
aws dynamodb batch-execute-statement  --statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlBatch {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create BatchExecuteStatementRequest
            BatchExecuteStatementRequest batchExecuteStatementRequest =
 createBatchExecuteStatementRequest();
            BatchExecuteStatementResult batchExecuteStatementResult =
 dynamoDB.batchExecuteStatement(batchExecuteStatementRequest);
            System.out.println("BatchExecuteStatement successful.");
            // Handle batchExecuteStatementResult

        } catch (Exception e) {
            handleBatchExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {

        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static BatchExecuteStatementRequest createBatchExecuteStatementRequest()
 {
        BatchExecuteStatementRequest request = new BatchExecuteStatementRequest();

        // Create statements
        List<BatchStatementRequest> statements = getPartiQLBatchStatements();

        request.setStatements(statements);
        return request;
    }

    private static List<BatchStatementRequest> getPartiQLBatchStatements() {
        List<BatchStatementRequest> statements = new
 ArrayList<BatchStatementRequest>();

        statements.add(new BatchStatementRequest()
                               .withStatement("INSERT INTO Music value 
 {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"));

        statements.add(new BatchStatementRequest()
                               .withStatement("UPDATE Music set 
 AwardDetail.BillBoard=[2020] where Artist='Acme Band' and SongTitle='PartiQL 
 Rocks'"));

        return statements;
    }

    // Handles errors during BatchExecuteStatement execution. Use recommendations in 
 error messages below to add error handling specific to 
    // your application use-case.
    private static void handleBatchExecuteStatementErrors(Exception exception) {
        try {
            throw exception;
        } catch (Exception e) {
            // There are no API specific errors to handle for BatchExecuteStatement, 
 common DynamoDB API errors are handled below
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with 
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for 
 your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom 
 retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing 
 provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists 
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
 it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

DynamoDB 用 PartiQL における IAM セキュリティポリシー

以下のアクセス権限が必要です。

- DynamoDB 用の PartiQL を使用して項目を読み込むには、`dynamodb:PartiQLSelect` アクセス許可を、テーブルまたはインデックスに付与します。
- DynamoDB 用の PartiQL を使用して項目を挿入するには、`dynamodb:PartiQLInsert` アクセス許可をテーブルまたはインデックスに付与します。
- DynamoDB 用の PartiQL を使用して項目を更新するには、`dynamodb:PartiQLUpdate` アクセス許可をテーブルまたはインデックスに付与します。
- DynamoDB 用の PartiQL を使用して項目を削除するには、`dynamodb:PartiQLDelete` アクセス許可をテーブルまたはインデックスに付与します。

例: テーブルに対して、すべての DynamoDB 用の PartiQL ステートメントを許可する (選択、挿入、更新、削除)

次の IAM ポリシーは、すべての DynamoDB 用の PartiQL ステートメントをテーブルに対して実行するための、アクセス許可を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

例: テーブルに対して、DynamoDB 用の PartiQL 選択ステートメントを許可する

次の IAM ポリシーは、特定のテーブルに対して `select` ステートメントを実行するためのアクセス許可を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

例: インデックスに対して、DynamoDB 用の PartiQL 挿入ステートメントを許可する

次の IAM ポリシーは、特定のインデックスに対して `insert` ステートメントを実行するためのアクセス許可を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
      }
   ]
}
```

例: テーブルに対して、DynamoDB 用の PartiQL トランザクションステートメントを許可する

次の IAM ポリシーは、特定のテーブルに対してトランザクションステートメントだけを実行するためのアクセス許可を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      }
   ]
}
```

例: テーブルに対して、DynamoDB 用の PartiQL 非トランザクションの読み込み/書き込みステートメントを許可し、PartiQL 読み込み/書き込みトランザクションをブロックします。

次の IAM ポリシーは、DynamoDB 用の PartiQL トランザクション読み込み/書き込みをブロックしている間に、DynamoDB 用の PartiQL 非トランザクションの読み込み/書き込みを実行するためのアクセス許可を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

例: DynamoDB 用の PartiQL で ステートメントの選択を許可し、テーブル全体のスキャンを行うステートメントを拒否する

次の IAM ポリシーは、テーブル全体のスキャンを行う `select` ステートメントをブロックしている間に、特定のテーブルに対して `select` ステートメントを実行するためのアクセス許可を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ],
         "Condition":{
            "Bool":{
               "dynamodb:FullTableScan":[
                  "true"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ]
      }
   ]
}
```

セカンダリインデックスを使用したデータアクセス性の向上

トピック

- DynamoDB のグローバルセカンダリインデックスの使用
- ローカルセカンダリインデックス

Amazon DynamoDB によって、プライマリキーの値を指定して、テーブルの項目に高速アクセスすることが可能になります。しかし多くのアプリケーションでは、プライマリキー以外の属性を使って、データに効率的にアクセスできるようにセカンダリ (または代替) キーを 1 つ以上設定することで、メリットが得られることがあります。これに対応するために、1 つのテーブルで 1 つ以上のセカンダリインデックスを作成して、それらのインデックスに対して Query または Scan リクエストを実行することができます。

セカンダリインデックス は、テーブルからの属性のサブセットと、Query オペレーションをサポートする代替キーで構成されるデータ構造です。Query をテーブルで使用する場合と同じように、Query を使用してインデックスからデータを取得できます。テーブルには、複数のセカンダリインデックスを含めることができます。これにより、アプリケーションは複数の異なるクエリパターンにアクセスできます。

> Note
>
> また、テーブルを Scan するのと同じように、インデックスも Scan できます。

すべてのセカンダリインデックスは 1 つのテーブルのみに関連付けられ、そこからデータを取得します。これはインデックスのベーステーブルと呼ばれます。インデックスを作成する場合は、インデックスの代替キー (パーティションキーおよびソートキー) を定義します。また、ベーステーブルからインデックスに射影 (コピー) したい属性を定義します。DynamoDB では、これらの属性とベーステーブルからのプライマリキー属性がインデックスにコピーされます。次に、テーブルに対してクエリまたはスキャンを実行する場合と同様に、インデックスに対してクエリまたはスキャンを実行します。

すべてのセカンダリインデックスは、DynamoDB によって自動的にメンテナンスされます。ベーステーブルの項目を追加、変更、または削除すると、そのテーブルのインデックスも更新され、この変更が反映されます。

DynamoDB は、次の 2 種類のセカンダリインデックスをサポートしています。

- グローバルセカンダリインデックス — パーティションキーおよびソートキーを持つインデックス。ベーステーブルのものとは異なる場合があります。このインデックスに関するクエリがすべてのパーティションにまたがり、ベーステーブル内のすべてのデータを対象とする可能性があるため、グローバルセカンダリインデックスは「グローバル」と見なされます。グローバルセカンダリインデックスは、ベーステーブルとは別に独自のパーティション領域に保存され、ベーステーブルとは別にスケーリングします。
- ローカルセカンダリインデックス — パーティションキーはベーステーブルと同じで、ソートキーが異なるインデックス。ローカルセカンダリインデックスは、ローカルセカンダリインデックスのすべてのパーティションの範囲が同じパーティションキーバリューを持つベーステーブルのパーティションに限定されるという意味で「ローカル」です。

グローバルセカンダリインデックスとローカルセカンダリインデックスの比較については、この動画を参照してください。

GSI と LSI のうち正しい選択を行う

使用するインデックスの種類を決定するときは、アプリケーションの要件を考慮する必要があります。次の表に、グローバルセカンダリインデックスとローカルセカンダリインデックスの主な違いを示します。

特徴	グローバルセカンダリインデックス	ローカルセカンダリインデックス
キースキーマ	グローバルセカンダリインデックスのプライマリキーはシンプル (パーティションキー) または複合 (パーティションキーとソートキー) のいずれかとすることができます。	ローカルセカンダリインデックスのプライマリキーは複合 (パーティションキーとソートキー) である必要があります。
キーの属性	インデックスパーティションキーとソートキー (存在する場合) は、文字列、数値、またはバイナリ型の任意のベーステーブル属性とすることができます。	インデックスのパーティションキーは、ベーステーブルのパーティションキーと同じ属性です。ソートキーは、文字列、数値、またはバイナリ型の任意のベーステーブル属性とすることができます。

特徴	グローバルセカンダリインデックス	ローカルセカンダリインデックス
パーティションキー値ごとのサイズ制限	グローバルセカンダリインデックスのサイズ制限はありません。	パーティションキーの値ごとに、すべてのインデックス付き項目の合計サイズが、10 GB 以下である必要があります。
オンラインインデックスオペレーション	グローバルセカンダリインデックスは、テーブルの作成と同時に作成できます。また、新しいグローバルセカンダリインデックスを既存のテーブルに追加したり、既存のグローバルセカンダリインデックスを削除したりすることもできます。詳細については、「」を参照してくださいグローバルセカンダリインデックスの管理	ローカルセカンダリインデックスは、テーブルの作成と同時に作成されます。ローカルセカンダリインデックスを既存のテーブルに追加したり、既存のローカルセカンダリインデックスを削除したりすることはできません。
クエリとパーティション	グローバルセカンダリインデックスでは、すべてのパーティションでテーブル全体に対してクエリを実行できます。	ローカルセカンダリインデックスでは、クエリのパーティションキー値で指定された 1 つのパーティションに対してクエリを実行できます。
読み込み整合性	グローバルセカンダリインデックスのクエリは結果整合性をサポートします。	ローカルセカンダリインデックスのクエリを実行するとき、結果整合性または強い整合性のどちらかを選択できます。

特徴	グローバルセカンダリインデックス	ローカルセカンダリインデックス
プロビジョニングされたスループットの消費	各グローバルセカンダリインデックスには、読み込み/書き込みアクティビティに対する独自のプロビジョニングされたスループット設定があります。グローバルセカンダリインデックスのクエリまたはスキャンでは、ベーステーブルからではなく、インデックスからキャパシティーユニットを使用します。同じことが、テーブルへの書き込みによるグローバルセカンダリインデックスの更新にも当てはまります。グローバルテーブルに関連付けられたグローバルセカンダリインデックスは、書き込みキャパシティユニットを使用します。	ローカルセカンダリインデックスのクエリまたはスキャンでは、ベーステーブルから読み込みキャパシティーユニットを使用します。テーブルに書き込むと、そのローカルセカンダリインデックスも更新されます。この更新では、ベーステーブルから書き込みキャパシティユニットを使用します。グローバルテーブルに関連付けられたローカルセカンダリインデックスは、レプリケートされた書き込みキャパシティユニットを使用します。
射影される属性	グローバルセカンダリインデックスのクエリまたはスキャンでは、インデックスに射影された属性のみをリクエストできます。DynamoDB では、テーブルから属性をフェッチしません。	ローカルセカンダリインデックスをクエリまたはスキャンする場合、インデックスに射影されていない属性をリクエストできます。DynamoDB は、これらの属性をテーブルから自動的に取得します。

セカンダリインデックスを持つテーブルを複数作成する場合は、順次作成する必要があります。たとえば、最初のテーブルを作成し、そのテーブルが `ACTIVE` になるまで待ちます。次のテーブルを作成し、そのテーブルが `ACTIVE` になるまで待ちます。セカンダリインデックスを持つ複数のテーブルを同時に作成しようとすると、DynamoDB は `LimitExceededException` を返します。

各セカンダリインデックスは、関連付けられているベーステーブルと同じテーブルクラスとキャパシティモードを使用します。各セカンダリインデックスには、以下を指定する必要があります。

- 作成するインデックスのタイプ – グローバルセカンダリインデックスまたはローカルセカンダリインデックスのいずれか。
- インデックスの名前。インデックスの名前付けルールは、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」に示すようにテーブルの場合と同じです。名前は関連付けられているベーステーブルに対して一意である必要がありますが、別のベーステーブルに関連付けられているインデックスでも同じ名前を使用できます。
- インデックスのキースキーマ。インデックスキースキーマの各属性は、型が `String`、`Number`、または `Binary` の最上位属性である必要があります。ドキュメントとセットを含むその他のデータ型は使用できません。キースキーマのその他の要件は、インデックスの種類によって異なります。
  - グローバルセカンダリインデックスの場合、パーティションキーはベーステーブルの任意のスカラー属性にすることができます。ソートキーはオプションです。このキーもベーステーブルの任意のスカラー属性にすることができます。
  - ローカルセカンダリインデックスの場合、パーティションキーは、ベーステーブルのパーティションキーと同じである必要があります。ソートキーは、非キーベーステーブル属性である必要があります。
- ベーステーブルからインデックスに射影する追加の属性 (ある場合)。この属性は、すべてのインデックスに自動的に射影されるテーブルのキー属性とは別の属性です。スカラー、ドキュメント、およびセットを含む任意のデータ型の属性を射影できます。
- インデックスのプロビジョニングされたスループット設定 (必要な場合) :
  - グローバルセカンダリインデックスの場合、読み込み/書き込みキャパシティーユニット設定を指定する必要があります。このプロビジョニングされたスループット設定は、ベーステーブルの設定から独立しています。
  - ローカルセカンダリインデックスの場合、読み込み/書き込みキャパシティーユニット設定を指定する必要はありません。ローカルセカンダリインデックスに対する読み込み/書き込みオペレーションは、そのベーステーブルのプロビジョニングされたスループット設定から使用します。

クエリに最大限の柔軟性を持たせるために、テーブルごとに最大 20 個のグローバルセカンダリインデックス (デフォルトのクォータ) および最大 5 個のローカルセカンダリインデックスを作成できます。

テーブルあたりのグローバルセカンダリインデックスのクォータは、次の AWS リージョンでは 5 です。

- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)
- 欧州 (ストックホルム)

テーブルのセカンダリインデックスの詳細なリストを取得するには、DescribeTable オペレーションを使用します。DescribeTable は、テーブルのすべてのセカンダリインデックスの名前、ストレージサイズ、および項目数を返します。これらの値はリアルタイムでは更新されませんが、約 6 時間ごとに更新されます。

セカンダリインデックスのデータには、Query または Scan オペレーションを使用してアクセスできます。使用するベーステーブル名とインデックス名、結果で返される属性、および適用する条件式またはフィルタを指定する必要があります。DynamoDB は、結果を昇順で返すことも降順で返すこともできます。

テーブルを削除すると、そのテーブルに関連付けられているすべてのインデックスも削除されます。

ベストプラクティスについては、DynamoDB でセカンダリインデックスを使用するためのベストプラクティス。 を参照してください。

DynamoDB のグローバルセカンダリインデックスの使用

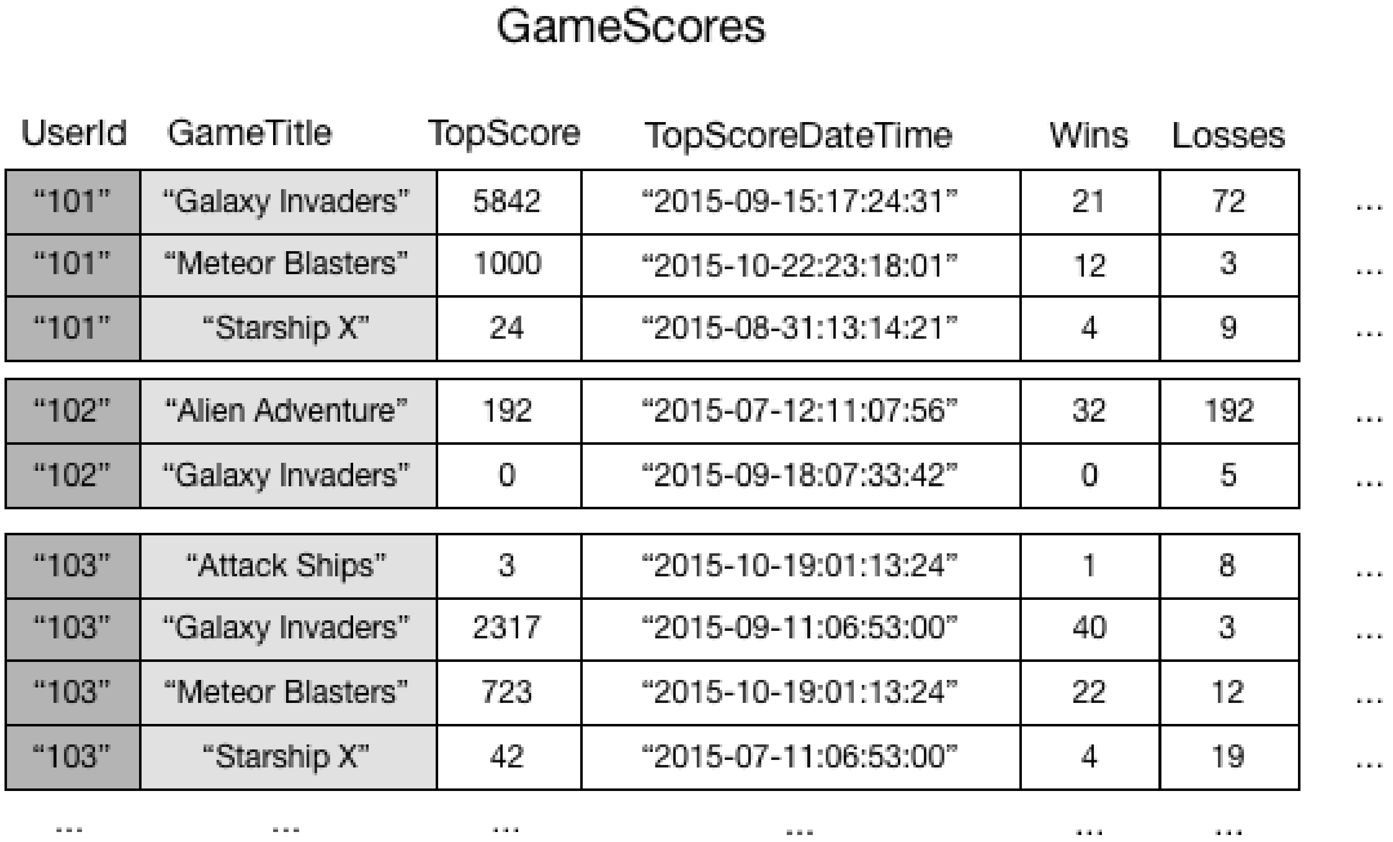
アプリケーションによっては、さまざまな属性をクエリ基準に使用して、いろいろな種類のクエリを実行する必要があります。このような要件に対応するために、1 つまたは複数のグローバルセカンダリインデックスを作成して、Amazon DynamoDB でこれらのインデックスに対して Query リクエストを発行できます。

トピック

- グローバルセカンダリインデックスのストレージに関する考慮事項
- グローバルセカンダリインデックスの管理
- グローバルセカンダリインデックスの操作: Java
- グローバルセカンダリインデックスの操作: .NET
- グローバルセカンダリインデックスの操作: AWS CLI

シナリオ: グローバルセカンダリインデックスの使用

たとえば、GameScores という名前のテーブルがあり、モバイルゲームアプリケーションのユーザーとスコアを記録しているとします。GameScores の各項目は、パーティションキー (UserId) およびソートキー (GameTitle) で特定されます。次の図は、テーブル内の項目の構成を示しています。一部表示されていない属性もあります。



GameScores

UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	

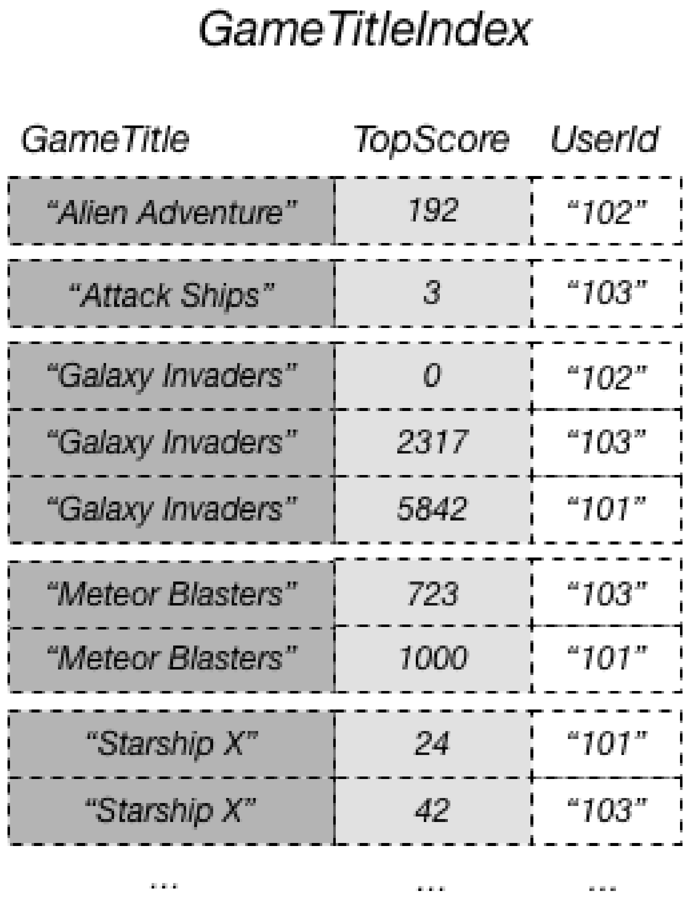
ここで、各ゲームの最高スコアを表示する順位表アプリケーションを作成すると仮定します。キー属性 (UserId と GameTitle) を指定したクエリは非常に効率的です。ただし、アプリケーションが GameScores だけに基づいて GameTitle からデータを取り出す必要がある場合、Scan オペレー

ションを使用する必要があります。テーブルに追加される項目が増えるにつれ、すべてのデータのスキャンは低速で非効率的になります。このため、次のような質問に答えることが難しくなります。

- ゲーム Meteor Blasters で記録された最高スコアはいくつですか?
- Galaxy Invaders で最高スコアを獲得したユーザーは誰ですか?
- 勝敗の最も高い比率は何ですか?

グローバルセカンダリインデックスを作成して、非キー属性に対するクエリの速度を上げることができます。グローバルセカンダリインデックスには、ベーステーブルからの属性の一部が格納されますが、それらはテーブルのプライマリキーとは異なるプライマリキーによって構成されます。インデックスキーは、テーブルからのキー属性を持つ必要がありません。また、テーブルと同じキースキーマを使用する必要もありません。

例えば、パーティションキーとして `GameTitle`、ソートキーとして `TopScore` を使用して、`GameTitleIndex` という名前のグローバルセカンダリインデックスを作成できます。ベーステーブルのプライマリキー属性は必ずインデックスに射影されるので、`UserId` 属性も存在します。次の図は、`GameTitleIndex` インデックスを示しています。



GameTitleIndex

GameTitle	TopScore	UserId
“Alien Adventure”	192	“102”
“Attack Ships”	3	“103”
“Galaxy Invaders”	0	“102”
“Galaxy Invaders”	2317	“103”
“Galaxy Invaders”	5842	“101”
“Meteor Blasters”	723	“103”
“Meteor Blasters”	1000	“101”
“Starship X”	24	“101”
“Starship X”	42	“103”
...	...	...

これで、GameTitleIndex に対してクエリを実行し、Meteor Blasters のスコアを簡単に入手できるようになります。結果は、ソートキーの値 TopScore 別に並べられます。ScanIndexForward パラメータを false に設定した場合、結果は降順で返されます。つまり、最高スコアが最初に返されます。

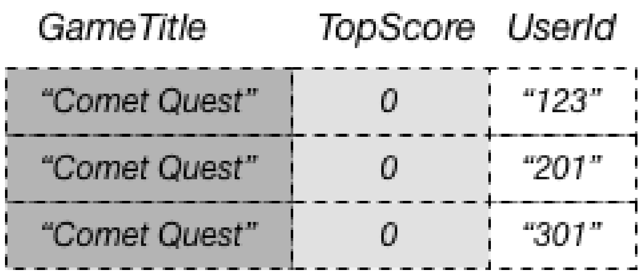
すべてのグローバルセカンダリインデックスには、パーティションキーが必要で、オプションのソートキーを指定できます。インデックスキースキーマは、テーブルスキーマとは異なるものにすることができます。シンプルなプライマリキー (パーティションキー) を持つテーブルを作成し、複合プライマリキー (パーティションキーおよびソートキー) を使用してグローバルセカンダリインデックスを作成できます。またはその逆もあります。インデックスキー属性は、はベーステーブルからの任意の最上位 String、Number、または Binary 属性で構成できます。その他のスカラー型、ドキュメント型、およびセット型は許可されません。

必要な場合、他のベーステーブル属性をインデックスに射影できます。インデックスのクエリを行うと、DynamoDB ではこれらの射影された属性を効率的に取り出すことができます。ただし、グローバルセカンダリインデックス のクエリでは、ベーステーブルから属性をフェッチできません。たとえば、上記の図に示すように、GameTitleIndex にクエリを実行した場合、クエリは TopScore 以外の非キー属性にアクセスすることはできません (キー属性 GameTitle と UserId は自動的に射影されます)。

DynamoDB テーブルでは、各キー値は一意である必要があります。ただし、グローバルセカンダリインデックスのキー値は一意である必要はありません。たとえば、Comet Quest という名前のゲームが特に難しく、多くの新しいユーザーが試しても、ゼロを上回るスコアを獲得することができないとします。以下にこれを表すいくつかのデータを示します。

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

このデータを GameScores テーブルに追加すると、DynamoDB はそのデータを GameTitleIndex に伝達します。GameTitle として Comet Quest を、TopScore として 0 を指定してインデックスに対するクエリを実行すると、次のデータが返されます。



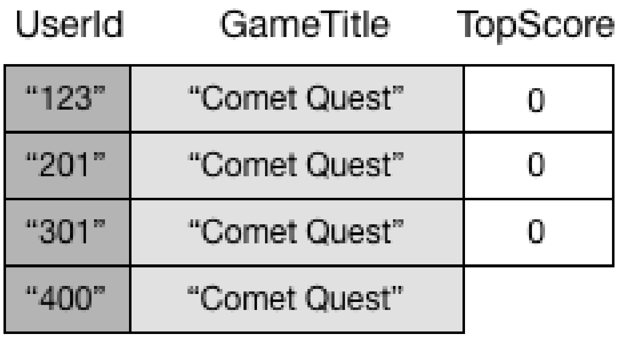
GameTitle	TopScore	UserId
“Comet Quest”	0	“123”
“Comet Quest”	0	“201”
“Comet Quest”	0	“301”

指定したキー値を持つ項目だけがレスポンスに表示されます。その一連のデータ内では、項目は特定の順に並んでいません。

グローバルセカンダリインデックスは、キー属性が実際に存在するデータ項目のみを追跡します。たとえば、別の新しい項目を `GameScores` テーブルに追加し、必須のプライマリキー属性だけを指定したとします。

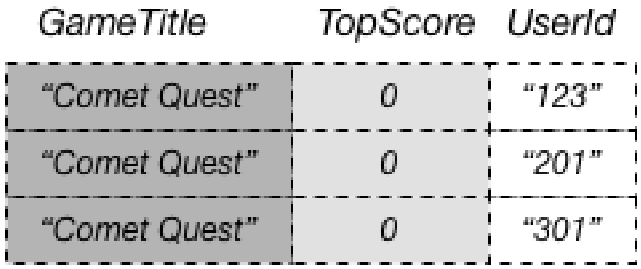
UserId	GameTitle
400	Comet Quest

`TopScore` 属性を指定していないので、DynamoDB は、この項目を `GameTitleIndex` に伝達しません。このため、すべての Comet Quest 項目を対象に、`GameScores` に対してクエリを実行した場合、次の 4 つの項目が返されます。



UserId	GameTitle	TopScore
“123”	“Comet Quest”	0
“201”	“Comet Quest”	0
“301”	“Comet Quest”	0
“400”	“Comet Quest”	

`GameTitleIndex` に対して同様のクエリを実行すると、4 つではなく 3 つの項目が返されます。これは、`TopScore` が存在しない項目はインデックスに反映されないためです。



GameTitle	TopScore	UserId
“Comet Quest”	0	“123”
“Comet Quest”	0	“201”
“Comet Quest”	0	“301”

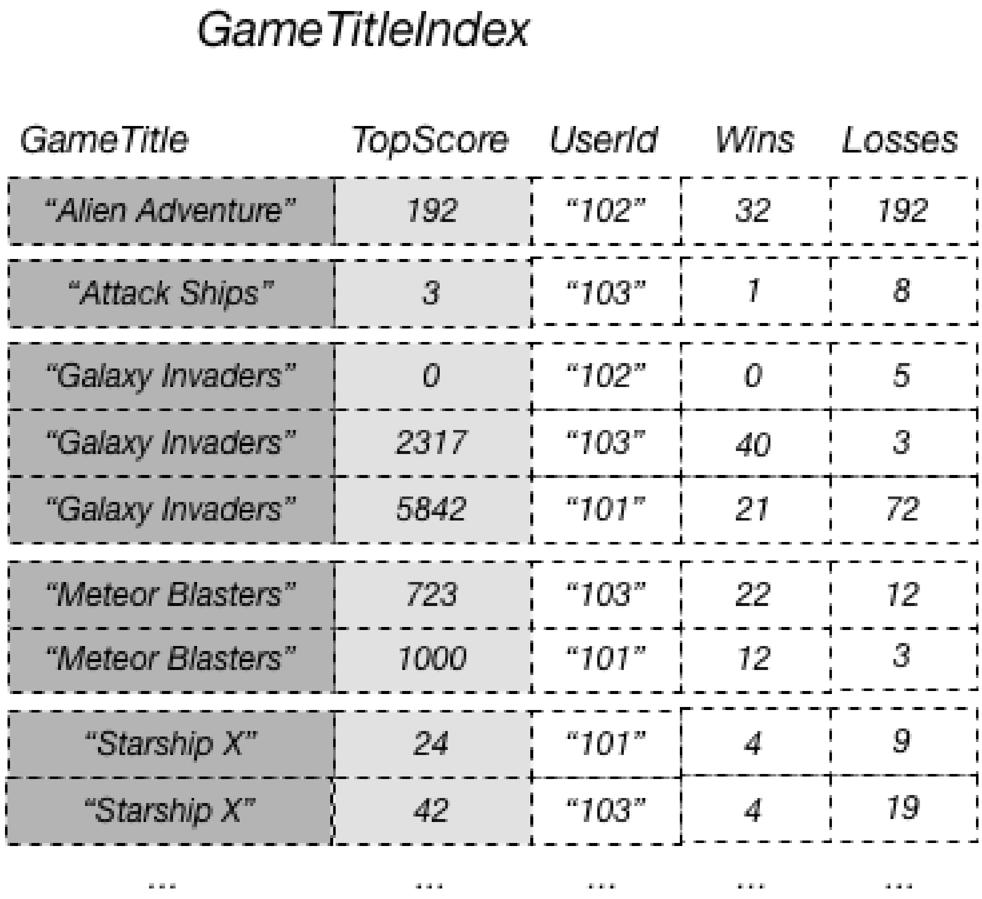
属性の射影

射影とは、テーブルからセカンダリインデックスにコピーされる属性のセットです。テーブルのパーティションキーとソートキーは常にインデックスに射影されます。アプリケーションのクエリ要件をサポートするために、他の属性を射影できます。インデックスをクエリすると、Amazon DynamoDB は、その属性が自身のテーブル内にあるかのように、プロジェクション内の任意の属性にアクセスできます。

セカンダリインデックスを作成するときは、インデックスに射影される属性を指定する必要があります。DynamoDB には、そのために次の 3 つのオプションが用意されています。

- KEYS_ONLY – インデックス内の各項目は、テーブルパーティションキーとソートキーの値、およびインデックスキーの値のみで構成されます。`KEYS_ONLY` オプションを指定すると、セカンダリインデックスが最小になります。
- INCLUDE – `KEYS_ONLY` の属性に加えて、セカンダリインデックスにその他の非キー属性が含まれるように指定できます。
- ALL – セカンダリインデックスには、ソーステーブルのすべての属性が含まれます。すべてのテーブルデータがインデックスに複製されるため、`ALL` の射影にすると、セカンダリインデックスが最大になります。

前の図では、`GameTitleIndex` には 1 つの射影された属性 `UserId` のみがあります。そのため、アプリケーションはクエリで `UserId` および `GameTitle` を使用して各ゲームのトップスコアラーの `TopScore` を効率的に判断できますが、トップスコアラーの勝敗の最も高い比率は効率的に判断できません。そのためには、基本テーブルで追加のクエリを実行して、各トップスコアラーの勝敗を取得する必要があります。このデータに対するクエリのサポートを効率化する方法として、次の図に示すように、これらの属性をベーステーブルからグローバルセカンダリインデックスに射影する方法があります。



GameTitleIndex

GameTitle	TopScore	UserId	Wins	Losses
“Alien Adventure”	192	“102”	32	192
“Attack Ships”	3	“103”	1	8
“Galaxy Invaders”	0	“102”	0	5
“Galaxy Invaders”	2317	“103”	40	3
“Galaxy Invaders”	5842	“101”	21	72
“Meteor Blasters”	723	“103”	22	12
“Meteor Blasters”	1000	“101”	12	3
“Starship X”	24	“101”	4	9
“Starship X”	42	“103”	4	19
...	...	...	...	...

非キー属性 `Wins` と `Losses` がインデックスに射影されるので、アプリケーションは、任意のゲーム、またはゲームとユーザー ID の任意の組み合わせに対して、勝敗の比率を特定できます。

グローバルセカンダリインデックスに射影する属性を選択する場合には、プロビジョニングされるスループットコストとストレージコストのトレードオフを考慮する必要があります。

- ごく一部の属性だけに最小のレイテンシーでアクセスする必要がある場合は、それらの属性だけをグローバルセカンダリインデックスに射影することを検討してください。インデックスが小さいほど少ないコストで格納でき、書き込みコストも低くなります。
- アプリケーションが非キー属性に頻繁にアクセスする場合には、それらの属性をグローバルセカンダリインデックスに射影することを検討してください。グローバルセカンダリインデックスのストレージコストの増加は、頻繁にテーブルスキャンを実行するコストの減少で相殺されます。
- ほとんどの非キー属性に頻繁にアクセスする場合は、それらの属性を (場合によっては、ベーステーブル全体を) グローバルセカンダリインデックスに射影することができます。これにより、最大限の柔軟性が得られます。ただし、ストレージコストは増加するか、倍になります。

- アプリケーションでテーブルのクエリを頻繁に行う必要がなく、テーブル内のデータに対する書き込みや更新が多数になる場合は、KEYS_ONLY を射影することを検討してください。グローバルセカンダリインデックスは、最小サイズになりますが、それでもクエリのアクティビティに必要な場合は利用可能です。

グローバルセカンダリインデックスからのデータの読み込み

グローバルセカンダリインデックスから項目を取得するには、Query および Scan オペレーションを使用します。GetItem および BatchGetItem オペレーションは、グローバルセカンダリインデックスでは使用できません。

グローバルセカンダリインデックスのクエリ

Query オペレーションを使用して、グローバルセカンダリインデックスの 1 つ以上の項目にアクセスすることができます。クエリでは、使用するベーステーブル名とインデックス名、クエリ結果で返される属性、および適用するクエリ条件を指定する必要があります。DynamoDB は、結果を昇順で返すことも降順で返すこともできます。

順位表アプリケーションのゲームデータをリクエストする Query から返される次のデータについて考えます。

```
{
    "TableName": "GameScores",
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

このクエリでは次のようになっています。

- DynamoDB は、GameTitle パーティションキーを使用して GameTitleIndex にアクセスし、Meteor Blasters のインデックス項目を特定します。このキーを持つすべてのインデックス項目が、すばやく取り出せるように隣り合わせに格納されます。
- このゲーム内で、DynamoDB はインデックスを使用して、このゲームのすべてのユーザーの ID と最高スコアにアクセスします。

- ScanIndexForward パラメータが false に設定されているので、結果は降順で返されます。

グローバルセカンダリインデックスのスキャン

Scan オペレーションを使用して、グローバルセカンダリインデックスからすべてのデータを取得することができます。リクエストにはベーステーブル名とインデックス名を指定する必要があります。Scan では、DynamoDB はインデックスのすべてのデータを読み込み、それをアプリケーションに返します。また、データの一部のみを返し、残りのデータを破棄するようにリクエストすることもできます。これを行うには、FilterExpression オペレーションの Scan パラメータを使用します。詳細については、「」を参照してくださいScan のフィルタ式

テーブルとグローバルセカンダリインデックス間のデータ同期

DynamoDB では、各グローバルセカンダリインデックスをそのベーステーブルと自動的に同期させています。アプリケーションがテーブルに項目を書き込むか、削除すると、そのテーブルのすべてのグローバルセカンダリインデックスは、結果整合性モデルを使用して、非同期で更新されます。アプリケーションがインデックスに直接書き込むことはありません。ただし、DynamoDB でこれらのインデックスがどのように維持されるかを理解することは重要です。

グローバルセカンダリーインデックスは、基本テーブルから読み込み/書き込みキャパシティーモードを継承します。詳細については、「キャパシティモードを切り替える際の考慮事項」を参照してください。

グローバルセカンダリインデックスを作成するときは、1 つ以上のインデックスキー属性およびそれらのデータ型を指定します。つまり、ベーステーブルに項目を書き込むとき、それらの属性のデータ型が、インデックスキースキーマのデータ型に一致する必要があります。GameTitleIndex の場合、インデックス内の GameTitle パーティションキーは、String データ型として定義されています。インデックス内の TopScore ソートキーは、Number の型です。GameScores テーブルに項目を追加し、GameTitle または TopScore に対して別のデータ型を指定する場合、データ型の不一致により DynamoDB によって ValidationException が返されます。

テーブルに項目を入力するか、削除すると、そのテーブルのグローバルセカンダリインデックスは、結果整合性のある方法で更新されます。テーブルデータへの変更は、通常は、瞬時にグローバルセカンダリインデックスに伝達されます。ただし、万が一障害が発生した場合は、長い伝達遅延が発生することがあります。このため、アプリケーションでは、グローバルセカンダリインデックスに関するクエリに対して、最新でない結果が返される状況を予期し、それに対応する必要があります。

テーブルに項目を書き込む場合には、グローバルセカンダリインデックスのソートキーに対して属性を指定する必要はありません。GameTitleIndex を例にとると、TopScore テーブルに新

しい項目を書き込むために、GameScores 属性に値を指定する必要はありません。このケースでは、DynamoDB はこの特定の項目のインデックスにデータを書き込むことはありません。

多数のグローバルセカンダリインデックスがあるテーブルは、インデックス数が少ないテーブルに比べて書き込みアクティビティに多くのコストを要します。詳細については、「グローバルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項」を参照してください。

グローバルセカンダリインデックスがあるテーブルクラス

グローバルセカンダリインデックスは、常にベーステーブルと同じテーブルクラスを使用します。テーブルに新しいグローバルセカンダリインデックスが追加されるたびに、新しいインデックスはベーステーブルと同じテーブルクラスを使用します。テーブルのテーブルクラスが更新されると、関連するすべてのグローバルセカンダリインデックスも更新されます。

グローバルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項

プロビジョニングモードのテーブルでグローバルセカンダリインデックスを作成するときには、そのインデックスに対して予想されるワークロードに応じた読み込み/書き込みキャパシティーユニットを指定する必要があります。グローバルセカンダリインデックスのプロビジョニングされたスループット設定は、そのベーステーブルの設定とは独立しています。グローバルセカンダリインデックスに対する Query オペレーションでは、ベーステーブルではなく、インデックスから読み込みキャパシティーユニットを使用します。テーブルで項目を入力、更新または削除すると、そのテーブルのグローバルセカンダリインデックスも更新されます。これらのインデックス更新は、ベーステーブルからではなく、インデックスから書き込みキャパシティーユニットを消費します。

例えば、グローバルセカンダリインデックスに対して Query を実行し、そのプロビジョニングされた読み込みキャパシティーを超えた場合、リクエストは調整されます。多量の書き込みアクティビティをテーブルに対して実行するときに、そのテーブルのグローバルセカンダリインデックスに十分な書き込み容量がない場合、そのテーブルに対する書き込みアクティビティは調整されます。

> ⚠ Important
>
> スロットリングを避けるため、グローバルセカンダリインデックスのプロビジョニングされた書き込みキャパシティーは、ベーステーブルの書き込みキャパシティーと同じかそれより大きい必要があります。これは、新しい更新ではベーステーブルとグローバルセカンダリインデックスの両方に書き込むためです。

グローバルセカンダリインデックスのプロビジョニングされたスループット設定を表示するには、`DescribeTable` オペレーションを使用します。テーブルのすべてのグローバルセカンダリインデックスに関する詳細情報が返されます。

読み込みキャパシティユニット

グローバルセカンダリインデックスでは、結果整合性のある読み込みをサポートしており、各読み込みで、読み込みキャパシティーユニットの半分を消費します。つまり、1 回のグローバルセカンダリインデックスのクエリでは、1 読み込みキャパシティーユニットあたり最大 2 × 4 KB = 8 KB を取り出すことができます。

グローバルセカンダリインデックスのクエリの場合、DynamoDB はプロビジョニングされた読み込みアクティビティをテーブルに対するクエリと同じ方法で計算します。唯一の違いは、ベーステーブル内の項目のサイズではなくインデックスエントリのサイズに基づいて計算が行われることです。読み込みキャパシティーユニットの数は、返されたすべての項目について射影されたすべての属性のサイズの合計です。結果は、次の 4 KB 境界まで切り上げられます。DynamoDB がプロビジョニングされたスループットの利用率を計算する方法の詳細については、「プロビジョンドキャパシティモード」を参照してください。

`Query` オペレーションによって返される結果の最大サイズは 1 MB です。これには、返されるすべての項目にわたる、すべての属性の名前と値のサイズが含まれます。

例えば、各項目に 2,000 バイトのデータが格納されているグローバルセカンダリインデックスがあるとします。このインデックスに対して `Query` を実行したら、クエリの `KeyConditionExpression` が 8 項目に一致したとします。一致する項目の合計サイズは、2,000 バイト × 8 項目 = 16,000 バイトです。これは、最も近い 4 KB 境界に切り上げられます。グローバルセカンダリインデックスのクエリは結果整合性なので、合計コストは、0.5 × (16 KB / 4 KB)、つまり、2 読み込みキャパシティーユニットです。

書き込みキャパシティユニット

テーブルの項目が追加、更新、または削除されたときに、グローバルセカンダリインデックスがこの影響を受ける場合、グローバルセカンダリインデックスは、そのオペレーションに対してプロビジョニングされた書き込みキャパシティーユニットを使用します。書き込み用にプロビジョニングされたスループットの合計コストは、ベーステーブルに対する書き込みと、グローバルセカンダリインデックスの更新で消費された書き込みキャパシティーユニットの合計になります。テーブルへの書き込みには、グローバルセカンダリインデックスの更新は必要ないので、インデックスから使用される書き込み容量はありません。

テーブルへの書き込みに成功するように、テーブルとそのすべてのグローバルセカンダリインデックスに対するプロビジョニングされたスループット設定は、書き込みに対応できるだけの十分な書き込みキャパシティーを備えている必要があります。十分でない場合、書き込みが調整されます。

グローバルセカンダリインデックスに項目を書き込むコストは、いくつかの要因に左右されます。

- インデックス付き属性が定義されたテーブルに新しい項目を書き込む場合、または既存の項目を更新して未定義のインデックス付き属性を定義する場合には、インデックスへの項目の挿入に 1 回の書き込みオペレーションが必要です。
- テーブルに対する更新によってインデックス付きキー属性の値が（A から B に）変更された場合には、インデックスから既存の項目を削除し、インデックスに新しい項目を挿入するために、2 回の書き込みが必要です。
- インデックス内に既存の項目があって、テーブルに対する書き込みによってインデックス付き属性が削除された場合は、インデックスから古い項目の射影を削除するために、1 回の書き込みが必要です。
- 項目の更新の前後にインデックス内に項目が存在しない場合は、インデックスで追加の書き込みコストは発生しません。
- テーブルに対する更新によってインデックスキースキーマの射影された属性の値のみが変更され、インデックス付きキー属性の値は変更されない場合は、インデックスに射影された属性の値を更新するために、1 回の書き込みが必要です。

これらすべての要因は、インデックス内の各項目のサイズが 1 KB 以下であるという前提で書き込みキャパシティーユニット数を算出します。インデックスエントリがそれよりも大きい場合は、書き込みキャパシティーユニットを追加する必要があります。クエリが返す必要がある属性を特定し、それらの属性だけをインデックスに射影することで、書き込みコストは最小になります。

グローバルセカンダリインデックスのストレージに関する考慮事項

アプリケーションがテーブルに項目を書き込むと、DynamoDB では、正しい属性のサブセットが、それらの属性が現れる必要があるグローバルセカンダリインデックスに自動的にコピーされます。AWS アカウントでは、テーブル内の項目のストレージと、そのベーステーブルのグローバルセカンダリインデックスにある属性のストレージに対して課金されます。

インデックス項目が使用するスペースの量は、次の量の合計になります。

- ベーステーブルのプライマリキー (パーティションキーとソートキー) のサイズのバイト数
- インデックスキー属性のサイズのバイト数

- 射影された属性 (存在する場合) のサイズのバイト数
- インデックス項目あたり 100 バイトのオーバーヘッド

グローバルセカンダリインデックスのストレージ要件の見積もりは、インデックス内の 1 項目の平均サイズの見積もり値に、ベーステーブル内のグローバルセカンダリインデックスキー属性を持つ項目の数を掛けて算出します。

特定の属性が定義されていない項目がテーブルに含まれていて、その属性がインデックスパーティションキーまたはソートキーとして定義されている場合、DynamoDB はその項目のデータをインデックスに書き込みません。

グローバルセカンダリインデックスの管理

このセクションでは、Amazon DynamoDB でグローバルセカンダリインデックスを作成、変更、削除する方法について説明します。

トピック

グローバルセカンダリインデックスを持つテーブルの作成

1 つ以上のグローバルセカンダリインデックスを持つテーブルを作成するには、`CreateTable` オペレーションと `GlobalSecondaryIndexes` パラメータを使用します。最大限のクエリの柔軟性を得るために、テーブルごとに最大 20 個のグローバルセカンダリインデックス (デフォルトのクォータ) を作成できます。

インデックスパーティションキーとして機能する属性を 1 つ指定する必要があります。必要に応じて、インデックスソートキーに別の属性を指定できます。これらのキー属性のいずれも、テーブルのキー属性と同じである必要はありません。例えば、GameScores テーブル (「DynamoDB のグローバルセカンダリインデックスの使用」参照) では、`TopScore` も `TopScoreDateTime` もキー属性ではありません。パーティションキーとして `TopScore`、ソートキーとして `TopScoreDateTime` を

使用して、グローバルセカンダリインデックスを作成できます。このようなインデックスを使用して、ハイスコアとゲームのプレイ時刻との間に相関があるかどうかを判定できる可能性があります。

各インデックスキー属性は、String、Number、または Binary タイプのスカラーである必要があります。(ドキュメントまたはセットは指定できません)。グローバルセカンダリインデックスには、任意のデータ型の属性を射影できます。これには、スカラー、ドキュメント、およびセットが含まれます。データ型の詳細なリストについては、「データ型」を参照してください。

プロビジョニング済みモードを使用する場合は、インデックスに ReadCapacityUnits および WriteCapacityUnits で構成される ProvisionedThroughput 設定をする必要があります。これらのプロビジョニングされたスループット設定は、テーブルの設定とは別ですが、同様の動作をします。詳細については、「」を参照してくださいグローバルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項

グローバルセカンダリーインデックスは、基本テーブルから読み込み/書き込みキャパシティーモードを継承します。詳細については、「キャパシティモードを切り替える際の考慮事項」を参照してください。

Note

バックフィル操作と進行中の書き込み操作で、グローバルセカンダリインデックス内の書き込みスループットが共有されます。新しい GSI を作成する際、選択したパーティションキーが、新しいインデックスのパーティションキー値全体でデータやトラフィックの分散が不均一だったり、狭くなっているかを確認することが重要になります。この場合、バックフィル操作と書き込み操作が同時に発生し、ベーステーブルへの書き込みをスロットリングしている可能性があります。このサービスでは、このシナリオの可能性を最小限に抑えるための対策を講じていますが、インデックスパーティションキー、選択した射影、またはインデックスプライマリキーのスパース性について、カスタマーデータのインサイトを見える形で取得していません。
新しいグローバルセカンダリインデックスで、パーティションキー値全体でデータやトラフィックの分散が狭められたり、歪められたりしている可能性がある場合は、運用上重要なテーブルに新しいインデックスを追加する前に、次の点を考慮してください。

- アプリケーションのトラフィック量が最も少ない時間帯にインデックスを追加するのが、最も安全な方法と言えます。
- ベーステーブルとインデックスで CloudWatch Contributor Insights 有効にすることを検討してください。これにより、トラフィックの分散に関する貴重なインサイトが得られます。

- プロビジョニングされたキャパシティーモードのベーステーブルとインデックスでは、新しいインデックスのプロビジョニングされた書き込みキャパシティーをベーステーブルの少なくとも 2 倍に設定します。プロセス全体を通して、CloudWatch メトリクス `WriteThrottleEvents`、`ThrottledRequests`、`OnlineIndexPercentageProgress`、`Onli`
 および `OnlineIndexThrottleEvents` を表示します。必要に応じて、プロビジョニングされた書き込みキャパシティーを調整し、進行中の操作に対してスロットリングによる大きな影響を与えることなく、適切な時間内にバックフィルを完了します。
- 書き込みスロットリングによる運用上の影響が発生した場合は、インデックスの作成をキャンセルする準備をしてください。新しい GSI のプロビジョニングされた書き込みキャパシティーを増やしてもその問題は解決しません。

テーブルのグローバルセカンダリインデックスの説明

テーブルのすべてのグローバルセカンダリインデックスのステータスを表示するには、`DescribeTable` オペレーションを使用します。レスポンスの `GlobalSecondaryIndexes` 部分は、テーブル上のすべてのインデックスと、それぞれの現在のステータス (`IndexStatus`) を示しています。

グローバルセカンダリインデックスの `IndexStatus` は、次のいずれかになります。

- `CREATING` – インデックスは現在作成されていますが、まだ使用することはできません。
- `ACTIVE` – インデックスは使用可能になり、アプリケーションはインデックスに対して `Query` オペレーションを実行できます。
- `UPDATING` – インデックスのプロビジョニングされたスループット設定の変更中です。
- `DELETING` – インデックスは現在削除されているため、使用できなくなっています。

DynamoDB がグローバルセカンダリインデックスの作成を完了すると、インデックスのステータスが `CREATING` から `ACTIVE` に変わります。

グローバルセカンダリインデックスの既存テーブルへの追加

既存のテーブルにグローバルセカンダリインデックスを追加するには、`UpdateTable` オペレーションと `GlobalSecondaryIndexUpdates` パラメータを使用します。以下の情報が必要です。

- インデックス名。名前は、テーブル内のすべてのインデックスにおいて一意である必要があります。

- インデックスのキースキーマ。インデックスパーティションキーには 1 つの属性を指定する必要があります。必要に応じて、インデックスソートキーに別の属性を指定できます。これらのキー属性のいずれも、テーブルのキー属性と同じである必要はありません。各スキーマ属性のデータ型は、String、Number、または Binary のスカラーである必要があります。
- テーブルからインデックスに射影される属性は以下の通りです。
  - KEYS_ONLY – インデックス内の各項目は、テーブルパーティションキーとソートキーの値、およびインデックスキーの値のみで構成されます。
  - INCLUDE – KEYS_ONLY の属性に加えて、セカンダリインデックスにその他の非キー属性が含まれるように指定できます。
  - ALL – インデックスには、ソーステーブルのすべての属性が含まれます。
- インデックスのプロビジョニングされたスループット設定で、ReadCapacityUnits および WriteCapacityUnits で構成されます。これは、テーブルのプロビジョニングされたスループット設定とは異なります。

UpdateTable オペレーションについて作成できるグローバルセカンダリインデックスは 1 つだけです。

インデックス作成のフェーズ

新しいグローバルセカンダリインデックスを既存のテーブルに追加すると、インデックスの構築中もテーブルが使用可能になります。ただし、新しいインデックスは、そのステータスが CREATING から ACTIVE に変わるまでクエリオペレーションに使用できません。

> **Note**
>
> グローバルセカンダリインデックスの作成では、Application Auto Scaling を使用しません。MIN Application Auto Scaling の容量を増やしても、グローバルセカンダリインデックスの作成時間は短縮されません。

その裏では、DynamoDB は 2 つのフェーズでインデックスを構築しています。

リソース割り当て

DynamoDB は、インデックスの構築に必要なコンピューティングリソースとストレージリソースを割り当てます。

リソース割り当てフェーズでは、IndexStatus 属性は CREATING、Backfilling 属性は false です。DescribeTable オペレーションを使用して、テーブルとそのセカンダリインデックスすべてのステータスを取得します。

インデックスがリソース割り当てフェーズにある間は、インデックスやその親テーブルを削除することはできません。インデックスまたはテーブルのプロビジョニングされたスループットを変更することもできません。テーブル上の他のインデックスを追加したり削除したりすることはできません。ただし、これらの他のインデックスのプロビジョニングされたスループットは変更できます。

バックフィル

テーブルの各項目について、DynamoDB は、その射影 (KEYS_ONLY、INCLUDE、または ALL) に基づいて、インデックスに書き込む属性のセットを決定します。次に、これらの属性をインデックスに書き込みます。バックフィルフェーズでは、DynamoDB はテーブルで追加、削除、または更新される項目を追跡します。これらの項目の属性も、必要に応じてインデックスで追加、削除、または更新されます。

バックフィルフェーズでは、IndexStatus 属性は CREATING に設定されていて、Backfilling 属性は true です。DescribeTable オペレーションを使用して、テーブルとそのセカンダリインデックスすべてのステータスを取得します。

インデックスがバックフィルされている間は、親テーブルを削除することはできません。ただし、インデックスを削除したり、テーブルとそのグローバルセカンダリインデックスのプロビジョニングされたスループットを変更したりすることはできます。

> ⓘ Note
>
> バックフィルフェーズでは、違反する項目の書き込みの一部が成功し、他の項目が拒否されることがあります。バックフィル後、新しいインデックスのキースキーマに違反する項目への書き込みはすべて拒否されます。バックフィルフェーズの終了後に Violation Detector ツールを実行して、発生した可能性のあるキー違反を検出して解決することをお勧めします。詳細については、「」を参照してくださいインデックスキー違反の検出と修正

リソース割り当てフェーズとバックフィルフェーズの進行中、インデックスは CREATING 状態になっています。この間、DynamoDB はテーブルに対して読み込みオペレーションを実行します。グローバルセカンダリインデックスを設定するためのベーステーブルからの読み込みオペレーションに

対しては課金されません。ただし、新しく作成されたグローバルセカンダリインデックスを設定するための書き込みオペレーションに対しては課金されます。

インデックス構築が完了すると、ステータスは `ACTIVE` に変わります。インデックスが `ACTIVE` になるまで、`Query` や `Scan` はできません。

> Note
>
> 場合によっては、DynamoDB は、インデックスキーの違反のため、テーブルからインデックスにデータを書き込めない場合があります。これは、以下の場合に発生します。
>
> - 属性値のデータ型が、インデックスキーのスキーマデータ型のデータ型と一致しません。
> - 属性のサイズが、インデックスキー属性の最大長を超えています。
> - インデックスキー属性には、空の文字列または空のバイナリ属性値があります。
>
> インデックスキー違反があっても、グローバルセカンダリインデックスの作成はできます。ただし、インデックスが `ACTIVE` になると、違反しているキーはインデックスに存在しなくなります。
> DynamoDB には、これらの問題を検出して解決するためのスタンドアロンツールが用意されています。詳細については、「インデックスキー違反の検出と修正」を参照してください。

大きなテーブルへのグローバルセカンダリインデックスの追加

グローバルセカンダリインデックスの構築に必要な時間は、次のようないくつかの要因によって異なります。

- テーブルのサイズ
- インデックスに含めることができるテーブル内の項目の数
- インデックスに射影される属性の数。
- インデックスのプロビジョニングされた書き込み容量
- インデックス構築中のメインテーブルへの書き込みアクティビティ

非常に大きなテーブルにグローバルセカンダリインデックスを追加する場合、作成プロセスが完了するまで時間がかかることがあります。進行状況をモニタリングし、インデックスに十分な書き込み容量があるかどうかを判断するには、次の Amazon CloudWatch メトリクスを参照してください。

- `OnlineIndexPercentageProgress`
- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

> Note
>
> DynamoDB に関する CloudWatch メトリクスの詳細については、「DynamoDB のメトリクス」を参照してください。

インデックスのプロビジョニングされた書き込みスループット設定が低すぎると、インデックスの構築が完了するまでに時間がかかります。プロビジョニングされた書き込み容量を一時的に増やすことにより、新しいグローバルセカンダリインデックスの構築にかかる時間を短縮することができます。

> Note
>
> 原則として、インデックスのプロビジョニングされた書き込み容量をテーブルの書き込み容量の 1.5 倍に設定することをお勧めします。これは、多くのユースケースに適した設定です。ただし、実際に必要な容量がそれよりも高かったり、低かったりする可能性があります。

インデックスがバックフィルされている間、DynamoDB は内部システム容量を使用してテーブルから読み込みます。これは、インデックスの作成による影響を最小限に抑え、テーブルの読み込みキャパシティーが不足しないようにするためです。

ただし、入ってくる書き込みアクティビティのボリュームが、インデックスのプロビジョニングされた書き込み容量を超える可能性があります。この場合、インデックスへの書き込みアクティビティが抑制されるため、インデックスの作成に時間がかかります。これはボトルネックのシナリオです。インデックスの構築中に、インデックスの Amazon CloudWatch メトリクスをモニタリングして、使用された書き込み容量がプロビジョニングされた容量を超えているかどうかを判断することをお勧めします。ボトルネックのシナリオでは、バックフィルフェーズ中の書き込み抑制を回避するために、インデックスにプロビジョニングされた書き込み容量を増やすことが必要です。

インデックスの作成後、アプリケーションの通常の使用状況を反映するように、プロビジョニングされた書き込み容量を設定することが必要です。

グローバルセカンダリインデックスの削除

グローバルセカンダリインデックスが不要になった場合には、UpdateTable オペレーションを使用して削除することができます。

グローバルセカンダリインデックスは、1 回の UpdateTable オペレーションで 1 つだけ削除できます。

グローバルセカンダリインデックスが削除されている間、親テーブルの読み込みまたは書き込みアクティビティに影響はありません。削除の進行中でも、他のインデックスでプロビジョニングされたスループットを変更できます。

> **Note**
>
> - DeleteTable アクションを使用してテーブルを削除すると、そのテーブルのすべてのグローバルセカンダリインデックスも削除されます。
> - アカウントではグローバルセカンダリインデックスの削除操作に対して課金されません。

作成時のグローバルセカンダリインデックスの変更

インデックスが作成されている間は、DescribeTable オペレーションを使用して、どのフェーズにあるかを判断します。インデックスの説明に含まれているブール属性 Backfilling は、DynamoDB がテーブルから項目を含むインデックスを現在ロードしているかどうかを示します。Backfilling が true の場合、リソース割り当てフェーズは完了していて、インデックスがバックフィルされています。

バックフィルが進行している間は、インデックスのプロビジョニングされたスループットパラメータを更新できます。インデックスの構築を高速化するために、インデックスの作成中にインデックスの書き込み容量を増やし、後で減らすこともできます。インデックスのプロビジョニングされたスループット設定を変更するには、UpdateTable オペレーションを使用します。インデックスのステータスが UPDATING に変わります。インデックスが使用可能になるまで、Backfilling は true です。

バックフィルフェーズでは、作成中のインデックスを削除できます。このフェーズでは、テーブルの他のインデックスを追加または削除することはできません。

> Note
>
> CreateTable オペレーションの一部として作成されたインデックスの場合、Backfilling 属性は DescribeTable 出力に現れません。詳細については、「インデックス作成のフェーズ」を参照してください。

インデックスキー違反の検出と修正

グローバルセカンダリインデックスの作成のバックフィルフェーズでは、Amazon DynamoDB はテーブルの各項目を調べて、インデックスに含める適格性があるかどうかを判断します。一部の項目は、インデックスキー違反の原因となるため、適格でない場合があります。このような場合、項目はテーブルに残りますが、インデックスにはその項目に対応するエントリがなくなります。

インデックスキー違反は次の場合に発生します。

- 属性値とインデックスキーのスキーマのデータ型が一致しない場合。たとえば、GameScores テーブルの項目の 1 つが、String 型の TopScore 値を持っていたとします。Number 型の TopScore のパーティションキーを持つグローバルセカンダリインデックスを追加した場合、テーブルの項目はインデックスキーに違反します。
- テーブルの属性値が、インデックスキー属性の最大長を超えている場合。パーティションキーの最大長は 2048 バイトで、ソートキーの最大長は 1024 バイトです。テーブル内の対応する属性値のいずれかがこれらの制限を超えると、テーブルの項目はインデックスキーに違反します。

> Note
>
> インデックスキーとして使用される属性に String 属性または Binary 属性値が設定されている場合、属性値の長さは 0 より大きくする必要があります。そうしないと、テーブルの項目がインデックスキーに違反します。
> 現時点では、このツールはこのインデックスキー違反にフラグを設定しません。

インデックスキー違反が発生した場合でも、バックフィルフェーズは中断することなく続行されます。ただし、違反している項目は、インデックスに含まれません。バックフィルフェーズ完了後は、新しいインデックスのキースキーマに違反する項目へのすべての書き込みが拒否されます。

インデックスキーに違反するテーブル内の属性値を識別して修正するには、Violation Detector ツールを使用します。Violation Detector を実行するには、スキャンするテーブルの名前、グローバルセ

カンダリインデックスパーティションキーとソートキーの名前とデータ型、インデックスキー違反が見つかった場合に実行するアクションを指定する設定ファイルを作成します。Violation Detector は、次の 2 つのモードのいずれかで実行できます。

- 検出モード – インデックスキー違反を検出します。検出モードを使用して、グローバルセカンダリインデックスでキー違反の原因となるテーブル内の項目を報告します。(必要に応じて、違反しているテーブル項目が見つかるとすぐに削除するように要求できます)。検出モードからの出力はファイルに書き込まれるので、これを使用してさらに分析できます。
- 修正モード – インデックスキー違反を修正します。修正モードでは、Violation Detector は検出モードからの出力ファイルと同じ形式の入力ファイルを読み込みます。修正モードでは、入力ファイルからレコードを読み込み、各レコードについて、テーブル内の対応する項目を削除または更新します。(項目の更新を選択した場合は、入力ファイルを編集して、これらの更新に適切な値を設定する必要があります)。

Violation Detector のダウンロードおよび実行

Violation Detector は、実行可能な Java Archive (`.jar` ファイル) で、Windows、macOS、または Linux コンピュータ上で実行されます。Violation Detector には Java 1.7 (以降) と Apache Maven が必要です。

- GitHub から Violation Detector をダウンロードします

`README.md` ファイルの指示に従って、Maven を使用して Violation Detector をダウンロードしてインストールします。

Violation Detector を起動するには、`ViolationDetector.java` を構築したディレクトリで、次のコマンドを入力します。

```
java -jar ViolationDetector.jar [options]
```

Violation Detector ツールのコマンドラインでは、以下のオプションが使用できます。

- `-h | --help` – Violation Detector の使用方法の概要とオプションを出力します。
- `-p | --configFilePath` `value` – Violation Detector 設定ファイルの完全修飾名。詳細については、「」を参照してくださいViolation Detector 設定ファイル
- `-t | --detect` `value` – テーブル内のインデックスキー違反を検出し、Violation Detector の出力ファイルに書き込みます。このパラメータの値が `keep` に設定されている場合、キー違反のある

項目は変更されません。値が delete に設定されている場合、キー違反のある項目はテーブルから削除されます。

- `-c | --correct value` – 入力ファイルからインデックスキー違反を読み込み、テーブル内の項目に対して修正アクションを実行します。このパラメータの値が update に設定されている場合、キー違反のある項目は、違反していない新しい値で更新されます。値が delete に設定されている場合、キー違反のある項目はテーブルから削除されます。

Violation Detector 設定ファイル

実行時に、Violation Detector ツールには設定ファイルが必要です。このファイルのパラメータによって、Violation Detector がアクセスできる DynamoDB リソースと、プロビジョニングされたスループットを使用できる量が決まります。以下の表は、これらのパラメータの説明です。

パラメータ名	説明	必須?

`awsCredentialsFile`	AWS 認証情報を含むファイルの完全修飾名。認証情報ファイルの形式は、次のようになります。
	`accessKey = access_key_id_goes_here`
	`secretKey =  secret_key_goes_here`
		はい

`dynamoDBRegion`	テーブルが存在する AWS リージョン。例: `us-west-2` 。	はい
`tableName`	スキャンされる DynamoDB テーブルの名前。	はい
`gsiHashKeyName`	インデックスパーティションキーの名前。	はい
`gsiHashKeyType`	インデックスパーティションキーのデータ型 –	はい

パラメータ名	説明	必須?
	String、Number、または Binary: S | N | B	
gsiRangeKeyName	インデックスソートキーの名前。インデックスにシンプルなプライマリキー (パーティションキー) のみがある場合は、このパラメータを指定しないでください。	いいえ
gsiRangeKeyType	インデックスソートキーのデータ型 – String、Number、または Binary: S | N | B インデックスにシンプルなプライマリキー (パーティションキー) のみがある場合は、このパラメータを指定しないでください。	いいえ
recordDetails	インデックスキー違反の詳細を出力ファイルに書き込むかどうか。設定が true の場合 (デフォルト)、違反項目に関する完全な情報が報告されます。設定が false の場合、違反の数のみが報告されます。	いいえ

パラメータ名	説明	必須?
recordGsiValueInViolationRecord	違反しているインデックスキーの値を出力ファイルに書き込むかどうか。設定が true の場合 (デフォルト)、キー値が報告されます。設定が false の場合、キー値は報告されません。	いいえ

detectionOutputPath	Violation Detector の出力ファイルのフルパス。このパラメータは、ローカルディレクトリまたはAmazon Simple Storage Service (Amazon S3) への書き込みをサポートしています。次に例を示します。
	detectionOutputPath = //*local/path/filename.csv*
	detectionOutputPath = s3://*bucket*/*filename.csv*
	出力ファイル内の情報は、CSV (Comma-Separated Values) 形式で表示されます。detectionOutputPath を設定しない場合、出力ファイル名は violation_detection.csv になり、現在の作業ディレクトリに書き込まれます。
		いいえ

パラメータ名	説明	必須?
`numOfSegments`	Violation Detector がテーブルをスキャンするときに使用される並列スキャンセグメントの数。デフォルト値は 1 です。これは、テーブルがシーケンシャルにスキャンされることを意味します。値が 2 以上の場合、Violation Detector はテーブルをその数の論理セグメントに分割し、同じ数のスキャンスレッドにします。 `numOfSegments` の設定は最大 4,096 です。 大きなテーブルの場合、並列スキャンはシーケンシャルスキャンよりも一般的に高速です。さらに、テーブルが複数のパーティションにまたがるほど大きい場合、並列スキャンによって読み込みアクティビティが複数のパーティションに均等に分散されます。DynamoDB の並列スキャンの詳細については、「並列スキャン」を参照してください。	いいえ

パラメータ名	説明	必須?
`numOfViolations`	出力ファイルに書き込むインデックスキー違反の上限値。設定が `-1` の場合 (デフォルト)、テーブル全体がスキャンされます。正の整数に設定すると、Violation Detector は設定した数の違反を検出した後に停止します。	いいえ
`numOfRecords`	スキャンされるテーブル内の項目数。設定が -1 の場合 (デフォルト)、テーブル全体がスキャンされます。正の整数に設定すると、Violation Detector はテーブル内の設定した数の項目をスキャンした後に停止します。	いいえ
`readWriteIOPSPercent`	テーブルスキャン中に使用される、プロビジョニングされた読み込みキャパシティーユニットの割合を調整します。有効な値の範囲は `1`～`100` です。デフォルト値 (25) は、Violation Detector がテーブルのプロビジョニングされた読み込みスループットの 25% 以下を使用することを意味します。	いいえ

パラメータ名	説明	必須?
`correctionInputPath`	Violation Detector 修正入力ファイルのフルパス。Violation Detector を修正モードで実行すると、このファイルの内容を使用して、グローバルセカンダリインデックスに違反するテーブル内のデータ項目を変更または削除します。 `correctionInputPath` ファイルの形式は、`detectionOutputPath` ファイルの形式と同じです。これにより、検出モードからの出力を修正モードの入力として処理できます。	いいえ

パラメータ名	説明	必須?

`correctionOutputPath`	Violation Detector 修正出力ファイルのフルパス。このファイルは、更新エラーがある場合にのみ作成されます。
	このパラメータは、ローカルディレクトリまたは Amazon S3 への書き込みをサポートします。次に例を示します。
	`correctionOutputPath = //local/path/filename.csv`
	`correctionOutputPath = s3://bucket/filename.csv`
	出力ファイルの情報は CSV 形式で表示されます。`correctionOutputPath` を設定しない場合、出力ファイル名は `violation_update_errors.csv` になり、現在の作業ディレクトリに書き込まれます。
		[いいえ]

検出

インデックスキー違反を検出するには、Violation Detector を `--detect` コマンドラインオプションで使用します。このオプションがどのように機能するかを示すには、DynamoDB でのコード例用のテーブルの作成とデータのロード の `ProductCatalog` テーブルを検討してください。以下は、テーブルの項目の一覧です。プライマリキー (`Id`) と `Price` 属性のみを表示しています。

ID (プライマリキー)	価格
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Price の値はすべて、Number 型です。ただし、DynamoDB はスキーマレスであるため、数値以外の Price の項目を追加することができます。たとえば、ProductCatalog テーブルに別の項目を追加するとします。

ID (プライマリキー)	価格
999	"Hello"

これで、テーブルには合計 9 つの項目が追加されました。

ここで、新しいグローバルセカンダリインデックスをテーブルに追加します: PriceIndex。このインデックスのプライマリキーは、パーティションキー Price で、Number 型です。インデックスが構築されると、8 つの項目が含まれますが、ProductCatalog テーブルには 9 つの項目があります。この不一致の理由は、値 "Hello" は String 型ですが、PriceIndex は Number 型のプライマリキーを持っているからです。String 値がグローバルセカンダリインデックスキーに違反するため、インデックスには存在しません。

この状況で Violation Detector を使用するには、まず次のような設定ファイルを作成します。

```
# Properties file for violation detection tool configuration.
# Parameters that are not specified will use default values.

awsCredentialsFile = /home/alice/credentials.txt
dynamoDBRegion = us-west-2
tableName = ProductCatalog
gsiHashKeyName = Price
gsiHashKeyType = N
recordDetails = true
recordGsiValueInViolationRecord = true
detectionOutputPath = ./gsi_violation_check.csv
correctionInputPath = ./gsi_violation_check.csv
numOfSegments = 1
readWriteIOPSPercent = 40
```

続いて、次の例のように、Violation Detector を実行します。

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --detect keep

Violation detection started: sequential scan, Table name: ProductCatalog, GSI name:
 PriceIndex
Progress: Items scanned in total: 9,    Items scanned by this thread: 9,    Violations
 found by this thread: 1, Violations deleted by this thread: 0
Violation detection finished: Records scanned: 9, Violations found: 1, Violations
 deleted: 0, see results at: ./gsi_violation_check.csv
```

`recordDetails` 設定パラメータが `true` に設定されている場合、Violation Detector は、次の例のように、各違反の詳細を出力ファイルに書き込みます。

```
Table Hash Key,GSI Hash Key Value,GSI Hash Key Violation Type,GSI Hash Key Violation
 Description,GSI Hash Key Update Value(FOR USER),Delete Blank Attributes When Updating?
(Y/N)

999,"{""S"":""Hello""}",Type Violation,Expected: N Found: S,,
```

出力ファイルは CSV 形式です。ファイルの最初の行はヘッダーで、2 行目以降はインデックスキーに違反する項目ごとに 1 つのレコードが続きます。これらの違反レコードのフィールドは次のとおりです。

- テーブルハッシュキー – テーブル内の項目のパーティションキー値です。
- テーブル範囲キー – テーブル内の項目のソートキー値です。

- GSI ハッシュキー値 – グローバルセカンダリインデックスのパーティションキー値です。
- GSI ハッシュキー違反タイプ – `Type Violation` または `Size Violation`。
- GSI ハッシュキー違反の説明 – 違反の原因です。
- GSI ハッシュキー更新値 (ユーザー用) – 修正モードでは、ユーザーが指定した属性の新しい値。
- GSI 範囲キー値 – グローバルセカンダリインデックスのソートキー値です。
- GSI 範囲キー違反タイプ – `Type Violation` または `Size Violation`。
- GSI 範囲キー違反の説明 – 違反の原因です。
- GSI 範囲キーの更新値 (ユーザー用) – 修正モードでは、ユーザーが指定した属性の新しい値。
- 更新時に空白の属性を削除 (Y/N) – 修正モードでは、テーブル内の違反項目を削除する (Y) か、保持する (N) かを決定します。ただし、次のフィールドのいずれかが空白の場合のみ実行されます。
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  これらのフィールドのいずれかが空白でない場合、`Delete Blank Attribute When Updating(Y/N)` は何も実行しません。

> **Note**
>
> 出力形式は、設定ファイルとコマンドラインオプションによって異なることがあります。例えば、テーブルに単純なプライマリキー (ソートキーなし) がある場合、出力にはソートキーフィールドは表示されません。
> ファイル内の違反レコードがソート順でない可能性があります。

修正

インデックスキーの違反を修正するには、Violation Detector を `--correct` コマンドラインオプションで使用します。修正モードでは、Violation Detector は `correctionInputPath` パラメータで指定された入力ファイルを読み込みます。このファイルの形式は `detectionOutputPath` ファイルの形式と同じなので、検出からの出力を修正のための入力として使用できます。

Violation Detector には、インデックスキー違反を修正する 2 種類の方法があります。

- 違反の削除 – 属性値に違反しているテーブル項目を削除します。
- 違反の更新 – テーブル項目を更新し、違反している属性を違反しない値に置き換えます。

いずれの場合も、検出モードからの出力ファイルを修正モードの入力として使用できます。

ProductCatalog の例に進みます。違反している項目をテーブルから削除するとします。これを行うには、次のコマンドラインを使用します。

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

この時点で、違反している項目を削除するかどうかを確認するメッセージが表示されます。

```
Are you sure to delete all violations on the table?y/n
y
Confirmed, will delete violations on the table...
Violation correction from file started: Reading records from file: ./
gsi_violation_check.csv, will delete these records from table.
Violation correction from file finished: Violations delete: 1, Violations Update: 0
```

ProductCatalog と PriceIndex の項目の数が同じになりました。

グローバルセカンダリインデックスの操作: Java

AWS SDK for Java ドキュメント API を使用して、1 つ以上のグローバルセカンダリインデックスを持つ Amazon DynamoDB テーブルを作成し、テーブルのインデックスを記述し、インデックスを使用してクエリを実行できます。

以下に、テーブルオペレーションの一般的な手順を示します。

1. DynamoDB クラスのインスタンスを作成します。
2. 対応するリクエストオブジェクトを作成して、オペレーションについて必要なパラメータとオプションパラメータを入力します。
3. 前のステップで作成したクライアントが提供する適切なメソッドを呼び出します。

トピック

- グローバルセカンダリインデックスを持つテーブルを作成します
- グローバルセカンダリインデックスを持つテーブルの説明
- グローバルセカンダリインデックスのクエリ
- 例: AWS SDK for Java ドキュメント API を使用したグローバルセカンダリインデックス

グローバルセカンダリインデックスを持つテーブルを作成します

グローバルセカンダリインデックスは、テーブルの作成と同時に作成できます。これを行うには、CreateTable を使用し、1 つ以上のグローバルセカンダリインデックスの仕様を指定します。次の Java コード例では、気象データに関する情報を保持するテーブルを作成しています。パーティションキーは Location で、ソートキーは Date です。グローバルセカンダリインデックス PrecipIndex は、さまざまな場所の降水データへの高速アクセスを可能にします。

DynamoDB ドキュメント API を使用して、グローバルセカンダリインデックスを持つテーブルを作成する手順を次に示します。

1. DynamoDB クラスのインスタンスを作成します。
2. リクエスト情報を指定する CreateTableRequest クラスのインスタンスを作成します。

   テーブル名、プライマリキー、およびプロビジョニングされたスループット値を指定する必要があります。グローバルセカンダリインデックスの場合は、インデックス名、プロビジョニングされたスループット設定、インデックスソートキーの属性定義、インデックスのキースキーマ、および属性射影を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、createTable メソッドを呼び出します。

以下の Java コード例は、前述のステップの例です。このコードは、グローバルセカンダリインデックス (PrecipIndex) を持つテーブル (WeatherData) を作成します。インデックスパーティションキーは Date で、ソートキーは Precipitation です。すべてのテーブル属性がインデックスに射影されます。ユーザーは、このインデックスに対するクエリを実行して、特定の日付の気象データを取得できます。必要に応じて、降水量によってデータを並べ替えることもできます。

Precipitation はテーブルのキー属性ではないため、必須ではありません。ただし、Precipitation のない WeatherData 項目は PrecipIndex に表示されません。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

// Attribute definitions
ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Location")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Date")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Precipitation")
    .withAttributeType("N"));

// Table key schema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Location")
    .withKeyType(KeyType.HASH));  //Partition key
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.RANGE));  //Sort key

// PrecipIndex
GlobalSecondaryIndex precipIndex = new GlobalSecondaryIndex()
    .withIndexName("PrecipIndex")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 10)
        .withWriteCapacityUnits((long) 1))
        .withProjection(new Projection().withProjectionType(ProjectionType.ALL));

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();

indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.HASH));  //Partition key
indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Precipitation")
    .withKeyType(KeyType.RANGE));  //Sort key

precipIndex.setKeySchema(indexKeySchema);

CreateTableRequest createTableRequest = new CreateTableRequest()
    .withTableName("WeatherData")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 5)
        .withWriteCapacityUnits((long) 1))
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(tableKeySchema)
    .withGlobalSecondaryIndexes(precipIndex);
```

```

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

DynamoDB がテーブルを作成し、テーブルのステータスを ACTIVE に設定するまで待機する必要があります。その後、テーブルへのデータ項目の入力を開始できます。

グローバルセカンダリインデックスを持つテーブルの説明

テーブルでグローバルセカンダリインデックスに関する情報を取得するには、DescribeTable を使用します。インデックスごとに、名前、キースキーマ、および射影された属性にアクセスできます。

テーブルのグローバルセカンダリインデックス情報にアクセスする手順を次に示します。

1. DynamoDB クラスのインスタンスを作成します。
2. 操作対象のインデックスを表すために、Table クラスのインスタンスを作成します。
3. Table オブジェクトの describe メソッドを呼び出します。

以下の Java コード例は、前述のステップの例です。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
TableDescription tableDesc = table.describe();


Iterator<GlobalSecondaryIndexDescription> gsiIter =
 tableDesc.getGlobalSecondaryIndexes().iterator();
while (gsiIter.hasNext()) {
    GlobalSecondaryIndexDescription gsiDesc = gsiIter.next();
    System.out.println("Info for index "
         + gsiDesc.getIndexName() + ":");

    Iterator<KeySchemaElement> kseIter = gsiDesc.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = gsiDesc.getProjection();
    System.out.println("\tThe projection type is: "
        + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: "
            + projection.getNonKeyAttributes());
    }
}
```

グローバルセカンダリインデックスのクエリ

テーブルに Query を実行するのとほぼ同じ方法で、グローバルセカンダリインデックスに対する Query を使用することができます。インデックス名、インデックスパーティションキーとソートキー (存在する場合) のクエリ条件、および返す属性を指定する必要があります。この例では、インデックスは PrecipIndex で、パーティションキーが Date で、ソートキーが Precipitation です。このインデックスクエリは、降水量がゼロより大きい特定の日付のすべての気象データを返します。

AWS SDK for Java ドキュメント API を使用してグローバルセカンダリインデックスをクエリする手順を次に示します。

1. DynamoDB クラスのインスタンスを作成します。
2. 操作対象のインデックスを表すために、Table クラスのインスタンスを作成します。
3. クエリするインデックスの Index クラスのインスタンスを作成します。
4. Index オブジェクトの query メソッドを呼び出します。

属性名 Date は DynamoDB の予約語です。したがって、式の属性名を KeyConditionExpression のプレースホルダーとして使用する必要があります。

以下の Java コード例は、前述のステップの例です。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
Index index = table.getIndex("PrecipIndex");

QuerySpec spec = new QuerySpec()
```

```
    .withKeyConditionExpression("#d = :v_date and Precipitation = :v_precip")
    .withNameMap(new NameMap()
        .with("#d", "Date"))
    .withValueMap(new ValueMap()
        .withString(":v_date","2013-08-10")
        .withNumber(":v_precip",0));

ItemCollection<QueryOutcome> items = index.query(spec);
Iterator<Item> iter = items.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next().toJSONPretty());
}
```

例: AWS SDK for Java ドキュメント API を使用したグローバルセカンダリインデックス

次の Java コード例は、グローバルセカンダリインデックスの操作方法を示しています。この例では、`Issues` というテーブルを作成しています。これは、ソフトウェア開発のためのシンプルなバグ追跡システムで使用できる可能性があります。パーティションキーは `IssueId` で、ソートキーは `Title` です。このテーブルには、次の 3 つのグローバルセカンダリインデックスがあります。

- `CreateDateIndex` – パーティションキーは `CreateDate` で、ソートキーは `IssueId` です。テーブルキーに加えて、属性 `Description` および `Status` はインデックスに射影されます。
- `TitleIndex` – パーティションキーは `Title` で、ソートキーは `IssueId` です。テーブルキー以外の属性はインデックスに射影されません。
- `DueDateIndex` – パーティションキーは `DueDate` で、ソートキーはありません。すべてのテーブル属性がインデックスに射影されます。

`Issues` テーブルが作成されると、プログラムはソフトウェアのバグレポートを表すデータをテーブルにロードします。次に、グローバルセカンダリインデックスを使用してデータのクエリを実行します。最後に、プログラムは `Issues` テーブルを削除します。

以下の例をテストするための詳細な手順については、「Java コードの例」を参照してください。

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.GlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class DocumentAPIGlobalSecondaryIndexExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    public static String tableName = "Issues";

    public static void main(String[] args) throws Exception {

        createTable();
        loadData();

        queryIndex("CreateDateIndex");
        queryIndex("TitleIndex");
        queryIndex("DueDateIndex");

        deleteTable(tableName);

    }

    public static void createTable() {

        // Attribute definitions
        ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
```

```
       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IssueId").withAttributeType("S"));
       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("Title").withAttributeType("S"));
       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("CreateDate").withAttributeType("S"));
       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("DueDate").withAttributeType("S"));

       // Key schema for table
       ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
       tableKeySchema.add(new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.HASH)); //
Partition

                   // key
       tableKeySchema.add(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.RANGE)); // Sort

                  // key

       // Initial provisioned throughput settings for the indexes
       ProvisionedThroughput ptIndex = new
ProvisionedThroughput().withReadCapacityUnits(1L)
               .withWriteCapacityUnits(1L);

       // CreateDateIndex
       GlobalSecondaryIndex createDateIndex = new
GlobalSecondaryIndex().withIndexName("CreateDateIndex")
               .withProvisionedThroughput(ptIndex)
               .withKeySchema(new
KeySchemaElement().withAttributeName("CreateDate").withKeyType(KeyType.HASH), //
Partition

                         // key
                       new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                // key
               .withProjection(
                       new
Projection().withProjectionType("INCLUDE").withNonKeyAttributes("Description",
"Status"));
```

```
        // TitleIndex
        GlobalSecondaryIndex titleIndex = new
GlobalSecondaryIndex().withIndexName("TitleIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.HASH), // Partition

                    // key
                       new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                // key
                .withProjection(new Projection().withProjectionType("KEYS_ONLY"));

        // DueDateIndex
        GlobalSecondaryIndex dueDateIndex = new
GlobalSecondaryIndex().withIndexName("DueDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("DueDate").withKeyType(KeyType.HASH)) //
Partition

                      // key
                .withProjection(new Projection().withProjectionType("ALL"));

        CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                .withProvisionedThroughput(
                        new ProvisionedThroughput().withReadCapacityUnits((long)
1).withWriteCapacityUnits((long) 1))

.withAttributeDefinitions(attributeDefinitions).withKeySchema(tableKeySchema)
                .withGlobalSecondaryIndexes(createDateIndex, titleIndex, dueDateIndex);

        System.out.println("Creating table " + tableName + "...");
        dynamoDB.createTable(createTableRequest);

        // Wait for table to become active
        System.out.println("Waiting for " + tableName + " to become ACTIVE...");
        try {
            Table table = dynamoDB.getTable(tableName);
            table.waitForActive();
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
       }
   }

   public static void queryIndex(String indexName) {

       Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
       System.out.print("Querying index " + indexName + "...");

       Index index = table.getIndex(indexName);

       ItemCollection<QueryOutcome> items = null;

       QuerySpec querySpec = new QuerySpec();

       if (indexName == "CreateDateIndex") {
           System.out.println("Issues filed on 2013-11-01");
           querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
                   .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
           items = index.query(querySpec);
       } else if (indexName == "TitleIndex") {
           System.out.println("Compilation errors");
           querySpec.withKeyConditionExpression("Title = :v_title and
begins_with(IssueId, :v_issue)")
                   .withValueMap(
                           new ValueMap().withString(":v_title", "Compilation
error").withString(":v_issue", "A-"));
           items = index.query(querySpec);
       } else if (indexName == "DueDateIndex") {
           System.out.println("Items that are due on 2013-11-30");
           querySpec.withKeyConditionExpression("DueDate = :v_date")
                   .withValueMap(new ValueMap().withString(":v_date", "2013-11-30"));
           items = index.query(querySpec);
       } else {
           System.out.println("\nNo valid index name provided");
           return;
       }

       Iterator<Item> iterator = items.iterator();
```

```
        System.out.println("Query: printing results...");

        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    public static void deleteTable(String tableName) {

        System.out.println("Deleting table " + tableName + "...");

        Table table = dynamoDB.getTable(tableName);
        table.delete();

        // Wait for table to be deleted
        System.out.println("Waiting for " + tableName + " to be deleted...");
        try {
            table.waitForDelete();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void loadData() {

        System.out.println("Loading data into table " + tableName + "...");

        // IssueId, Title,
        // Description,
        // CreateDate, LastUpdateDate, DueDate,
        // Priority, Status

        putItem("A-101", "Compilation error", "Can't compile Project X - bad version
 number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10", 1, "Assigned");

        putItem("A-102", "Can't read data file", "The main data file is missing, or the
 permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30", 2, "In progress");

        putItem("A-103", "Test failure", "Functional test of Project X produces
 errors", "2013-11-01", "2013-11-02",
                "2013-11-10", 1, "In progress");
        putItem("A-104", "Compilation error", "Variable 'messageCount' was not
 initialized.", "2013-11-15",
                "2013-11-16", "2013-11-30", 3, "Assigned");

        putItem("A-105", "Network issue", "Can't ping IP address 127.0.0.1. Please fix
 this.", "2013-11-15",
                "2013-11-16", "2013-11-19", 5, "Assigned");

    }

    public static void putItem(

            String issueId, String title, String description, String createDate, String
 lastUpdateDate, String dueDate,
            Integer priority, String status) {

        Table table = dynamoDB.getTable(tableName);

        Item item = new Item().withPrimaryKey("IssueId", issueId).withString("Title",
 title)
                .withString("Description", description).withString("CreateDate",
 createDate)
                .withString("LastUpdateDate", lastUpdateDate).withString("DueDate",
 dueDate)
                .withNumber("Priority", priority).withString("Status", status);

        table.putItem(item);
    }

}
```

グローバルセカンダリインデックスの操作: .NET

AWS SDK for .NET 低レベル API を使用して、1 つ以上のグローバルセカンダリインデックスを含む Amazon DynamoDB テーブルを作成し、テーブルのインデックスを記述し、インデックスを使用してクエリを実行できます。これらのオペレーションは、対応する DynamoDB オペレーションにマッピングされます。詳細については、「Amazon DynamoDB API リファレンス」を参照してください。

以下に、.NET 低レベル API を使用したテーブルオペレーションの一般的な手順を示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. 対応するリクエストオブジェクトを作成して、オペレーションについて必要なパラメータとオプションパラメータを入力します。

   たとえば、テーブルを作成するには CreateTableRequest オブジェクトを作成し、テーブルやインデックスにクエリを実行するには QueryRequest オブジェクトを作成します。
3. 前述のステップで作成したクライアントから提供された適切なメソッドを実行します。

トピック

- グローバルセカンダリインデックスを持つテーブルを作成します
- グローバルセカンダリインデックスを持つテーブルの説明
- グローバルセカンダリインデックスのクエリ
- 例: AWS SDK for .NET 低レベル API を使用したグローバルセカンダリインデックス

グローバルセカンダリインデックスを持つテーブルを作成します

グローバルセカンダリインデックスは、テーブルの作成と同時に作成できます。これを行うには、CreateTable を使用し、1 つ以上のグローバルセカンダリインデックスの仕様を指定します。次の C# コード例では、気象データに関する情報を保持するテーブルを作成しています。パーティションキーは Location で、ソートキーは Date です。グローバルセカンダリインデックス PrecipIndex は、さまざまな場所の降水データへの高速アクセスを可能にします。

.NET 低レベル API を使用して、グローバルセカンダリインデックスを含むテーブルを作成する手順を次に示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. リクエスト情報を指定する CreateTableRequest クラスのインスタンスを作成します。

   テーブル名、プライマリキー、およびプロビジョニングされたスループット値を指定する必要があります。グローバルセカンダリインデックスの場合は、インデックス名、プロビジョニングされたスループット設定、インデックスソートキーの属性定義、インデックスのキースキーマ、および属性射影を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、CreateTable メソッドを実行します。

以下の C# サンプルコードは、前述のステップの例です。このコードは、グローバルセカンダリインデックス (PrecipIndex) を持つテーブル (WeatherData) を作成します。インデックスパーティ

ションキーは Date で、ソートキーは Precipitation です。すべてのテーブル属性がインデックスに射影されます。ユーザーは、このインデックスに対するクエリを実行して、特定の日付の気象データを取得できます。必要に応じて、降水量によってデータを並べ替えることもできます。

Precipitation はテーブルのキー属性ではないため、必須ではありません。ただし、Precipitation のない WeatherData 項目は PrecipIndex に表示されません。

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

// Attribute definitions
var attributeDefinitions = new List<AttributeDefinition>()
{
    {new AttributeDefinition{
        AttributeName = "Location",
        AttributeType = "S"}},
    {new AttributeDefinition{
        AttributeName = "Date",
        AttributeType = "S"}},
    {new AttributeDefinition(){
        AttributeName = "Precipitation",
        AttributeType = "N"}
    }
};

// Table key schema
var tableKeySchema = new List<KeySchemaElement>()
{
    {new KeySchemaElement {
        AttributeName = "Location",
        KeyType = "HASH"}},  //Partition key
    {new KeySchemaElement {
        AttributeName = "Date",
        KeyType = "RANGE"}  //Sort key
    }
};

// PrecipIndex
var precipIndex = new GlobalSecondaryIndex
{
    IndexName = "PrecipIndex",
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)10,
        WriteCapacityUnits = (long)1
    },
    Projection = new Projection { ProjectionType = "ALL" }
};

var indexKeySchema = new List<KeySchemaElement> {
    {new KeySchemaElement { AttributeName = "Date", KeyType = "HASH"}},  //Partition
 key
    {new KeySchemaElement{AttributeName = "Precipitation",KeyType = "RANGE"}}  //Sort
 key
};

precipIndex.KeySchema = indexKeySchema;

CreateTableRequest createTableRequest = new CreateTableRequest
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)5,
        WriteCapacityUnits = (long)1
    },
    AttributeDefinitions = attributeDefinitions,
    KeySchema = tableKeySchema,
    GlobalSecondaryIndexes = { precipIndex }
};

CreateTableResponse response = client.CreateTable(createTableRequest);
Console.WriteLine(response.CreateTableResult.TableDescription.TableName);
Console.WriteLine(response.CreateTableResult.TableDescription.TableStatus);
```

DynamoDB がテーブルを作成し、テーブルのステータスを `ACTIVE` に設定するまで待機する必要があります。その後、テーブルへのデータ項目の入力を開始できます。

グローバルセカンダリインデックスを持つテーブルの説明

テーブルでグローバルセカンダリインデックスに関する情報を取得するには、`DescribeTable` を使用します。インデックスごとに、名前、キースキーマ、および射影された属性にアクセスできます。

.NET 低レベル API を使用してテーブルのグローバルセカンダリインデックス情報にアクセスするには次の手順に従います。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. リクエストオブジェクトをパラメータとして指定して、describeTable メソッドを実行します。

   リクエスト情報を指定する DescribeTableRequest クラスのインスタンスを作成します。テーブル名を入力する必要があります。
3.

以下の C# サンプルコードは、前述のステップの例です。

Example

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest
 { TableName = tableName});

List<GlobalSecondaryIndexDescription> globalSecondaryIndexes =
response.DescribeTableResult.Table.GlobalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

foreach (GlobalSecondaryIndexDescription gsiDescription in globalSecondaryIndexes) {
     Console.WriteLine("Info for index " + gsiDescription.IndexName + ":");

     foreach (KeySchemaElement kse in gsiDescription.KeySchema) {
          Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
     }

      Projection projection = gsiDescription.Projection;
      Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

      if (projection.ProjectionType.ToString().Equals("INCLUDE")) {
           Console.WriteLine("\t\tThe non-key projected attributes are: "
                + projection.NonKeyAttributes);
      }
}
```

グローバルセカンダリインデックスのクエリ

テーブルに Query を実行するのとほぼ同じ方法で、グローバルセカンダリインデックスに対する Query を使用することができます。インデックス名、インデックスパーティションキーとソートキー (存在する場合) のクエリ条件、および返す属性を指定する必要があります。この例では、インデックスは PrecipIndex で、パーティションキーが Date で、ソートキーが Precipitation です。このインデックスクエリは、降水量がゼロより大きい特定の日付のすべての気象データを返します。

.NET 低レベル API を使用して、グローバルセカンダリインデックスを含むテーブルにクエリを実行する手順を次に示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. リクエスト情報を指定する QueryRequest クラスのインスタンスを作成します。
3. リクエストオブジェクトをパラメータとして指定して、query メソッドを実行します。

属性名 Date は DynamoDB の予約語です。したがって、式の属性名を KeyConditionExpression のプレースホルダーとして使用する必要があります。

以下の C# サンプルコードは、前述のステップの例です。

Example

```
client = new AmazonDynamoDBClient();

QueryRequest queryRequest = new QueryRequest
{
    TableName = "WeatherData",
    IndexName = "PrecipIndex",
    KeyConditionExpression = "#dt = :v_date and Precipitation > :v_precip",
    ExpressionAttributeNames = new Dictionary<String, String> {
        {"#dt", "Date"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_date", new AttributeValue { S =  "2013-08-01" }},
        {":v_precip", new AttributeValue { N =  "0" }}
    },
    ScanIndexForward = true
};
```

```
var result = client.Query(queryRequest);

var items = result.Items;
foreach (var currentItem in items)
{
    foreach (string attr in currentItem.Keys)
    {
        Console.Write(attr + "---> ");
        if (attr == "Precipitation")
        {
            Console.WriteLine(currentItem[attr].N);
    }
    else
    {
        Console.WriteLine(currentItem[attr].S);
    }

         }
     Console.WriteLine();
}
```

例: AWS SDK for .NET 低レベル API を使用したグローバルセカンダリインデックス

次の C# コード例は、グローバルセカンダリインデックスの操作方法を示しています。この例では、Issues というテーブルを作成しています。これは、ソフトウエア開発のためのシンプルなバグ追跡システムで使用できる可能性があります。パーティションキーは IssueId で、ソートキーは Title です。このテーブルには、次の 3 つのグローバルセカンダリインデックスがあります。

- CreateDateIndex – パーティションキーは CreateDate で、ソートキーは IssueId です。テーブルキーに加えて、属性 Description および Status はインデックスに射影されます。
- TitleIndex – パーティションキーは Title で、ソートキーは IssueId です。テーブルキー以外の属性はインデックスに射影されません。
- DueDateIndex – パーティションキーは DueDate で、ソートキーはありません。すべてのテーブル属性がインデックスに射影されます。

Issues テーブルが作成されると、プログラムはソフトウエアのバグレポートを表すデータをテーブルにロードします。次に、グローバルセカンダリインデックスを使用してデータのクエリを実行します。最後に、プログラムは Issues テーブルを削除します。

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelGlobalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        public static String tableName = "Issues";

        public static void Main(string[] args)
        {
            CreateTable();
            LoadData();

            QueryIndex("CreateDateIndex");
            QueryIndex("TitleIndex");
            QueryIndex("DueDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press enter");
            Console.Read();
        }

        private static void CreateTable()
        {
            // Attribute definitions
            var attributeDefinitions = new List<AttributeDefinition>()
        {
            {new AttributeDefinition {
                 AttributeName = "IssueId", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "Title", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "CreateDate", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "DueDate", AttributeType = "S"
             }}
        };

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>() {
            {
                new KeySchemaElement {
                    AttributeName= "IssueId",
                    KeyType = "HASH" //Partition key
                }
            },
            {
                new KeySchemaElement {
                    AttributeName = "Title",
                    KeyType = "RANGE" //Sort key
                }
            }
        };

            // Initial provisioned throughput settings for the indexes
            var ptIndex = new ProvisionedThroughput
            {
                ReadCapacityUnits = 1L,
                WriteCapacityUnits = 1L
            };

            // CreateDateIndex
            var createDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "CreateDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "CreateDate", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "INCLUDE",
                    NonKeyAttributes = {
                    "Description", "Status"
                }
                }
            };

            // TitleIndex
            var titleIndex = new GlobalSecondaryIndex()
            {
                IndexName = "TitleIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "Title", KeyType = "HASH" //Partition key
                },
                new KeySchemaElement {
                    AttributeName = "IssueId", KeyType = "RANGE" //Sort key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "KEYS_ONLY"
                }
            };

            // DueDateIndex
            var dueDateIndex = new GlobalSecondaryIndex()
            {
                IndexName = "DueDateIndex",
                ProvisionedThroughput = ptIndex,
                KeySchema = {
                new KeySchemaElement {
                    AttributeName = "DueDate",
                    KeyType = "HASH" //Partition key
                }
            },
                Projection = new Projection
                {
                    ProjectionType = "ALL"
                }
            };



            var createTableRequest = new CreateTableRequest
            {
                TableName = tableName,
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = (long)1,
                    WriteCapacityUnits = (long)1
                },
                AttributeDefinitions = attributeDefinitions,
                KeySchema = tableKeySchema,
                GlobalSecondaryIndexes = {
                createDateIndex, titleIndex, dueDateIndex
            }
            };

            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);

            WaitUntilTableReady(tableName);
        }

        private static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            // IssueId, Title,
            // Description,
            // CreateDate, LastUpdateDate, DueDate,
            // Priority, Status

            putItem("A-101", "Compilation error",
                "Can't compile Project X - bad version number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "Assigned");

            putItem("A-102", "Can't read data file",
                "The main data file is missing, or the permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30",
                2, "In progress");
            putItem("A-103", "Test failure",
                "Functional test of Project X produces errors",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "In progress");

            putItem("A-104", "Compilation error",
                "Variable 'messageCount' was not initialized.",
                "2013-11-15", "2013-11-16", "2013-11-30",
                3, "Assigned");

            putItem("A-105", "Network issue",
                "Can't ping IP address 127.0.0.1. Please fix this.",
                "2013-11-15", "2013-11-16", "2013-11-19",
                5, "Assigned");
        }

        private static void putItem(
            String issueId, String title,
            String description,
            String createDate, String lastUpdateDate, String dueDate,
            Int32 priority, String status)
        {
            Dictionary<String, AttributeValue> item = new Dictionary<string,
 AttributeValue>();

            item.Add("IssueId", new AttributeValue
            {
                S = issueId
            });
            item.Add("Title", new AttributeValue
            {
                S = title
            });
            item.Add("Description", new AttributeValue
            {
                S = description
            });
            item.Add("CreateDate", new AttributeValue
            {
                S = createDate
            });
            item.Add("LastUpdateDate", new AttributeValue
            {
                S = lastUpdateDate
            });
            item.Add("DueDate", new AttributeValue
            {
                S = dueDate
            });
            item.Add("Priority", new AttributeValue
            {
                N = priority.ToString()
            });
            item.Add("Status", new AttributeValue
            {
                S = status
            });

            try
            {
                client.PutItem(new PutItemRequest
                {
                    TableName = tableName,
                    Item = item
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }

        private static void QueryIndex(string indexName)
        {
            Console.WriteLine
                ("\n***********************************************************\n");
            Console.WriteLine("Querying index " + indexName + "...");

            QueryRequest queryRequest = new QueryRequest
            {
                TableName = tableName,
                IndexName = indexName,
                ScanIndexForward = true
            };


            String keyConditionExpression;
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue>();

            if (indexName == "CreateDateIndex")
            {
                Console.WriteLine("Issues filed on 2013-11-01\n");

                keyConditionExpression = "CreateDate = :v_date and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-01"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });
            }
            else if (indexName == "TitleIndex")
            {
                Console.WriteLine("Compilation errors\n");

                keyConditionExpression = "Title = :v_title and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_title", new AttributeValue
                {
                    S = "Compilation error"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else if (indexName == "DueDateIndex")
            {
                Console.WriteLine("Items that are due on 2013-11-30\n");

                keyConditionExpression = "DueDate = :v_date";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-30"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo valid index name provided");
                return;
            }

            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "Priority")
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].N);
                    }
                    else
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].S);
                    }
                }
                Console.WriteLine();
            }
        }

        private static void DeleteTable(string tableName)
        {
            Console.WriteLine("Deleting table " + tableName + "...");
            client.DeleteTable(new DeleteTableRequest
            {
                TableName = tableName
            });
            WaitForTableToBeDeleted(tableName);
        }

        private static void WaitUntilTableReady(string tableName)
        {
            string status = null;
            // Let us wait until table is created. Call DescribeTable.
            do
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                catch (ResourceNotFoundException)
                {
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

グローバルセカンダリインデックスの操作: AWS CLI

AWS CLI を使用して、1 つ以上のグローバルセカンダリインデックスを含む Amazon DynamoDB テーブルを作成し、テーブルのインデックスを記述し、インデックスを使用してクエリを実行できます。

トピック

- グローバルセカンダリインデックスを持つテーブルを作成します
- グローバルセカンダリインデックスを既存のテーブルに追加します
- グローバルセカンダリインデックスを持つテーブルの説明
- グローバルセカンダリインデックスのクエリ

グローバルセカンダリインデックスを持つテーブルを作成します

グローバルセカンダリインデックスは、テーブルの作成と同時に作成できます。 これを行うには、create-table パラメータを使用し、1 つ以上のグローバルセカンダリインデックスの仕様を指定します。次の例では、GameTitleIndex と呼ばれるグローバルセカンダリインデックスを含む GameScores という名前のテーブルを作成します。ベーステーブルには、パーティションキー UserId とソートキー GameTitle があり、特定のゲームの個々のユーザーのベストスコアを効率的に見つけることができます。一方、GSI にはパーティションキー GameTitle とソートキー TopScore があり、特定のゲームの全体的な最高スコアをすばやく見つけることができます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

DynamoDB がテーブルを作成し、テーブルのステータスを ACTIVE に設定するまで待機する必要があります。その後、テーブルへのデータ項目の入力を開始できます。describe-table を使用して、テーブル作成のステータスを判断できます。

グローバルセカンダリインデックスを既存のテーブルに追加します

グローバルセカンダリインデックスは、テーブルの作成後に追加したり変更したりすることもできます。これを行うには、update-table パラメータを使用し、1 つ以上のグローバルセカンダリインデックスの仕様を指定します。次の例では、前の例と同じスキーマを使用していますが、テーブルが既に作成されており、後で GSI を追加する場合を想定しています。

```
aws dynamodb update-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=TopScore,AttributeType=N  \
    --global-secondary-index-updates \
        "[
            {
                \"Create\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                    \"Projection\":{
                        \"ProjectionType\":\"INCLUDE\",
                        \"NonKeyAttributes\":[\"UserId\"]
                    }
                }
            }
        ]"
```

グローバルセカンダリインデックスを持つテーブルの説明

テーブルのグローバルセカンダリインデックスに関する情報を取得するには、describe-table パラメータを使用します。インデックスごとに、名前、キースキーマ、および射影された属性にアクセスできます。

```
aws dynamodb describe-table --table-name GameScores
```

グローバルセカンダリインデックスのクエリ

テーブルに query を実行するのとほぼ同じ方法で、グローバルセカンダリインデックスに対する query オペレーションを使用することができます。インデックス名、インデックスソートキーのクエリ条件、および返す属性を指定する必要があります。この例では、インデックスは GameTitleIndex、インデックスソートキーは GameTitle です。

返される属性は、インデックスに射影された属性だけです。このクエリを変更して非キー属性を選択することもできますが、これには比較的コストのかかるテーブルフェッチアクティビティが必要です。テーブルのフェッチの詳細については、「属性の射影」を参照してください。

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

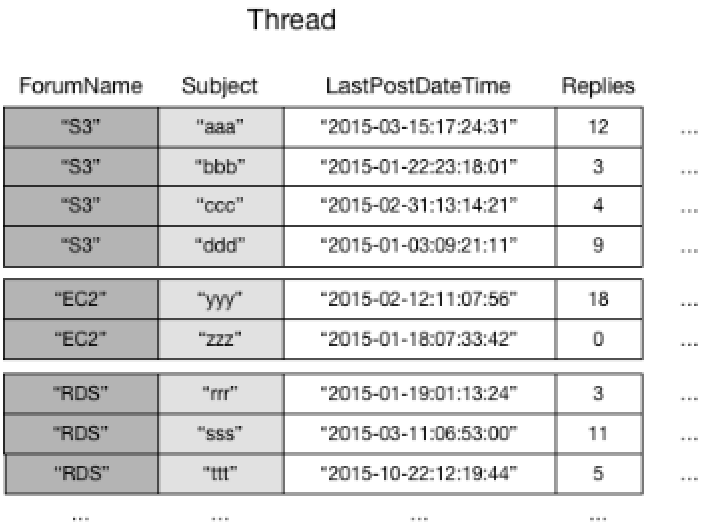
ローカルセカンダリインデックス

ベーステーブルのプライマリキーを使用してデータのクエリを実行する必要があるのは、一部のアプリケーションだけです。ただし、代替のソートキーが役に立つ場合があります。アプリケーションでソートキーを選択できるようにするには、Amazon DynamoDB テーブルに 1 つ以上のローカルセカンダリインデックスを作成し、これらのインデックスに対して Query または Scan のリクエストを発行できます。

トピック

シナリオ: ローカルセカンダリインデックスの使用

たとえば、`Thread` で定義した DynamoDB でのコード例用のテーブルの作成とデータのロード テーブルがあります。このテーブルは、AWS ディスカッションフォーラムのようなアプリケーションで役立ちます。次の図は、テーブル内の項目の構成を示しています。一部表示されていない属性もあります。



Thread

ForumName	Subject	LastPostDateTime	Replies	
“S3”	“aaa”	“2015-03-15:17:24:31”	12	...
“S3”	“bbb”	“2015-01-22:23:18:01”	3	...
“S3”	“ccc”	“2015-02-31:13:14:21”	4	...
“S3”	“ddd”	“2015-01-03:09:21:11”	9	...
“EC2”	“yyy”	“2015-02-12:11:07:56”	18	...
“EC2”	“zzz”	“2015-01-18:07:33:42”	0	...
“RDS”	“rrr”	“2015-01-19:01:13:24”	3	...
“RDS”	“sss”	“2015-03-11:06:53:00”	11	...
“RDS”	“ttt”	“2015-10-22:12:19:44”	5	...
...	...	...	...	

DynamoDB には、同じパーティションキー値を持つすべての項目が継続的に保存されます。この例では、特定の `ForumName` を指定すると、`Query` オペレーションはそのフォーラムのすべてのスレッドをすばやく見つけることができます。同じパーティションキー値を持つ項目のグループでは、

項目がソートキーの値によって並べ替えられます。ソートキー (Subject) がクエリでも提供される場合、DynamoDB は返される結果を絞り込むことができます。例えば、「S3」フォーラムで、文字「a」で始まる Subject を持つすべてのスレッドを返すことができます。

リクエストによっては、より複雑なデータアクセスパターンが要求される場合があります。以下に例を示します。

- ビューと返信の数が最も多いのはどのフォーラムか?
- 特定のフォーラムでメッセージ数が最も多いのはどのスレッドか?
- 特定の期間中に特定のフォーラムに投稿されたスレッド数は?

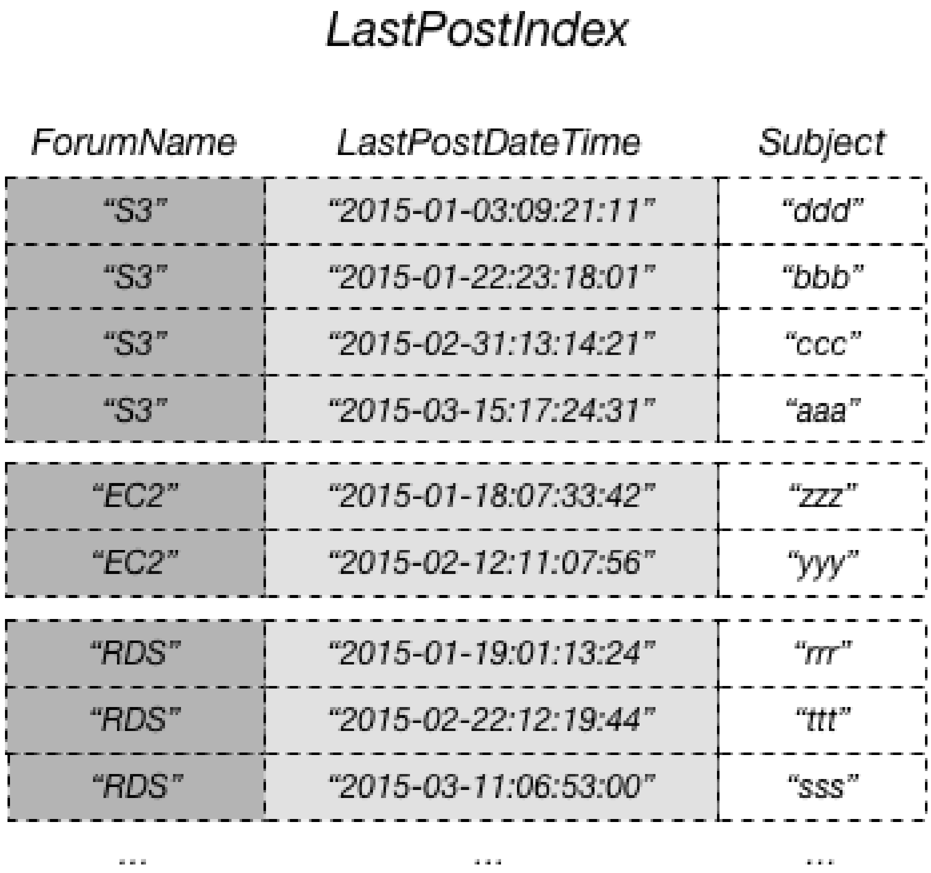
これらの質問に答えるには、Query アクションでは十分ではありません。代わりに、テーブル全体の Scan を実行する必要があります。膨大な数の項目があるテーブルでは、これにより、プロビジョニングされた読み込みスループットが大量に消費されることになり、完了するまでに長時間かかります。

ただし、Replies や LastPostDateTime などの非キー属性に 1 つ以上のローカルセカンダリインデックスを指定することができます。

ローカルセカンダリインデックスは特定のパーティションキー値の代替ソートキーを維持します。またローカルセカンダリインデックスには、ベーステーブルの一部またはすべての属性のコピーが含まれます。テーブルを作成する際に、ローカルセカンダリインデックスに射影する属性を指定します。ローカルセカンダリインデックスのデータは、ベーステーブルと同じパーティションキーと、異なるソートキーで構成されます。これにより、この異なるディメンションにわたってデータ項目に効率的にアクセスできます。クエリまたはスキャンの柔軟性を向上させるために、テーブルごとに最大 5 つのローカルセカンダリインデックスを作成できます。

たとえば、あるアプリケーションで、特定のフォーラムに過去 3 か月以内に投稿されたすべてのスレッドを検索する必要があるとします。ローカルセカンダリインデックスがない場合、アプリケーションは Scan テーブル全体の Thread を実行して、指定された時間枠から外れた投稿を破棄する必要があります。ローカルセカンダリインデックスがあれば、Query オペレーションでは LastPostDateTime をソートキーとして使用し、データをすばやく見つけることができます。

次の図は、LastPostIndex という名前のローカルセカンダリインデックスを示しています。パーティションキーは Thread テーブルのパーティションキーと同じですが、ソートキーは LastPostDateTime です。



LastPostIndex

ForumName	LastPostDateTime	Subject
"S3"	"2015-01-03:09:21:11"	"ddd"
"S3"	"2015-01-22:23:18:01"	"bbb"
"S3"	"2015-02-31:13:14:21"	"ccc"
"S3"	"2015-03-15:17:24:31"	"aaa"
"EC2"	"2015-01-18:07:33:42"	"zzz"
"EC2"	"2015-02-12:11:07:56"	"yyy"
"RDS"	"2015-01-19:01:13:24"	"rrr"
"RDS"	"2015-02-22:12:19:44"	"ttt"
"RDS"	"2015-03-11:06:53:00"	"sss"
...	...	...

ローカルセカンダリインデックスは、すべて次の条件を満たす必要があります。

- パーティションキーはそのベーステーブルのパーティションキーと同じである。
- ソートキーは完全に 1 つのスカラー属性で構成されている。
- ベーステーブルのソートキーがインデックスに射影され、非キー属性として機能する。

この例で、パーティションキーは ForumName、ローカルセカンダリインデックスのソートキーは LastPostDateTime です。さらに、ベーステーブルのソートキーの値 (この例では Subject) がインデックスに射影されますが、その値はインデックスキーの一部ではありません。アプリケーションが ForumName と LastPostDateTime に基づくリストを必要とする場合は、Query に対して LastPostIndex リクエストを実行できます。クエリ結果は LastPostDateTime によってソートされ、昇順または降順で返すことができます。このクエリは、特定の時間枠内に LastPostDateTime がある項目だけを返すなどのキー条件を適用することもできます。

すべてのローカルセカンダリインデックスには、そのベーステーブルからのパーティションキーとソートキーが自動的に格納されます。必要に応じて、非キー属性をインデックスに射影できます。インデックスのクエリを行うと、DynamoDB ではこれらの射影された属性を効率的に取り出すこと

ができます。ローカルセカンダリインデックスにクエリを実行すると、インデックスに射影されていない属性もクエリで取得できます。DynamoDB では、これらの属性をベーステーブルから自動的にフェッチしますが、レイテンシーが大きくなり、プロビジョニングされたスループットコストが高くなります。

任意のローカルセカンダリインデックスについて、異なるパーティションキーの値ごとに最大 10 GB のデータを保存できます。この数字には、ベーステーブル内のすべての項目に加えて、同じパーティションキーの値を持つインデックス内のすべての項目も含まれています。詳細については、「ローカルセカンダリインデックス内の項目コレクション」を参照してください。

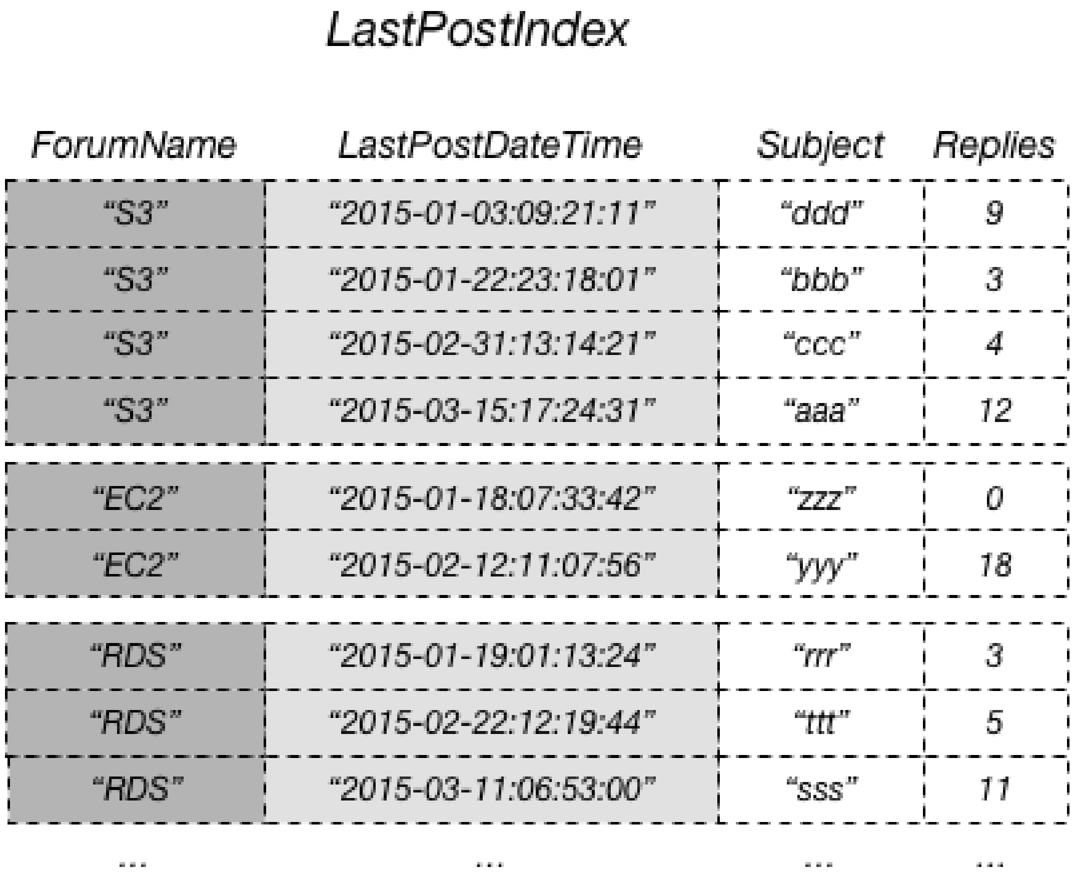
属性の射影

LastPostIndex では、アプリケーションはクエリの基準として ForumName と LastPostDateTime を使用できます。ただし、追加の属性を取得する場合、DynamoDB は Thread テーブルに対して追加の読み取りオペレーションを実行する必要があります。このような追加の読み込みをフェッチと言い、それによってクエリにプロビジョニングするスループットの合計量が増加することがあります。

ウェブページに「S3」内のすべてのスレッドと各スレッドの返信のリストを表示し、最新の返信の最後の返信日時で並べ替えるとします。このリストに入力するには、次の属性が必要になります。

- Subject
- Replies
- LastPostDateTime

このデータのクエリを最も効率的に行い、フェッチオペレーションを回避するには、次の図に示すように、ローカルセカンダリインデックスにテーブルからの Replies 属性を射影します。



LastPostIndex

ForumName	LastPostDateTime	Subject	Replies
“S3”	“2015-01-03:09:21:11”	“ddd”	9
“S3”	“2015-01-22:23:18:01”	“bbb”	3
“S3”	“2015-02-31:13:14:21”	“ccc”	4
“S3”	“2015-03-15:17:24:31”	“aaa”	12
“EC2”	“2015-01-18:07:33:42”	“zzz”	0
“EC2”	“2015-02-12:11:07:56”	“yyy”	18
“RDS”	“2015-01-19:01:13:24”	“rrr”	3
“RDS”	“2015-02-22:12:19:44”	“ttt”	5
“RDS”	“2015-03-11:06:53:00”	“sss”	11
...	...	...	...

射影とは、テーブルからセカンダリインデックスにコピーされる属性のセットです。テーブルのパーティションキーとソートキーは常にインデックスに射影されます。アプリケーションのクエリ要件をサポートするために、他の属性を射影できます。インデックスをクエリすると、Amazon DynamoDB は、その属性が自身のテーブル内にあるかのように、プロジェクション内の任意の属性にアクセスできます。

セカンダリインデックスを作成するときは、インデックスに射影される属性を指定する必要があります。DynamoDB には、そのために次の 3 つのオプションが用意されています。

- KEYS_ONLY – インデックス内の各項目は、テーブルパーティションキーとソートキーの値、およびインデックスキーの値のみで構成されます。KEYS_ONLY オプションを指定すると、セカンダリインデックスが最小になります。
- INCLUDE – KEYS_ONLY の属性に加えて、セカンダリインデックスにその他の非キー属性が含まれるように指定できます。

- ALL – セカンダリインデックスには、ソーステーブルのすべての属性が含まれます。すべてのテーブルデータがインデックスに複製されるため、ALL の射影にすると、セカンダリインデックスが最大になります。

前の図では、非キー属性の Replies が LastPostIndex に射影されています。アプリケーションでは LastPostIndex テーブル全体ではなく Thread に対するクエリを実行し、ウェブページに Subject、Replies、LastPostDateTime を表示することができます。他の非キー属性がリクエストされた場合、DynamoDB はそれらの属性を Thread テーブルからフェッチする必要があります。

アプリケーションの観点から見ると、ベーステーブルから追加の属性をフェッチする処理は、自動的で透過的なので、アプリケーションロジックを書き直す必要はありません。ただし、このようなフェッチによって、ローカルセカンダリインデックスを使用することで得られるパフォーマンスの利点が大幅に小さくなる可能性があります。

ローカルセカンダリインデックスに射影する属性を選択する場合には、プロビジョニングされるスループットコストとストレージコストのトレードオフを考慮する必要があります。

- ごく一部の属性だけに最小のレイテンシーでアクセスする必要がある場合は、それらの属性だけをローカルセカンダリインデックスに射影することを検討してください。インデックスが小さいほど少ないコストで格納でき、書き込みコストも低くなります。まれにしかフェッチされない属性がある場合には、それらの属性を格納するコストよりも、プロビジョニングされるスループットのコストのほうが長期的に大きくなる可能性があります。
- アプリケーションが非キー属性に頻繁にアクセスする場合には、それらの属性をローカルセカンダリインデックスに射影することを検討してください。ローカルセカンダリインデックスのストレージコストの増加は、頻繁にテーブルスキャンを実行するコストの減少で相殺されます。
- ほとんどの非キー属性に頻繁にアクセスする場合は、それらの属性を (場合によっては、ベーステーブル全体を) ローカルセカンダリインデックスに射影することができます。それによってフェッチが不要になるため、柔軟性が最大になり、プロビジョニングされるスループットが最小限になります。ただし、ストレージコストが増加し、すべての属性を射影する場合には 2 倍にまで増大します。
- アプリケーションでテーブルのクエリを頻繁に行う必要がなく、テーブル内のデータに対する書き込みや更新が多数になる場合は、KEYS_ONLY を射影することを検討してください。ローカルセカンダリインデックスは、最小サイズになりますが、それでもクエリのアクティビティに必要な場合は利用可能です。

ローカルセカンダリインデックスの作成

テーブルで 1 つ以上のローカルセカンダリインデックスを作成するには、CreateTable オペレーションの LocalSecondaryIndexes パラメータを使用します。テーブルのローカルセカンダリインデックスは、テーブルが作成された時点で作成されます。テーブルを削除すると、そのテーブルにあるローカルセカンダリインデックスも削除されます。

ローカルセカンダリインデックスのソートキーになる 1 つの非キー属性を指定する必要があります。選択する属性はスカラー String、Number、または Binary である必要があります。その他のスカラー型、ドキュメント型、およびセット型は許可されません。データ型の詳細なリストについては、「データ型」を参照してください。

> **Important**
>
> ローカルセカンダリインデックスがあるテーブルには、パーティションキーの値ごとに 10 GB のサイズ制限があります。ローカルセカンダリインデックスがあるテーブルには、1 つのパーティションキー値の合計サイズが 10 GB を超えない限り、任意の数の項目を格納できます。詳細については、「」を参照してください項目コレクションのサイズ制限

ローカルセカンダリインデックスには、任意のデータ型の属性を射影できます。これには、スカラー、ドキュメント、およびセットが含まれます。データ型の詳細なリストについては、「データ型」を参照してください。

ローカルセカンダリインデックスからのデータの読み込み

Query および Scan オペレーションを使用して、ローカルセカンダリインデックスから項目を取得できます。GetItem および BatchGetItem オペレーションは、ローカルセカンダリインデックスでは使用できません。

ローカルセカンダリインデックスのクエリ

DynamoDB テーブルでは、各項目のパーティションキー値とソートキー値の組み合わせは、一意である必要があります。ただし、ローカルセカンダリインデックスでは、ソートキーの値は、特定のパーティションキーの値に対して一意である必要がありません。ローカルセカンダリインデックスに同じソートキーを持つ複数の項目がある場合、Query オペレーションは、同じパーティションキーの値を持つすべての項目を返します。レスポンスでは、一致する項目は特定の順序で返されません。

結果整合性の読み込みまたは完全整合性の読み込みを使用して、ローカルセカンダリインデックスのクエリを行うことができます。必要な整合性のタイプを指定するには、ConsistentRead オペ

レーションの Query パラメータを使用します。ローカルセカンダリインデックスからの完全整合性の読み込みでは、常に最新の更新された値が返されます。クエリがベーステーブルからさらに属性をフェッチする必要がある場合、それらの属性はインデックスについて整合性があることになります。

Example

特定のフォーラムのディスカッションスレッドにあるデータをリクエストする Query から返される次のデータを考えてみます。

```
{
    "TableName": "Thread",
    "IndexName": "LastPostIndex",
    "ConsistentRead": false,
    "ProjectionExpression": "Subject, LastPostDateTime, Replies, Tags",
    "KeyConditionExpression":
        "ForumName = :v_forum and LastPostDateTime between :v_start and :v_end",
    "ExpressionAttributeValues": {
        ":v_start": {"S": "2015-08-31T00:00:00.000Z"},
        ":v_end": {"S": "2015-11-31T00:00:00.000Z"},
        ":v_forum": {"S": "EC2"}
    }
}
```

このクエリでは次のようになっています。

- DynamoDB は ForumName パーティションキーを使用して LastPostIndex にアクセスし、「EC2」のインデックス項目を特定します。このキーを持つすべてのインデックス項目が、すばやく取り出せるように隣り合わせに格納されます。
- このフォーラム内で、DynamoDB はインデックスを使用して、指定された LastPostDateTime 条件に一致するキーを検索します。
- Replies 属性はインデックスに射影されているため、DynamoDB は追加でプロビジョニングされたスループットを使用することなく、この属性を取得できます。
- Tags 属性はインデックスに射影されていないため、DynamoDB は Thread テーブルにアクセスしてこの属性をフェッチする必要があります。
- 結果が、LastPostDateTime によってソートされ、返されます。インデックスのエントリはまずパーティションキーの値によって、次にソートキーの値によって並べ替えられ、保存される順序で Query から返されます（ScanIndexForward パラメータを使用すると、結果が降順で返されます）。

ローカルセカンダリインデックスには Tags 属性が射影されていないため、DynamoDB は、読み取りキャパシティユニットを追加で使用して、ベーステーブルからこの属性をフェッチする必要があります。このクエリを頻繁に実行する必要がある場合は、Tags 属性を LastPostIndex に射影して、ベーステーブルからフェッチされないようにする必要があります。ただし、Tags 属性をまれにしか使用しない場合は、Tags 属性をインデックスに射影するためにストレージを追加することが有効でない可能性があります。

ローカルセカンダリインデックスのスキャン

Scan を使用して、ローカルセカンダリインデックスからすべてのデータを取得できます。リクエストにはベーステーブル名とインデックス名を指定する必要があります。Scan では、DynamoDB はインデックスのすべてのデータを読み込み、それをアプリケーションに返します。また、データの一部のみを返し、残りのデータを破棄するようにリクエストすることもできます。これを行うには、FilterExpression API の Scan パラメータを使用します。詳細については、「Scan のフィルタ式」を参照してください。

項目の書き込みとローカルセカンダリインデックス

DynamoDB によって、すべてのローカルセカンダリインデックスがそれぞれのベーステーブルと自動的に同期されます。アプリケーションがインデックスに直接書き込むことはありません。ただし、DynamoDB でこれらのインデックスがどのように維持されるかを理解することは重要です。

ローカルセカンダリインデックスを作成する場合は、インデックスのソートキーになる属性を指定します。その属性のデータ型も指定します。つまり、ベーステーブルに項目を書き込むとき、その項目にインデックスキー属性が定義されている場合は、その型がインデックスキースキーマのデータ型に一致する必要があります。LastPostIndex の場合、インデックス内の LastPostDateTime ソートキーは、String データ型として定義されています。Thread テーブルに項目を追加し、LastPostDateTime に対して別のデータ型を指定する場合 (Number など)、データ型の不一致により DynamoDB によって ValidationException が返されます。

ベーステーブル内の項目とローカルセカンダリインデックス内の項目を 1 対 1 の関係にする必要はありません。この動作は、多くのアプリケーションにとってメリットになります。

多数のローカルセカンダリインデックスがあるテーブルは、インデックス数が少ないテーブルに比べて書き込みアクティビティに多くのコストを要します。詳細については、「ローカルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項」を参照してください。

⚠ Important

ローカルセカンダリインデックスがあるテーブルには、パーティションキーの値ごとに 10 GB のサイズ制限があります。ローカルセカンダリインデックスがあるテーブルには、1 つのパーティションキー値の合計サイズが 10 GB を超えない限り、任意の数の項目を格納できます。詳細については、「項目コレクションのサイズ制限」を参照してください。

ローカルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項

DynamoDB にテーブルを作成する場合には、テーブルで予想されるワークロードに応じた読み込み/書き込みキャパシティーユニットをプロビジョニングします。このワークロードには、テーブルのローカルセカンダリインデックスの読み込み/書き込みアクティビティが含まれます。

プロビジョニングされたスループット容量の現在の料金を確認するには、「Amazon DynamoDB 料金」を参照してください。

読み込みキャパシティユニット

ローカルセカンダリインデックスに対してクエリを実行する場合、使用される読み込みキャパシティーユニットの数は、データのアクセス方法によって異なります。

テーブルに対するクエリと同様に、インデックスクエリでは `ConsistentRead` の値に応じて、結果整合性のある読み込みまたは強力な整合性のある読み込みを実行できます。1 回の強力な整合性のある読み込みでは、1 つの読み込みキャパシティーユニットが消費され、結果整合性のある読み込みではその半分が消費されます。したがって結果整合性のある読み込みを選択することで、読み込みキャパシティーユニットのコストを削減できます。

インデックスキーと射影された属性だけをリクエストするインデックスクエリの場合、DynamoDB はテーブルに対するクエリの場合と同じ方法で、プロビジョニングされた読み込みアクティビティを計算します。唯一の違いは、ベーステーブル内の項目のサイズではなくインデックスエントリのサイズに基づいて計算が行われることです。読み込みキャパシティーユニットの数は、返されたすべての項目について射影されたすべての属性のサイズの合計です。結果は、次の 4 KB 境界まで切り上げられます。DynamoDB がプロビジョニングされたスループットの利用率を計算する方法の詳細については、「プロビジョンドキャパシティモード」を参照してください。

ローカルセカンダリインデックスに射影されていない属性を読み込むインデックスクエリの場合、DynamoDB は射影された属性をインデックスから読み込むのに加えて、それらの属性をベーステーブルからフェッチする必要があります。これらのフェッチは、`Select` オペレーションの

ProjectionExpression または Query パラメータに、射影されていない属性を含めた場合に実行されます。フェッチによってクエリ応答のレイテンシーが増加し、プロビジョニングされるスループットのコストも増加します。前述のローカルセカンダリインデックスからの読み込みに加えて、フェッチされるベーステーブル項目それぞれについて、読み込みキャパシティーユニットの料金がかかります。この料金は、リクエストした属性だけでなく、項目全体をテーブルから読み込むために発生するものです。

Query オペレーションによって返される結果の最大サイズは 1 MB です。これには、返されるすべての項目にわたる、すべての属性の名前と値のサイズが含まれます。ただし、ローカルセカンダリインデックスに対するクエリによって、DynamoDB がベーステーブルから項目の属性をフェッチする場合には、結果で示されるデータの最大サイズが小さくなる可能性があります。この場合、結果のサイズは次の合計になります。

- インデックス内で一致する項目のサイズ (次の 4 KB に切り上げ)
- ベーステーブル内で一致する各項目のサイズ (項目ごとに次の 4 KB に切り上げ)

この式を使用すると、クエリオペレーションによって返される結果の最大サイズは依然として 1 MB になります。

たとえば、各項目のサイズが 300 バイトであるテーブルがあるとします。そのテーブルにはローカルセカンダリインデックスがありますが、各項目のうち 200 バイトだけがインデックスに射影されます。このインデックスに対して Query を行うときに各項目についてテーブルのフェッチが必要で、クエリによって 4 つの項目が返されるとします。DynamoDB では、次のように合計されます。

- インデックス内で一致する項目のサイズは 200 バイト × 4 項目 = 800 バイトになり、それが 4 KB に切り上げられます。
- ベーステーブル内で一致する項目のサイズ: (300 バイト、4 KB に切り上げ) × 4 項目 = 16 KB。

それによって、結果データの合計サイズは 20 KB になります。

書き込みキャパシティユニット

テーブル内の項目が追加、更新、または削除された場合にローカルセカンダリインデックスを更新すると、テーブルにプロビジョニングされた書き込みキャパシティーユニットが消費されます。書き込み用にプロビジョニングされたスループットの合計コストは、テーブルに対する書き込みと、ローカルセカンダリインデックスの更新で消費された書き込みキャパシティーユニットの合計になります。

ローカルセカンダリインデックスに項目を書き込むコストは、いくつかの要因に左右されます。

- インデックス付き属性が定義されたテーブルに新しい項目を書き込む場合、または既存の項目を更新して未定義のインデックス付き属性を定義する場合には、インデックスへの項目の挿入に 1 回の書き込みオペレーションが必要です。
- テーブルに対する更新によってインデックス付きキー属性の値が（A から B に）変更された場合には、インデックスから既存の項目を削除し、インデックスに新しい項目を挿入するために、2 回の書き込みが必要です。
- インデックス内に既存の項目があって、テーブルに対する書き込みによってインデックス付き属性が削除された場合は、インデックスから古い項目の射影を削除するために、1 回の書き込みが必要です。
- 項目の更新の前後にインデックス内に項目が存在しない場合は、インデックスで追加の書き込みコストは発生しません。

これらすべての要因は、インデックス内の各項目のサイズが 1 KB 以下であるという前提で書き込みキャパシティーユニット数を算出します。インデックスエントリがそれよりも大きい場合は、書き込みキャパシティーユニットを追加する必要があります。クエリが返す必要がある属性を特定し、それらの属性だけをインデックスに射影することで、書き込みコストは最小になります。

ローカルセカンダリインデックスのストレージに関する考慮事項

アプリケーションがテーブルに項目を書き込むと、DynamoDB では正しい属性のサブセットが、それらの属性が表示されるローカルセカンダリインデックスに自動的にコピーされます。AWS アカウントでは、テーブル内の項目のストレージと、そのベーステーブルのローカルセカンダリインデックスにある属性のストレージに対して課金されます。

インデックス項目が使用するスペースの量は、次の量の合計になります。

- ベーステーブルのプライマリキー (パーティションキーとソートキー) のサイズのバイト数
- インデックスキー属性のサイズのバイト数
- 射影された属性（存在する場合）のサイズのバイト数
- インデックス項目あたり 100 バイトのオーバーヘッド

ローカルセカンダリインデックスに必要なストレージは、インデックス内の 1 項目の平均サイズの見積もり値に、インデックス内の項目数を掛けて見積もることができます。

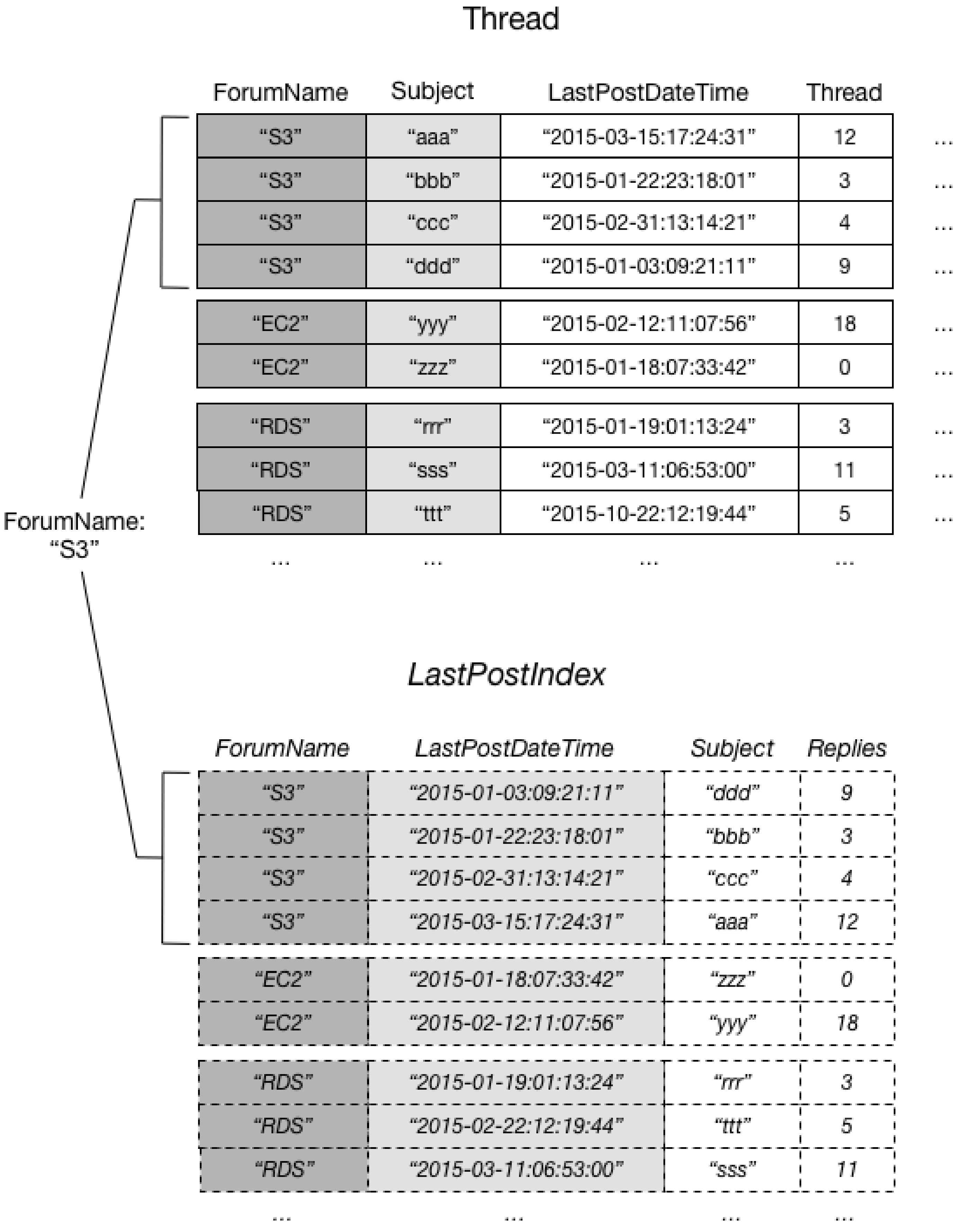
特定の属性が定義されていない項目がテーブルに含まれており、その属性がインデックスソートキーとして定義されている場合、DynamoDB はその項目に関連するデータをインデックスに書き込みません。

ローカルセカンダリインデックス内の項目コレクション

> **Note**
>
> このセクションは、ローカルセカンダリインデックスがあるテーブルだけに関係します。

DynamoDB において、項目コレクションとは、テーブルおよびそのローカルセカンダリインデックス全体で同じパーティションキーの値を持つ項目のグループです。このセクションで使用する例では、`Thread` テーブルのパーティションキーは `ForumName` で、`LastPostIndex` のパーティションキーも `ForumName` です。同じ `ForumName` を持つテーブルとインデックス項目は、すべて同じ項目コレクションの一部です。例えば、`Thread` テーブルと `LastPostIndex` ローカルセカンダリインデックスには、フォーラム EC2 用の項目コレクションと、フォーラム RDS 用の別の項目コレクションがあります。

次の図は、フォーラム S3 の項目コレクションを示しています。



Thread

ForumName	Subject	LastPostDateTime	Thread	
“S3”	“aaa”	“2015-03-15:17:24:31”	12	…
“S3”	“bbb”	“2015-01-22:23:18:01”	3	…
“S3”	“ccc”	“2015-02-31:13:14:21”	4	…
“S3”	“ddd”	“2015-01-03:09:21:11”	9	…
“EC2”	“yyy”	“2015-02-12:11:07:56”	18	…
“EC2”	“zzz”	“2015-01-18:07:33:42”	0	…
“RDS”	“rrr”	“2015-01-19:01:13:24”	3	…
“RDS”	“sss”	“2015-03-11:06:53:00”	11	…
“RDS”	“ttt”	“2015-10-22:12:19:44”	5	…
…	…	…	…	

LastPostIndex

ForumName	*LastPostDateTime*	*Subject*	*Replies*
“S3”	*“2015-01-03:09:21:11”*	*“ddd”*	*9*
“S3”	*“2015-01-22:23:18:01”*	*“bbb”*	*3*
“S3”	*“2015-02-31:13:14:21”*	*“ccc”*	*4*
“S3”	*“2015-03-15:17:24:31”*	*“aaa”*	*12*
“EC2”	*“2015-01-18:07:33:42”*	*“zzz”*	*0*
“EC2”	*“2015-02-12:11:07:56”*	*“yyy”*	*18*
“RDS”	*“2015-01-19:01:13:24”*	*“rrr”*	*3*
“RDS”	*“2015-02-22:12:19:44”*	*“ttt”*	*5*
“RDS”	*“2015-03-11:06:53:00”*	*“sss”*	*11*
…	…	…	…

この図では、項目コレクションは、`Thread` および `LastPostIndex` 内のすべての項目で構成されていて、`ForumName` パーティションキーの値は「S3」です。テーブルにその他のローカルセカンダリインデックスがあった場合には、`ForumName` が「S3」であるそれらのインデックス内の項目も、項目コレクションの一部になります。

DynamoDB では次のオペレーションのいずれかを使用して、項目コレクションに関する情報を返すことができます。

- `BatchWriteItem`
- `DeleteItem`
- `PutItem`
- `UpdateItem`
- `TransactWriteItems`

これらのオペレーションでは、それぞれ `ReturnItemCollectionMetrics` パラメータがサポートされています。このパラメータを `SIZE` に設定すると、インデックス内の各項目コレクションのサイズに関する情報が表示されます。

Example

以下に、`UpdateItem` が `Thread` に設定されている、`ReturnItemCollectionMetrics` テーブルでの `SIZE` オペレーションの出力の例を示します。更新された項目には `ForumName` 値「EC2」があったため、出力にはその項目コレクションに関する情報が含まれています。

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

`SizeEstimateRangeGB` オブジェクトは、この項目コレクションのサイズが 0 GB と 1 GB の間であることを示します。DynamoDB では、このサイズ見積りが定期的に更新されるため、項目を変更した時には次回の数字が異なる場合があります。

項目コレクションのサイズ制限

1 つ以上のローカルセカンダリインデックスを含む項目コレクションのサイズは、すべて最大 10 GB です。これは、ローカルセカンダリインデックスのないテーブルの項目コレクションには適用されません。また、グローバルセカンダリインデックスの項目コレクションにも適用されません。1 つ以上のローカルセカンダリインデックスを含むテーブルのみに影響します。

項目コレクションが 10 GB 制限を超えると、DynamoDB は `ItemCollectionSizeLimitExceededException` を返します。この場合、その項目コレクションに項目を追加することも、項目サイズを大きくすることもできません。（項目コレクションのサイズを小さくする読み込み/書き込みオペレーションは、引き続き実行できます)。その他の項目コレクションには項目を追加することができます。

項目コレクションのサイズを小さくするには、次のいずれかを実行します。

- 問題になっているパーティションキーの値を持つ不要な項目を削除します。ベーステーブルからこれらの項目を削除すると、DynamoDB では同じパーティションキーの値を持つインデックスエントリも削除されます。
- 属性を削除するか属性のサイズを小さくすることで、項目を更新します。これらの属性がローカルセカンダリインデックスに射影されている場合には、DynamoDB では対応するインデックスエントリのサイズも小さくなります。
- 同じパーティションキーおよびソートキーを使用して新しいテーブルを作成し、古いテーブルから新しいテーブルに項目を移動します。これは、頻繁にアクセスされない履歴データがテーブルに含まれている場合に適した方法です。また、この履歴データを Amazon Simple Storage Service (Amazon S3) にアーカイブすることを検討することもできます。

項目コレクションの合計サイズが 10 GB 未満になると、再び同じパーティションキーの値を使用して項目を追加できます。

項目コレクションのサイズを監視するようにアプリケーションを設定することをお勧めします。1 つの方法としては、`ReturnItemCollectionMetrics`、`SIZE`、`BatchWriteItem`、または `DeleteItem` を使用する場合に、`PutItem` パラメータを `UpdateItem` に設定するというものがあります。アプリケーションで出力内の `ReturnItemCollectionMetrics` オブジェクトを調査し、項目コレクションがユーザー定義の制限（たとえば 8 GB）を超えた場合にエラーメッセージを記録するようにします。制限を 10 GB より低く設定すれば早期警告システムになり、項目コレクションが上限に達する前に余裕をもって何らかの対処をとることができます。

項目コレクションおよびパーティション

1 つ以上のローカルセカンダリインデックスを含むテーブルでは、各項目コレクションは 1 つのパーティションに保存されます。このような項目コレクションの合計サイズは、そのパーティションのキャパシティー (10 GB) に制限されます。データモデルにサイズに制限のない項目コレクションが含まれている場合や、一部の項目コレクションが将来に 10 GB を超えると合理的に予期されるアプリケーションでは、代わりにグローバルセカンダリインデックスの使用を検討します。

アプリケーションを設計する場合は、テーブルデータが異なるパーティションキー値に均一に分配されるようにする必要があります。ローカルセカンダリインデックス を持つテーブルについては、単一のパーティション内の単一の項目コレクションに読み込み/書き込みアクティビティの「ホットスポット」が作られないように、アプリケーションを設計します。

ローカルセカンダリインデックスの操作: Java

AWS SDK for Java ドキュメント API を使用して、1 つ以上のローカルセカンダリインデックスを持つ Amazon DynamoDB テーブルを作成し、テーブルのインデックスを記述し、インデックスを使用してクエリを実行できます。

以下に、AWS SDK for Java ドキュメント API を使用したテーブルオペレーションの一般的な手順を示します。

1. `DynamoDB` クラスのインスタンスを作成します。
2. 対応するリクエストオブジェクトを作成して、オペレーションについて必要なパラメータとオプションパラメータを入力します。
3. 前のステップで作成したクライアントが提供する適切なメソッドを呼び出します。

トピック

ローカルセカンダリインデックスを持つテーブルを作成する

ローカルセカンダリインデックスは、テーブルの作成と同時に作成する必要があります。これを行うには、`createTable` メソッドを使用し、1 つ以上のローカルセカンダリインデックスの仕様を指定

します。次の Java コード例では、ミュージックコレクション内の曲に関する情報を保持するためのテーブルを作成しています。パーティションキーは Artist で、ソートキーは SongTitle です。セカンダリインデックス AlbumTitleIndex は、アルバムタイトルによるクエリを容易にします。

DynamoDB ドキュメント API を使用して、ローカルセカンダリインデックスを持つテーブルを作成する手順を次に示します。

1. DynamoDB クラスのインスタンスを作成します。
2. リクエスト情報を指定する CreateTableRequest クラスのインスタンスを作成します。

   テーブル名、プライマリキー、およびプロビジョニングされたスループット値を指定する必要があります。ローカルセカンダリインデックスの場合は、インデックス名、インデックスソートキーの名前とデータ型、インデックスのキースキーマ、および属性射影を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、createTable メソッドを呼び出します。

以下の Java コード例は、前述のステップの例です。このコードは、AlbumTitle 属性に関するセカンダリインデックスを持つテーブル (Music) を作成します。テーブルパーティションキーとソートキー、およびインデックスソートキーのみが、インデックスに射影される属性です。

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName);

//ProvisionedThroughput
createTableRequest.setProvisionedThroughput(new
 ProvisionedThroughput().withReadCapacityUnits((long)5).withWriteCapacityUnits((long)5));

//AttributeDefinitions
ArrayList<AttributeDefinition> attributeDefinitions= new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Artist").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("SongTitle").withAttributeType("S"));
```

```
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("AlbumTitle").withAttributeType("S"));

createTableRequest.setAttributeDefinitions(attributeDefinitions);

//KeySchema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE));  //Sort
 key

createTableRequest.setKeySchema(tableKeySchema);

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("AlbumTitle").withKeyType(KeyType.RANGE));  //
Sort key

Projection projection = new Projection().withProjectionType(ProjectionType.INCLUDE);
ArrayList<String> nonKeyAttributes = new ArrayList<String>();
nonKeyAttributes.add("Genre");
nonKeyAttributes.add("Year");
projection.setNonKeyAttributes(nonKeyAttributes);

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()

 .withIndexName("AlbumTitleIndex").withKeySchema(indexKeySchema).withProjection(projection);

ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
 ArrayList<LocalSecondaryIndex>();
localSecondaryIndexes.add(localSecondaryIndex);
createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

DynamoDB がテーブルを作成し、テーブルのステータスを ACTIVE に設定するまで待機する必要があります。その後、テーブルへのデータ項目の入力を開始できます。

ローカルセカンダリインデックスを持つテーブルの説明

テーブルのローカルセカンダリインデックスに関する情報を取得するには、describeTable メソッドを使用します。インデックスごとに、名前、キースキーマ、および射影された属性にアクセスできます。

AWS SDK for Java ドキュメント API を使用して、テーブルのローカルセカンダリインデックス情報にアクセスするためのステップを次に示します。

1. DynamoDB クラスのインスタンスを作成します。
2. Table クラスのインスタンスを作成します。テーブル名を入力する必要があります。
3. Table オブジェクトの describeTable メソッドを呼び出します。

以下の Java コード例は、前述のステップの例です。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);

TableDescription tableDescription = table.describe();

List<LocalSecondaryIndexDescription> localSecondaryIndexes
    = tableDescription.getLocalSecondaryIndexes();

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

Iterator<LocalSecondaryIndexDescription> lsiIter = localSecondaryIndexes.iterator();
while (lsiIter.hasNext()) {

    LocalSecondaryIndexDescription lsiDescription = lsiIter.next();
    System.out.println("Info for index " + lsiDescription.getIndexName() + ":");
    Iterator<KeySchemaElement> kseIter = lsiDescription.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = lsiDescription.getProjection();
    System.out.println("\tThe projection type is: " + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: " +
 projection.getNonKeyAttributes());
    }
}
```

ローカルセカンダリインデックスのクエリ

テーブルに Query を実行するのとほぼ同じ方法で、ローカルセカンダリインデックスに対する Query オペレーションを使用することができます。インデックス名、インデックスソートキーのクエリ条件、および返す属性を指定する必要があります。この例では、インデックスは AlbumTitleIndex、インデックスソートキーは AlbumTitle です。

返される属性は、インデックスに射影された属性だけです。このクエリを変更して非キー属性を選択することもできますが、これには比較的コストのかかるテーブルフェッチアクティビティが必要です。テーブルのフェッチの詳細については、「属性の射影」を参照してください。

AWS SDK for Java ドキュメント API を使用してローカルセカンダリインデックスをクエリする手順を次に示します。

1. DynamoDB クラスのインスタンスを作成します。
2. Table クラスのインスタンスを作成します。テーブル名を入力する必要があります。
3. Index クラスのインスタンスを作成します。インデックス名を入力する必要があります。
4. Index クラスの query メソッドを呼び出します。

以下の Java コード例は、前述のステップの例です。

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";
```

```
Table table = dynamoDB.getTable(tableName);
Index index = table.getIndex("AlbumTitleIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"));

ItemCollection<QueryOutcome> items = index.query(spec);

Iterator<Item> itemsIter = items.iterator();

while (itemsIter.hasNext()) {
    Item item = itemsIter.next();
    System.out.println(item.toJSONPretty());
}
```

例: Java ドキュメント API を使用したローカルセカンダリインデックス

次の Java コード例は、Amazon DynamoDB でローカルセカンダリインデックスを操作する方法を示しています。この例では、パーティションキーを CustomerId として 、ソートキーを OrderId として CustomerOrders という名前のテーブルを作成しています。このテーブルには、次の 2 つのローカルセカンダリインデックスがあります。

- OrderCreationDateIndex – ソートキーは OrderCreationDate です。次の属性がインデックスに射影されます。
  - ProductCategory
  - ProductName
  - OrderStatus
  - ShipmentTrackingId
- IsOpenIndex – ソートキーは IsOpen です。すべてのテーブル属性がインデックスに射影されます。

CustomerOrders テーブルが作成されると、プログラムは顧客の注文を表すデータをテーブルにロードします。次に、ローカルセカンダリインデックスを使用してデータにクエリを実行します。最後に、プログラムは CustomerOrders テーブルを削除します。

以下の例をテストするための詳細な手順については、「Java コードの例」を参照してください。

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.PutItemOutcome;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ReturnConsumedCapacity;
import com.amazonaws.services.dynamodbv2.model.Select;

public class DocumentAPILocalSecondaryIndexExample {

        static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        static DynamoDB dynamoDB = new DynamoDB(client);

        public static String tableName = "CustomerOrders";

        public static void main(String[] args) throws Exception {

                createTable();
                loadData();

                query(null);
                query("IsOpenIndex");
```

```
               query("OrderCreationDateIndex");

               deleteTable(tableName);

       }

       public static void createTable() {

               CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                               .withProvisionedThroughput(
                                               new
ProvisionedThroughput().withReadCapacityUnits((long) 1)

.withWriteCapacityUnits((long) 1));

               // Attribute definitions for table partition and sort keys
               ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
               attributeDefinitions
                               .add(new
AttributeDefinition().withAttributeName("CustomerId").withAttributeType("S"));
               attributeDefinitions.add(new
AttributeDefinition().withAttributeName("OrderId").withAttributeType("N"));

               // Attribute definition for index primary key attributes
               attributeDefinitions
                               .add(new
AttributeDefinition().withAttributeName("OrderCreationDate")
                                               .withAttributeType("N"));
               attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IsOpen").withAttributeType("N"));

               createTableRequest.setAttributeDefinitions(attributeDefinitions);

               // Key schema for table
               ArrayList<KeySchemaElement> tableKeySchema = new
ArrayList<KeySchemaElement>();
               tableKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                              // key
               tableKeySchema.add(new 
KeySchemaElement().withAttributeName("OrderId").withKeyType(KeyType.RANGE)); // Sort

                            // key

               createTableRequest.setKeySchema(tableKeySchema);

               ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new 
ArrayList<LocalSecondaryIndex>();

               // OrderCreationDateIndex
               LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
                               .withIndexName("OrderCreationDateIndex");

               // Key schema for OrderCreationDateIndex
               ArrayList<KeySchemaElement> indexKeySchema = new 
ArrayList<KeySchemaElement>();
               indexKeySchema.add(new 
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); // 
Partition

                              // key
               indexKeySchema.add(new 
KeySchemaElement().withAttributeName("OrderCreationDate")
                               .withKeyType(KeyType.RANGE)); // Sort
                                                             // key

               orderCreationDateIndex.setKeySchema(indexKeySchema);

               // Projection (with list of projected attributes) for
               // OrderCreationDateIndex
               Projection projection = new 
Projection().withProjectionType(ProjectionType.INCLUDE);
               ArrayList<String> nonKeyAttributes = new ArrayList<String>();
               nonKeyAttributes.add("ProductCategory");
               nonKeyAttributes.add("ProductName");
               projection.setNonKeyAttributes(nonKeyAttributes);

               orderCreationDateIndex.setProjection(projection);

               localSecondaryIndexes.add(orderCreationDateIndex);

               // IsOpenIndex
```

```
                LocalSecondaryIndex isOpenIndex = new
LocalSecondaryIndex().withIndexName("IsOpenIndex");

                // Key schema for IsOpenIndex
                indexKeySchema = new ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                               // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("IsOpen").withKeyType(KeyType.RANGE)); // Sort

                           // key

                // Projection (all attributes) for IsOpenIndex
                projection = new Projection().withProjectionType(ProjectionType.ALL);

                isOpenIndex.setKeySchema(indexKeySchema);
                isOpenIndex.setProjection(projection);

                localSecondaryIndexes.add(isOpenIndex);

                // Add index definitions to CreateTable request
                createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

                System.out.println("Creating table " + tableName + "...");
                System.out.println(dynamoDB.createTable(createTableRequest));

                // Wait for table to become active
                System.out.println("Waiting for " + tableName + " to become
ACTIVE...");
                try {
                        Table table = dynamoDB.getTable(tableName);
                        table.waitForActive();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void query(String indexName) {

                Table table = dynamoDB.getTable(tableName);
```

```
 System.out.println("\n***************************************************************\n");
                System.out.println("Querying table " + tableName + "...");

                QuerySpec querySpec = new
 QuerySpec().withConsistentRead(true).withScanIndexForward(true)

 .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

                if (indexName == "IsOpenIndex") {

                        System.out.println("\nUsing index: '" + indexName + "': Bob's
 orders that are open.");
                        System.out.println("Only a user-specified list of attributes
 are returned\n");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid
 and IsOpen = :v_isopen")
                                        .withValueMap(new
 ValueMap().withString(":v_custid", "bob@example.com")
                                                        .withNumber(":v_isopen", 1));

                        querySpec.withProjectionExpression(
                                        "OrderCreationDate, ProductCategory,
 ProductName, OrderStatus");

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else if (indexName == "OrderCreationDateIndex") {
                        System.out.println("\nUsing index: '" + indexName
                                        + "': Bob's orders that were placed after
 01/31/2015.");
                        System.out.println("Only the projected attributes are returned
\n");
                        Index index = table.getIndex(indexName);
                        querySpec.withKeyConditionExpression(
                                        "CustomerId = :v_custid and OrderCreationDate
>= :v_orddate")
                                        .withValueMap(
                                                        new
ValueMap().withString(":v_custid", "bob@example.com")

.withNumber(":v_orddate",

20150131));

                        querySpec.withSelect(Select.ALL_PROJECTED_ATTRIBUTES);

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else {
                        System.out.println("\nNo index: All of Bob's orders, by
OrderId:\n");

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid")
                                        .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com"));

                        ItemCollection<QueryOutcome> items = table.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                }

        }

        public static void deleteTable(String tableName) {
                Table table = dynamoDB.getTable(tableName);
                System.out.println("Deleting table " + tableName + "...");
                table.delete();

                // Wait for table to be deleted
                System.out.println("Waiting for " + tableName + " to be deleted...");
                try {
                        table.waitForDelete();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void loadData() {

                Table table = dynamoDB.getTable(tableName);

                System.out.println("Loading data into table " + tableName + "...");

                Item item = new Item().withPrimaryKey("CustomerId",
 "alice@example.com").withNumber("OrderId", 1)
                                .withNumber("IsOpen",
 1).withNumber("OrderCreationDate", 20150101)
                                .withString("ProductCategory", "Book")
                                .withString("ProductName", "The Great Outdoors")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                PutItemOutcome putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
 "alice@example.com").withNumber("OrderId", 2)
                                .withNumber("IsOpen",
 1).withNumber("OrderCreationDate", 20150221)
                                .withString("ProductCategory", "Bike")
                                .withString("ProductName", "Super Mountain")
                                .withString("OrderStatus", "ORDER RECEIVED");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
 "alice@example.com").withNumber("OrderId", 3)
```

```
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150304).withString("ProductCategory", "Music")
                                .withString("ProductName", "A Quiet
Interlude").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "176493");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 1)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150111).withString("ProductCategory", "Movie")
                                .withString("ProductName", "Calm Before The Storm")
                                .withString("OrderStatus", "SHIPPING DELAY")
                                .withString("ShipmentTrackingId", "859323");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 2)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150124).withString("ProductCategory", "Music")
                                .withString("ProductName", "E-Z
Listening").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "756943");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 3)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150221).withString("ProductCategory", "Music")
                                .withString("ProductName", "Symphony
9").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "645193");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 4)
```

```
                               .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150222)
                               .withString("ProductCategory", "Hardware")
                               .withString("ProductName", "Extra Heavy Hammer")
                               .withString("OrderStatus", "PACKING ITEMS");
               // no ShipmentTrackingId attribute

               putItemOutcome = table.putItem(item);

               item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 5)
                               /* no IsOpen attribute */
                               .withNumber("OrderCreationDate",
20150309).withString("ProductCategory", "Book")
                               .withString("ProductName", "How To
Cook").withString("OrderStatus", "IN TRANSIT")
                               .withString("ShipmentTrackingId", "440185");

               putItemOutcome = table.putItem(item);

               item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 6)
                               // no IsOpen attribute
                               .withNumber("OrderCreationDate",
20150318).withString("ProductCategory", "Luggage")
                               .withString("ProductName", "Really Big
Suitcase").withString("OrderStatus", "DELIVERED")
                               .withString("ShipmentTrackingId", "893927");

               putItemOutcome = table.putItem(item);

               item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 7)
                               /* no IsOpen attribute */
                               .withNumber("OrderCreationDate",
20150324).withString("ProductCategory", "Golf")
                               .withString("ProductName", "PGA Pro
II").withString("OrderStatus", "OUT FOR DELIVERY")
                               .withString("ShipmentTrackingId", "383283");

               putItemOutcome = table.putItem(item);
               assert putItemOutcome != null;
       }
```

```
}
```

ローカルセカンダリインデックスの操作: .NET

トピック

- ローカルセカンダリインデックスを持つテーブルを作成する
- ローカルセカンダリインデックスを持つテーブルの説明
- ローカルセカンダリインデックスのクエリ
- 例: AWS SDK for .NET 低レベル API を使用したローカルセカンダリインデックス

AWS SDK for .NET 低レベル API を使用して、1 つ以上のローカルセカンダリインデックスを含む Amazon DynamoDB テーブルを作成し、テーブルのインデックスを記述し、インデックスを使用してクエリを実行できます。これらのオペレーションは、対応する低レベル DynamoDB API アクションにマッピングされます。詳細については、「.NET コード例」を参照してください。

以下に、.NET 低レベル API を使用したテーブルオペレーションの一般的な手順を示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. 対応するリクエストオブジェクトを作成して、オペレーションについて必要なパラメータとオプションパラメータを入力します。

   例えば、テーブルを作成するには CreateTableRequest オブジェクトを作成し、テーブルやインデックスにクエリを実行するには QueryRequest オブジェクトを作成します。
3. 前述のステップで作成したクライアントから提供された適切なメソッドを実行します。

ローカルセカンダリインデックスを持つテーブルを作成する

ローカルセカンダリインデックスは、テーブルの作成と同時に作成する必要があります。これを行うには、CreateTable を使用し、1 つ以上のローカルセカンダリインデックスの仕様を指定します。次の C# コード例では、ミュージックコレクション内の曲に関する情報を保持するためのテーブルを作成しています。パーティションキーは Artist で、ソートキーは SongTitle です。セカンダリインデックス AlbumTitleIndex は、アルバムタイトルによるクエリを容易にします。

.NET 低レベル API を使用して、ローカルセカンダリインデックスを含むテーブルを作成する手順を次に示します。

1. `AmazonDynamoDBClient` クラスのインスタンスを作成します。
2. リクエスト情報を指定する `CreateTableRequest` クラスのインスタンスを作成します。

   テーブル名、プライマリキー、およびプロビジョニングされたスループット値を指定する必要があります。ローカルセカンダリインデックスの場合は、インデックス名、インデックスソートキーの名前とデータ型、インデックスのキースキーマ、および属性射影を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、`CreateTable` メソッドを実行します。

以下の C# サンプルコードは、前述のステップの例です。このコードは、`AlbumTitle` 属性に関するセカンダリインデックスを持つテーブル (`Music`) を作成します。テーブルパーティションキーとソートキー、およびインデックスソートキーのみが、インデックスに射影される属性です。

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

CreateTableRequest createTableRequest = new CreateTableRequest()
{
    TableName = tableName
};

//ProvisionedThroughput
createTableRequest.ProvisionedThroughput = new ProvisionedThroughput()
{
    ReadCapacityUnits = (long)5,
    WriteCapacityUnits = (long)5
};

//AttributeDefinitions
List<AttributeDefinition> attributeDefinitions = new List<AttributeDefinition>();

attributeDefinitions.Add(new AttributeDefinition()
{
    AttributeName = "Artist",
    AttributeType = "S"
});

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "SongTitle",
     AttributeType = "S"
 });

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "AlbumTitle",
     AttributeType = "S"
 });

createTableRequest.AttributeDefinitions = attributeDefinitions;

//KeySchema
List<KeySchemaElement> tableKeySchema = new List<KeySchemaElement>();

tableKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
tableKeySchema.Add(new KeySchemaElement() { AttributeName = "SongTitle", KeyType =
 "RANGE" });  //Sort key

createTableRequest.KeySchema = tableKeySchema;

List<KeySchemaElement> indexKeySchema = new List<KeySchemaElement>();
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "AlbumTitle", KeyType =
 "RANGE" });  //Sort key

Projection projection = new Projection() { ProjectionType = "INCLUDE" };

List<string> nonKeyAttributes = new List<string>();
nonKeyAttributes.Add("Genre");
nonKeyAttributes.Add("Year");
projection.NonKeyAttributes = nonKeyAttributes;

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()
{
    IndexName = "AlbumTitleIndex",
    KeySchema = indexKeySchema,
    Projection = projection
};

List<LocalSecondaryIndex> localSecondaryIndexes = new List<LocalSecondaryIndex>();
localSecondaryIndexes.Add(localSecondaryIndex);
createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;
```

```
CreateTableResponse result = client.CreateTable(createTableRequest);
Console.WriteLine(result.CreateTableResult.TableDescription.TableName);
Console.WriteLine(result.CreateTableResult.TableDescription.TableStatus);
```

DynamoDB がテーブルを作成し、テーブルのステータスを ACTIVE に設定するまで待機する必要があります。その後、テーブルへのデータ項目の入力を開始できます。

ローカルセカンダリインデックスを持つテーブルの説明

テーブルのローカルセカンダリインデックスに関する情報を取得するには、DescribeTable API を使用します。インデックスごとに、名前、キースキーマ、および射影された属性にアクセスできます。

.NET 低レベル API を使用してテーブルのローカルセカンダリインデックス情報にアクセスするには次の手順に従います。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. リクエスト情報を指定する DescribeTableRequest クラスのインスタンスを作成します。テーブル名を入力する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、describeTable メソッドを実行します。
4.

以下の C# サンプルコードは、前述のステップの例です。

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest()
 { TableName = tableName });
List<LocalSecondaryIndexDescription> localSecondaryIndexes =
    response.DescribeTableResult.Table.LocalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.
foreach (LocalSecondaryIndexDescription lsiDescription in localSecondaryIndexes)
{
    Console.WriteLine("Info for index " + lsiDescription.IndexName + ":");
    foreach (KeySchemaElement kse in lsiDescription.KeySchema)
    {
        Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
    }

    Projection projection = lsiDescription.Projection;

    Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

    if (projection.ProjectionType.ToString().Equals("INCLUDE"))
    {
        Console.WriteLine("\t\tThe non-key projected attributes are:");

        foreach (String s in projection.NonKeyAttributes)
        {
            Console.WriteLine("\t\t" + s);
        }

    }
}
```

ローカルセカンダリインデックスのクエリ

テーブルに Query を実行するのとほぼ同じ方法で、ローカルセカンダリインデックスに対する Query を使用することができます。インデックス名、インデックスソートキーのクエリ条件、および返す属性を指定する必要があります。この例では、インデックスは AlbumTitleIndex、インデックスソートキーは AlbumTitle です。

返される属性は、インデックスに射影された属性だけです。このクエリを変更して非キー属性を選択することもできますが、これには比較的コストのかかるテーブルフェッチアクティビティが必要です。テーブルのフェッチの詳細については、「属性の射影」を参照してください。

.NET 低レベル API を使用して、ローカルセカンダリインデックスを含むテーブルにクエリを実行する手順を次に示します。

1. AmazonDynamoDBClient クラスのインスタンスを作成します。
2. リクエスト情報を指定する QueryRequest クラスのインスタンスを作成します。
3. リクエストオブジェクトをパラメータとして指定して、query メソッドを実行します。

以下の C# サンプルコードは、前述のステップの例です。

Example

```
QueryRequest queryRequest = new QueryRequest
{
    TableName = "Music",
    IndexName = "AlbumTitleIndex",
    Select = "ALL_ATTRIBUTES",
    ScanIndexForward = true,
    KeyConditionExpression = "Artist = :v_artist and AlbumTitle = :v_title",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":v_artist",new AttributeValue {S = "Acme Band"}},
        {":v_title",new AttributeValue {S = "Songs About Life"}}
    },
};

QueryResponse response = client.Query(queryRequest);

foreach (var attribs in response.Items)
{
    foreach (var attrib in attribs)
    {
        Console.WriteLine(attrib.Key + " ---> " + attrib.Value.S);
    }
    Console.WriteLine();
}
```

例: AWS SDK for .NET 低レベル API を使用したローカルセカンダリインデックス

次の C# コード例は、Amazon DynamoDB でローカルセカンダリインデックスを操作する方法を示しています。この例では、パーティションキーを `CustomerId` として 、ソートキーを `OrderId` として `CustomerOrders` という名前のテーブルを作成しています。このテーブルには、次の 2 つのローカルセカンダリインデックスがあります。

- `OrderCreationDateIndex` – ソートキーは `OrderCreationDate` です。次の属性がインデックスに射影されます。
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`

- IsOpenIndex – ソートキーは IsOpen です。すべてのテーブル属性がインデックスに射影されます。

CustomerOrders テーブルが作成されると、プログラムは顧客の注文を表すデータをテーブルにロードします。次に、ローカルセカンダリインデックスを使用してデータにクエリを実行します。最後に、プログラムは CustomerOrders テーブルを削除します。

以下の例をテストするための詳細な手順については、「.NET コード例」を参照してください。

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelLocalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "CustomerOrders";

        static void Main(string[] args)
        {
            try
            {
                CreateTable();
                LoadData();

                Query(null);
                Query("IsOpenIndex");
                Query("OrderCreationDateIndex");

                DeleteTable(tableName);

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateTable()
        {
            var createTableRequest =
                new CreateTableRequest()
                {
                    TableName = tableName,
                    ProvisionedThroughput =
                    new ProvisionedThroughput()
                    {
                        ReadCapacityUnits = (long)1,
                        WriteCapacityUnits = (long)1
                    }
                };

            var attributeDefinitions = new List<AttributeDefinition>()
        {
            // Attribute definitions for table primary key
            { new AttributeDefinition() {
                  AttributeName = "CustomerId", AttributeType = "S"
              } },
            { new AttributeDefinition() {
                  AttributeName = "OrderId", AttributeType = "N"
              } },
            // Attribute definitions for index primary key
            { new AttributeDefinition() {
                  AttributeName = "OrderCreationDate", AttributeType = "N"
              } },
            { new AttributeDefinition() {
                  AttributeName = "IsOpen", AttributeType = "N"
              }}
        };

            createTableRequest.AttributeDefinitions = attributeDefinitions;

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                  //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderId", KeyType = "RANGE"
              } }                                                //Sort key
        };

            createTableRequest.KeySchema = tableKeySchema;

            var localSecondaryIndexes = new List<LocalSecondaryIndex>();

            // OrderCreationDateIndex
            LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
            {
                IndexName = "OrderCreationDateIndex"
            };

            // Key schema for OrderCreationDateIndex
            var indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              } },                                                    //Partition key
            { new KeySchemaElement() {
                  AttributeName = "OrderCreationDate", KeyType = "RANGE"
              } }                                                            //Sort key
        };

            orderCreationDateIndex.KeySchema = indexKeySchema;

            // Projection (with list of projected attributes) for
            // OrderCreationDateIndex
            var projection = new Projection()
            {
                ProjectionType = "INCLUDE"
            };

            var nonKeyAttributes = new List<string>()
        {
            "ProductCategory",
            "ProductName"
        };
            projection.NonKeyAttributes = nonKeyAttributes;
            orderCreationDateIndex.Projection = projection;

            localSecondaryIndexes.Add(orderCreationDateIndex);

            // IsOpenIndex
            LocalSecondaryIndex isOpenIndex
                = new LocalSecondaryIndex()
                {
                    IndexName = "IsOpenIndex"
                };

            // Key schema for IsOpenIndex
            indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                  AttributeName = "CustomerId", KeyType = "HASH"
              }},                                                     //Partition key
            { new KeySchemaElement() {
                  AttributeName = "IsOpen", KeyType = "RANGE"
              }}                                                  //Sort key
        };

            // Projection (all attributes) for IsOpenIndex
            projection = new Projection()
            {
                ProjectionType = "ALL"
            };

            isOpenIndex.KeySchema = indexKeySchema;
            isOpenIndex.Projection = projection;

            localSecondaryIndexes.Add(isOpenIndex);

            // Add index definitions to CreateTable request
            createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);
            WaitUntilTableReady(tableName);
        }

        public static void Query(string indexName)
        {
```

```
Console.WriteLine("\n**********************************************************\n");
           Console.WriteLine("Querying table " + tableName + "...");

           QueryRequest queryRequest = new QueryRequest()
           {
               TableName = tableName,
               ConsistentRead = true,
               ScanIndexForward = true,
               ReturnConsumedCapacity = "TOTAL"
           };


           String keyConditionExpression = "CustomerId = :v_customerId";
           Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue> {
           {":v_customerId", new AttributeValue {
                S = "bob@example.com"
            }}
       };


           if (indexName == "IsOpenIndex")
           {
               Console.WriteLine("\nUsing index: '" + indexName
                         + "': Bob's orders that are open.");
               Console.WriteLine("Only a user-specified list of attributes are
returned\n");
               queryRequest.IndexName = indexName;

               keyConditionExpression += " and IsOpen = :v_isOpen";
               expressionAttributeValues.Add(":v_isOpen", new AttributeValue
               {
                   N = "1"
               });

               // ProjectionExpression
               queryRequest.ProjectionExpression = "OrderCreationDate,
ProductCategory, ProductName, OrderStatus";
           }
           else if (indexName == "OrderCreationDateIndex")
           {
               Console.WriteLine("\nUsing index: '" + indexName
                         + "': Bob's orders that were placed after 01/31/2013.");
               Console.WriteLine("Only the projected attributes are returned\n");
               queryRequest.IndexName = indexName;

               keyConditionExpression += " and OrderCreationDate > :v_Date";
               expressionAttributeValues.Add(":v_Date", new AttributeValue
               {
                   N = "20130131"
               });

               // Select
               queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
           }
           else
           {
               Console.WriteLine("\nNo index: All of Bob's orders, by OrderId:\n");
           }
           queryRequest.KeyConditionExpression = keyConditionExpression;
           queryRequest.ExpressionAttributeValues = expressionAttributeValues;

           var result = client.Query(queryRequest);
           var items = result.Items;
           foreach (var currentItem in items)
           {
               foreach (string attr in currentItem.Keys)
               {
                   if (attr == "OrderId" || attr == "IsOpen"
                       || attr == "OrderCreationDate")
                   {
                       Console.WriteLine(attr + "---> " + currentItem[attr].N);
                   }
                   else
                   {
                       Console.WriteLine(attr + "---> " + currentItem[attr].S);
                   }
               }
               Console.WriteLine();
           }
           Console.WriteLine("\nConsumed capacity: " +
result.ConsumedCapacity.CapacityUnits + "\n");
       }

       private static void DeleteTable(string tableName)
       {
           Console.WriteLine("Deleting table " + tableName + "...");
           client.DeleteTable(new DeleteTableRequest()
           {
               TableName = tableName
           });
           WaitForTableToBeDeleted(tableName);
       }

       public static void LoadData()
       {
           Console.WriteLine("Loading data into table " + tableName + "...");

           Dictionary<string, AttributeValue> item = new Dictionary<string,
AttributeValue>();

           item["CustomerId"] = new AttributeValue
           {
               S = "alice@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "1"
           };
           item["IsOpen"] = new AttributeValue
           {
               N = "1"
           };
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130101"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Book"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "The Great Outdoors"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "PACKING ITEMS"
           };
           /* no ShipmentTrackingId attribute */
           PutItemRequest putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "alice@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "2"
           };
           item["IsOpen"] = new AttributeValue
           {
               N = "1"
           };
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130221"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Bike"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "Super Mountain"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "ORDER RECEIVED"
           };
           /* no ShipmentTrackingId attribute */
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);
           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "alice@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "3"
           };
           /* no IsOpen attribute */
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130304"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Music"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "A Quiet Interlude"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "IN TRANSIT"
           };
           item["ShipmentTrackingId"] = new AttributeValue
           {
               S = "176493"
           };
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "bob@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "1"
           };
           /* no IsOpen attribute */
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130111"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Movie"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "Calm Before The Storm"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "SHIPPING DELAY"
           };
           item["ShipmentTrackingId"] = new AttributeValue
           {
               S = "859323"
           };
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "bob@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "2"
           };
           /* no IsOpen attribute */
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130124"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Music"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "E-Z Listening"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "DELIVERED"
           };
           item["ShipmentTrackingId"] = new AttributeValue
           {
               S = "756943"
           };
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "bob@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "3"
           };
           /* no IsOpen attribute */
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130221"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Music"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "Symphony 9"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "DELIVERED"
           };
           item["ShipmentTrackingId"] = new AttributeValue
           {
               S = "645193"
           };
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "bob@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "4"
           };
           item["IsOpen"] = new AttributeValue
           {
               N = "1"
           };
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130222"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Hardware"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "Extra Heavy Hammer"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "PACKING ITEMS"
           };
           /* no ShipmentTrackingId attribute */
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "bob@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "5"
           };
           /* no IsOpen attribute */
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130309"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Book"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "How To Cook"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "IN TRANSIT"
           };
           item["ShipmentTrackingId"] = new AttributeValue
           {
               S = "440185"
           };
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "bob@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "6"
           };
           /* no IsOpen attribute */
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130318"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Luggage"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "Really Big Suitcase"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "DELIVERED"
           };
           item["ShipmentTrackingId"] = new AttributeValue
           {
               S = "893927"
           };
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);

           item = new Dictionary<string, AttributeValue>();
           item["CustomerId"] = new AttributeValue
           {
               S = "bob@example.com"
           };
           item["OrderId"] = new AttributeValue
           {
               N = "7"
           };
           /* no IsOpen attribute */
           item["OrderCreationDate"] = new AttributeValue
           {
               N = "20130324"
           };
           item["ProductCategory"] = new AttributeValue
           {
               S = "Golf"
           };
           item["ProductName"] = new AttributeValue
           {
               S = "PGA Pro II"
           };
           item["OrderStatus"] = new AttributeValue
           {
               S = "OUT FOR DELIVERY"
           };
           item["ShipmentTrackingId"] = new AttributeValue
           {
               S = "383283"
           };
           putItemRequest = new PutItemRequest
           {
               TableName = tableName,
               Item = item,
               ReturnItemCollectionMetrics = "SIZE"
           };
           client.PutItem(putItemRequest);
       }

       private static void WaitUntilTableReady(string tableName)
       {
           string status = null;
           // Let us wait until table is created. Call DescribeTable.
           do
           {
               System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
               try
               {
                   var res = client.DescribeTable(new DescribeTableRequest
                   {
                       TableName = tableName
                   });

                   Console.WriteLine("Table name: {0}, status: {1}",
                             res.Table.TableName,
                             res.Table.TableStatus);
                   status = res.Table.TableStatus;
               }
               catch (ResourceNotFoundException)
               {
                   // DescribeTable is eventually consistent. So you might
                   // get resource not found. So we handle the potential exception.
               }
           } while (status != "ACTIVE");
       }

       private static void WaitForTableToBeDeleted(string tableName)
       {
           bool tablePresent = true;

           while (tablePresent)
           {
               System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
               try
               {
                   var res = client.DescribeTable(new DescribeTableRequest
                   {
                       TableName = tableName
                   });

                   Console.WriteLine("Table name: {0}, status: {1}",
                             res.Table.TableName,
                             res.Table.TableStatus);
               }
               catch (ResourceNotFoundException)
               {
                   tablePresent = false;
               }
           }
       }
   }
}
```

ローカルセカンダリインデックスの操作: AWS CLI

AWS CLI を使用して、1 つ以上のローカルセカンダリインデックスを含む Amazon DynamoDB テーブルを作成し、テーブルのインデックスを記述し、インデックスを使用してクエリを実行できます。

トピック

- ローカルセカンダリインデックスを持つテーブルを作成する
- ローカルセカンダリインデックスを持つテーブルの説明
- ローカルセカンダリインデックスのクエリ

ローカルセカンダリインデックスを持つテーブルを作成する

ローカルセカンダリインデックスは、テーブルの作成と同時に作成する必要があります。これを行うには、create-table パラメータを使用し、1 つ以上のローカルセカンダリインデックスの仕様を指定します。次の例では、ミュージックコレクション内の曲に関する情報を保持するためのテーブル (Music) を作成しています。パーティションキーは Artist で、ソートキーは SongTitle です。AlbumTitle 属性に関するセカンダリインデックス AlbumTitleIndex は、アルバムタイトルによるクエリを容易にします。

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S  \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[{\"IndexName\": \"AlbumTitleIndex\",
        \"KeySchema\":[{\"AttributeName\":\"Artist\",\"KeyType\":\"HASH\"},
                      {\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"RANGE\"}],
        \"Projection\":{\"ProjectionType\":\"INCLUDE\",  \"NonKeyAttributes\":[\"Genre
\", \"Year\"]}}]"
```

DynamoDB がテーブルを作成し、テーブルのステータスを ACTIVE に設定するまで待機する必要があります。その後、テーブルへのデータ項目の入力を開始できます。describe-table を使用して、テーブル作成のステータスを判断できます。

ローカルセカンダリインデックスを持つテーブルの説明

テーブルのローカルセカンダリインデックスに関する情報を取得するには、describe-table パラメータを使用します。インデックスごとに、名前、キースキーマ、および射影された属性にアクセスできます。

```
aws dynamodb describe-table --table-name Music
```

ローカルセカンダリインデックスのクエリ

テーブルに query を実行するのとほぼ同じ方法で、ローカルセカンダリインデックスに対する query オペレーションを使用することができます。インデックス名、インデックスソートキーのクエリ条件、および返す属性を指定する必要があります。この例では、インデックスは AlbumTitleIndex、インデックスソートキーは AlbumTitle です。

返される属性は、インデックスに射影された属性だけです。このクエリを変更して非キー属性を選択することもできますが、これには比較的コストのかかるテーブルフェッチアクティビティが必要です。テーブルのフェッチの詳細については、「属性の射影」を参照してください。

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v_artist and AlbumTitle = :v_title" \
    --expression-attribute-values  '{":v_artist":{"S":"Acme Band"},":v_title":
{"S":"Songs About Life"} }'
```

DynamoDB トランザクションで複雑なワークフローを管理する

Amazon DynamoDB Transactions は、テーブル内およびテーブル間の複数の項目を調整したり、変更しないといった、デベロッパーのエクスペリエンスを簡素化します。トランザクションによって DynamoDB に不可分性、一貫性、分離性、耐久性 (ACID) が実現されるため、アプリケーション内でのデータの精度を維持することができます。

DynamoDB トランザクション読み込み/書き込み API を使用し、1 つのオースオアナッシングオペレーションとして複数の項目の追加、更新、または削除が必要となる複雑なビジネスワークフローを管理できます。たとえば、ビデオゲームデベロッパーであれば、ゲーム内での項目交換やゲーム内購入の際に、プレイヤーのプロファイル更新の正確性を確保できます。

トランザクション書き込み API を使用して、複数の `Put`、`Update`、`Delete`、`ConditionCheck` の各アクションをグループ化できます。その後、アクションを単一の `TransactWriteItems` オペレーションとして送信できます。このオペレーションはユニットとして成功または失敗します。同じことが複数の `Get` アクションにも当てはまります。この場合、1 つの `TransactGetItems` オペレーションとしてグループ化し、送信できます。

DynamoDB テーブルのトランザクションを有効にするために、追加コストはかかりません。料金の支払いは、トランザクションの一部である読み込みまたは書き込みに対してのみ行われます。DynamoDB は、トランザクション内の各項目の基になっている 2 つの読み込みまたは書き込みを実行します。1 つはトランザクションの準備用で、もう 1 つはトランザクションのコミット用です。これらの基になっている 2 つの読み込み/書き込みオペレーションは、Amazon CloudWatch メトリクスに表示されます。

DynamoDB トランザクションを開始するには、最新の AWS SDK または AWS Command Line Interface (AWS CLI) をダウンロードします。その後、「DynamoDB トランザクションの例」に従います。

以下のセクションでは、トランザクション API の詳細についての概要とそれらを DynamoDB で使用する方法を示します。

トピック

- Amazon DynamoDB Transactions: 仕組み
- DynamoDB トランザクションでの IAM の使用
- DynamoDB トランザクションの例

Amazon DynamoDB Transactions: 仕組み

Amazon DynamoDB Transactions を使用すれば、複数のアクションをまとめてグループ化し、1 つのオールオアナッシングの `TransactWriteItems` または `TransactGetItems` オペレーションとして送信できます。以下のセクションでは、API オペレーション、容量管理、ベストプラクティス、DynamoDB でのトランザクション操作の使用に関する他の詳細について説明します。

トピック

- TransactWriteItems API
- TransactGetItems API
- DynamoDB トランザクションの分離レベル
- DynamoDB でのトランザクション競合の処理
- DynamoDB アクセラレーター (DAX) でのトランザクション API の使用
- トランザクションの容量管理
- トランザクションのベストプラクティス
- グローバルテーブルでのトランザクション API の使用
- DynamoDB トランザクションと AWSLabs トランザクションクライアントライブラリ

TransactWriteItems API

`TransactWriteItems` は、最大 100 の書き込みアクションを 1 つのオールオアナッシングオペレーションにグループ化する、同期的でべき等な書き込みオペレーションです。これらのアクションは、同じ AWS アカウントおよび同じリージョン内の 1 つ以上の DynamoDB テーブルにある最大 100 個の異なる項目をターゲットにすることができます。トランザクション内のアイテムの合計サイズは 4 MB を超えることはできません。すべて成功するかどれも成功しないかのどちらとなるように、アトミックに実行されます。

> Note
>
> - `TransactWriteItems` オペレーションは、含まれるすべてのアクションを正常に完了する必要があり、そうでない場合は変更がまったく行われないという点で `BatchWriteItem` オペレーションとは異なります。`BatchWriteItem` オペレーションでは、バッチ内の一部のアクションのみ成功し、他のアクションは成功しないことがあり得ます。
> - インデックスを使用してトランザクションを実行することはできません。

同じトランザクション内の複数のオペレーションが同じ項目をターゲットとすることはできません。たとえば、同じトランザクション内で同じ項目に対して `ConditionCheck` を実行し、`Update` アクションも実行することはできません。

以下のタイプのアクションをトランザクションに追加できます。

- Put — PutItem オペレーションを開始し、条件付きで、または条件をまったく指定せずに、新しい項目を作成するか、古い項目を新しい項目に置き換えます。
- Update — UpdateItem オペレーションを開始し、既存の項目の属性を編集するか、まだ存在しない場合は新しい項目をテーブルに追加します。条件付きまたは条件なしで既存の項目で属性を追加、削除、更新するには、このアクションを使用します。
- Delete — DeleteItem オペレーションを開始し、プライマリキーにより識別される 1 つの項目をテーブルで削除します。
- ConditionCheck — 項目が存在することを確認するか、項目の特定の属性の条件を確認します。

トランザクションが完了すると、そのトランザクションによって加えられた変更は、グローバルセカンダリインデックス (GSI)、ストリーム、バックアップに反映されます。反映は、即座に実行されないため、反映中の時点でテーブルがバックアップから復元されたり (RestoreTableFromBackup)、特定時点へエクスポートしたり (ExportTableToPointInTime) する場合、最近のトランザクション中に行われた変更の一部のみが含まれる可能性があります。

冪等性

TransactWriteItems 呼び出しを行ってリクエストが冪等であることを確認する場合、オプションでクライアントトークンを含めることができます。トランザクションを冪等にすると、接続のタイムアウトや他の接続問題に伴って同じオペレーションが複数回送信された場合に、アプリケーションエラーを防ぐことができます。

元の TransactWriteItems 呼び出しが成功した場合、同じクライアントトークンを持つ以降の TransactWriteItems 呼び出しは変更なしで正常に返ります。ReturnConsumedCapacity パラメータが設定されている場合、最初の TransactWriteItems 呼び出しは変更時に消費された書き込みキャパシティーユニットの数を返します。同じクライアントトークンを持つ以降の TransactWriteItems 呼び出しは、項目の読み取り時に消費された読み取りキャパシティユニットの数を返します。

冪等性について重要な点

- クライアントトークンは、それを使用するリクエストが完了してから 10 分間有効です。10 分後、同じクライアントトークンを使用するリクエストは新しいリクエストとして扱われます。10 分が経過してから、同じリクエストに同じクライアントトークンを再利用しないでください。
- 10 分間のべき等性期間内に同じクライアントトークンを使用してリクエストを繰り返すとき、他の一部のリクエストパラメータを変更した場合、DynamoDB は IdempotentParameterMismatch 例外を返します。

書き込みのエラー処理

以下の条件下では、書き込みトランザクションが成功しません。

- いずれかの条件式の条件が満たされていない場合。
- 同じ TransactWriteItems オペレーション内の複数のアクションが同じ項目をターゲットとしているために、トランザクション検証エラーが発生した場合。
- TransactWriteItems リクエストが、TransactWriteItems リクエスト内の 1 つ以上の項目に対する継続中の TransactWriteItems オペレーションと競合する場合。この場合、リクエストは TransactionCanceledException で失敗します。
- トランザクションを完了するプロビジョンドキャパシティーが足りない場合。
- 項目サイズが大きくなりすぎる (400 KB 超)、ローカルセカンダリインデックス (LSI) が大きくなりすぎる、またはトランザクションにより変更が加えられたために同様の検証エラーが発生した場合。
- 無効なデータ形式などのユーザーエラーがある場合。

TransactWriteItems オペレーションとの競合がどのように処理されるかについて詳しくは、「DynamoDB でのトランザクション競合の処理」を参照してください。

TransactGetItems API

TransactGetItems は、最大 100 個の Get アクションをまとめてグループ化する同期読み取りオペレーションです。これらのアクションは、同じ AWS アカウントおよびリージョン内の 1 つ以上の DynamoDB テーブルにある最大 100 個の異なる項目をターゲットにすることができます。トランザクション内の項目の合計サイズは 4 MB を超えることはできません。

Get アクションは、すべて成功するかすべて失敗するかのどちらとなるように、アトミックに実行されます。

- Get — GetItem オペレーションを開始し、指定されたプライマリキーを持つ項目の属性のセットを取得します。一致する項目が見つからない場合、Get はデータを返しません。

読み込みのエラー処理

以下の条件下では、読み取りトランザクションが成功しません。

- TransactGetItems リクエストが、TransactWriteItems リクエスト内の 1 つ以上の項目に対する継続中の TransactGetItems オペレーションと競合する場合。この場合、リクエストは TransactionCanceledException で失敗します。
- トランザクションを完了するプロビジョンドキャパシティーが足りない場合。
- 無効なデータ形式などのユーザーエラーがある場合。

TransactGetItems オペレーションとの競合がどのように処理されるかについて詳しくは、「DynamoDB でのトランザクション競合の処理」を参照してください。

DynamoDB トランザクションの分離レベル

トランザクションオペレーション (TransactWriteItems または TransactGetItems) と他のオペレーションの分離レベルは、次のとおりです。

SERIALIZABLE

直列化可能分離レベルでは、複数の同時オペレーションの結果は、前のオペレーションが完了するまでオペレーションが開始されない場合と同じになります。

以下のタイプのオペレーション間には、直列化可能分離があります。

- トランザクションオペレーションと標準書き込みオペレーション (PutItem、UpdateItem、または DeleteItem) の間。
- トランザクションオペレーションと標準読み取りオペレーション (GetItem) の間。
- TransactWriteItems オペレーションと TransactGetItems オペレーションの間。

トランザクションオペレーション間と BatchWriteItem オペレーション内の個々の標準書き込み間には直列化可能分離がありますが、トランザクションとユニットとしての BatchWriteItem オペレーションの間には直列化可能分離はありません。

同様に、トランザクションオペレーションと GetItems オペレーションの個別の BatchGetItem 間の分離レベルは直列化可能です。ただし、トランザクションとユニットとしての BatchGetItem オペレーション間の分離レベルはコミット済み読み取りです。

単一の GetItem リクエストは、TransactWriteItems リクエストの前または後に行う 2 つの方法のいずれかで、TransactWriteItems リクエストに関してシリアル化することができます。同時実行 TransactWriteItems リクエストのキーに対する複数の GetItem リクエストは、任意の順序で実行できるため、結果は読み込みがコミットされます。

たとえば、項目 A と項目 B の GetItem リクエストが、項目 A と項目 B の両方を変更する TransactWriteItems リクエストと同時に実行される場合、次の 4 つの可能性があります。

- 両方の GetItem リクエストは、TransactWriteItems リクエストの前に実行されます。
- 両方の GetItem リクエストは、TransactWriteItems リクエストの後に実行されます。
- 項目 A の GetItem リクエストは、TransactWriteItems リクエストの前に実行されます。項目 B の場合、GetItem は TransactWriteItems の後に実行されます。
- 項目 B の GetItem リクエストは、TransactWriteItems リクエストの前に実行されます。項目 A の場合、GetItem は TransactWriteItems の後に実行されます。

複数の GetItem リクエストにシリアル化可能な分離レベルが望ましい場合は、TransactGetItems を使用してください。

処理中に同じトランザクション書き込みリクエストの一部であった複数の項目に対して非トランザクション読み取りが行われた場合、一部の項目の新しい状態と他の項目の古い状態を読み取ることができる可能性があります。トランザクション書き込みリクエストに含まれていたすべての項目の新しい状態を読み取ることができるのは、トランザクション書き込みの応答が成功した場合のみです。

コミット済み読み取り

コミット済み読み取り分離により、読み取りオペレーションは常に項目のコミット済み値を返します。つまり、読み取りによって、最終的に成功しなかったトランザクション書き込みの状態を表すビューが項目に表示されることはありません。コミット済み読み取り分離では、読み取り操作の直後に項目の変更が防止されません。

分離レベルは、トランザクションオペレーションと、複数の標準読み取り (BatchGetItem、Query、または Scan) が関係する読み取りオペレーションの間ではコミット済み読み取りです。トランザクション書き込みにより BatchGetItem、Query、または Scan オペレーションの途中で項目が更新された場合、その後の読み取りオペレーションの部分では、新しくコミットされた値 (ConsistentRead) を使用) か、場合によってはそれ以前のコミット済み値 (結果整合性のある読み込み) を返します。

オペレーションの概要

以下の表は、トランザクションオペレーション (TransactWriteItems または TransactGetItems) と他のオペレーションの間の分離レベルをまとめたものです。

オペレーション	分離レベル
`DeleteItem`	直列化可能
`PutItem`	直列化可能
`UpdateItem`	直列化可能
`GetItem`	直列化可能
`BatchGetItem`	コミット済み読み取り*
`BatchWriteItem`	直列化不可*
`Query`	コミット済み読み取り
`Scan`	コミット済み読み取り
他のトランザクションオペレーション	直列化可能

アスタリスク (*) が付いたレベルは、ユニットとしてオペレーションに適用されます。ただし、これらのオペレーション内の個々のアクションの分離レベルは直列化可能です。

DynamoDB でのトランザクション競合の処理

トランザクション競合は、トランザクション内の項目に対する項目レベルの同時リクエスト中に発生する場合があります。トランザクション競合は、次のシナリオで発生する場合があります。

- 項目に対する `PutItem`、`UpdateItem`、または `DeleteItem` リクエストが、同じ項目を含む継続中の `TransactWriteItems` リクエストと競合する。
- `TransactWriteItems` リクエスト内の項目が、継続中の別の `TransactWriteItems` リクエストの一部である。
- `TransactGetItems` リクエスト内の項目が、継続中の `TransactWriteItems`、`BatchWriteItem`、`PutItem`、`UpdateItem`、または `DeleteItem` リクエストの一部である。

Note

- PutItem、UpdateItem、または DeleteItem リクエストが拒否された場合、リクエストは TransactionConflictException で失敗します。
- TransactWriteItems または TransactGetItems 内の項目レベルのリクエストが拒否された場合、リクエストは TransactionCanceledException で失敗します。そのリクエストが失敗した場合、AWS SDK はリクエストを再試行しません。

  AWS SDK for Java を使用している場合、例外には TransactItems リクエストパラメータの項目リスト通りに順序付けられた CancellationReasons のリストが含まれます。他の言語の場合、リストの文字列表現が例外のエラーメッセージに含まれます。
- 継続中の TransactWriteItems オペレーションまたは TransactGetItems オペレーションが同時 GetItem リクエストと競合している場合、両方のオペレーションが成功する可能性があります。

TransactionConflict CloudWatch メトリクスは、項目レベルのリクエストが失敗するたびに増分されます。

DynamoDB アクセラレーター (DAX) でのトランザクション API の使用

TransactWriteItems と TransactGetItems が、どちらも DynamoDB と同じ分離レベルで DynamoDB アクセラレーター (DAX) でサポートされています。

TransactWriteItems は DAX を介して書き込みます。DAX は DynamoDB に TransactWriteItems コールを渡し、応答を返します。書き込み後にキャッシュにデータを追加するために、DAX は、TransactWriteItems オペレーション内の各項目に対してバックグラウンドで TransactGetItems をコールします。これにより、追加の読み込み容量単位が消費されます。（詳しくは、トランザクションの容量管理 を参照してください）。この機能により、アプリケーションロジックをシンプルに保ち、トランザクション処理と非トランザクション処理の両方に DAX を使用できます。

TransactGetItems コールは、項目がローカルにキャッシュされることなく DAX を通過します。これは、DAX の強い整合性のある読み込み API と同じです。

トランザクションの容量管理

DynamoDB テーブルのトランザクションを有効にするために、追加コストはかかりません。料金の支払いは、トランザクションの一部である読み込みまたは書き込みに対してのみ行われます。DynamoDB は、トランザクション内の各項目の基になっている 2 つの読み込みまたは書き込みを実行します。1 つはトランザクションの準備用で、もう 1 つはトランザクションのコミット用です。基になっている 2 つの読み込み/書き込みオペレーションは、Amazon CloudWatch メトリクスに表示されます。

容量をテーブルにプロビジョニングするとき、トランザクション API により要求される追加の読み取りと書き込みを計画してください。たとえば、アプリケーションが 1 秒あたり 1 件のトランザクションを実行し、各トランザクションは 500 バイトの項目を 3 個テーブルに書き込むとします。各項目には、2 つの書き込みキャパシティーユニット (WCU) が必要です。1 つはトランザクションの準備用で、もう 1 つはトランザクションのコミット用です。したがって、テーブルには WCU を 6 個プロビジョニングする必要があります。

前の例で DynamoDB アクセラレーター (DAX) を使用していた場合、`TransactWriteItems` のコールで項目ごとに 2 つの読み込み容量単位 (RCU) も使用します。したがって、テーブルには追加の RCU を 6 個プロビジョニングする必要があります。

同様に、アプリケーションが 1 秒あたり 1 件の読み込みトランザクションを実行し、各トランザクションは 500 バイトの項目を 3 個テーブルで読み取る場合、読み込み容量単位 (RCU) を 6 個テーブルにプロビジョンする必要があります。各項目を読み取るには、2 つの RCU が必要です。1 つはトランザクションの準備用で、もう 1 つはトランザクションのコミット用です。

さらに、SDK のデフォルトの動作は、`TransactionInProgressException` 例外が発生した場合にトランザクションを再試行します。これらの再試行で消費される追加の読み込みキャパシティーユニット (RCU) を計画してください。同じことは、`ClientRequestToken` を使用して独自のコードでトランザクションを再試行しようとする場合に当てはまります。

トランザクションのベストプラクティス

DynamoDB トランザクションの使用時に推奨される以下の手法を検討してください。

- テーブルで自動スケーリングを有効にするか、トランザクションの項目ごとに 2 つの読み取りまたは書き込みオペレーションを実行するのに十分なスループット容量をプロビジョニングしたことを確認します。
- AWS によって提供された SDK を使用していない場合、`TransactWriteItems` 呼び出しを行うときに `ClientRequestToken` 属性を含め、リクエストがべき等となるようにします。

- 必要でない場合は、トランザクションにオペレーションをまとめてグループ化しないでください。たとえば、10 個のオペレーションを持つ 1 つのトランザクションを、アプリケーションの正確性を低下させずに複数のトランザクションに分割できる場合、トランザクションを分割することをお勧めします。トランザクションをシンプルにするとスループットが向上し、成功する可能性が高まります。
- 同じ項目を同時に更新する複数のトランザクションによって、トランザクションをキャンセルする競合が発生する可能性があります。そのような競合を最小限に抑えるため、データモデリングには以下の DynamoDB のベストプラクティスをお勧めします。
- 属性のセットが 1 つのトランザクションの一部として複数の項目間で頻繁に更新される場合、属性を 1 つの項目にグループ化し、トランザクションのスコープを減らすことを検討してください。
- データを大量に取り込むためにトランザクションを使用しないでください。一括書き込みには、`BatchWriteItem` の使用をお勧めします。

グローバルテーブルでのトランザクション API の使用

DynamoDB トランザクションに含まれる操作は、そのトランザクションが最初に実行されたリージョンでのみトランザクション可能であることが保証されます。トランザクション内で適用された変更がリージョン間でグローバルテーブルのレプリカにレプリケートされても、トランザクション性は保持されません。

DynamoDB トランザクションと AWSLabs トランザクションクライアントライブラリ

DynamoDB トランザクションには、AWSLabs トランザクションクライアントライブラリを置き換えるより費用対効率、堅牢性、パフォーマンスに優れた手段が用意されています。ネイティブのサーバー側トランザクション API を使用するようにアプリケーションを更新することをお勧めします。

DynamoDB トランザクションでの IAM の使用

AWS Identity and Access Management (IAM) を使用すると、トランザクションオペレーションが Amazon DynamoDB で実行可能なアクションを制限できます。DynamoDB での IAM ポリシーの詳細な使用については、DynamoDB のアイデンティティベースのポリシー を参照してください。

`Put`、`Update`、`Delete`、および `Get` アクションの権限は、基になる `PutItem`、`UpdateItem`、`DeleteItem`、および `GetItem` オペレーションに使用される権限により決定されます。`ConditionCheck` アクションの場合、IAM ポリシーで `dynamodb:ConditionCheck` 許可を使用できます。

以下に、DynamoDB トランザクションの設定に使用できる IAM ポリシーの例を示します。

例 1: トランザクションオペレーションを許可する

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

例 2: トランザクションオペレーションのみを許可する

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        }
    ]
}
```

例 3: トランザクションを使用しない読み込み/書き込みを許可し、トランザクションを使用する読み込み/書き込みをブロックする

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
             "Action": [
                 "dynamodb:PutItem",
                 "dynamodb:DeleteItem",
                 "dynamodb:GetItem",
                 "dynamodb:UpdateItem"
             ],
             "Resource": [
                 "arn:aws:dynamodb:*:*:table/table04"
             ]
         }
    ]
}
```

例 4: ConditionCheck の失敗時に情報が返されないようにする

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/table01",
            "Condition": {
                "StringEqualsIfExists": {
                    "dynamodb:ReturnValues": "NONE"
                }
            }
        }
    ]
}
```

DynamoDB トランザクションの例

Amazon DynamoDB Transactions が役に立つ状況の例として、このサンプルの Java アプリケーションをオンラインマーケットプレイスで検討してください。

アプリケーションには、バックエンドに 3 つの DynamoDB テーブルがあります。

- `Customers` — このテーブルには、マーケットプレイスの顧客に関する詳細が保存されます。プライマリキーは `CustomerId` 一意の識別子です。

- ProductCatalog — このテーブルには、マーケットプレイスで販売されている製品の価格や在庫状況などの詳細が保存されます。プライマリキーは ProductId 一意の識別子です。
- Orders — このテーブルには、マーケットプレイスからの注文に関する詳細が保存されます。プライマリキーは OrderId 一意の識別子です。

注文を作成する

次のコードスニペットは、DynamoDB トランザクションを使用して、注文の作成と処理に必要な複数のステップを調整する方法を示しています。単一のオールオアナッシングオペレーションを使用すると、トランザクションのいずれかの部分が失敗しても、トランザクション内のアクションは実行されず、変更も行われません。

この例では、customerId が 09e8e9c8-ec48 である顧客からの注文を設定します。次に、次の単純な注文処理ワークフローを使用して、単一のトランザクションとして実行します。

1. 顧客 ID が有効であることを確認します。
2. 製品が IN_STOCK であることを確認し、製品のステータスを SOLD に更新します。
3. 注文がまだ存在していないことを確認し、注文を作成します。

顧客を検証する

まず、customerId が 09e8e9c8-ec48 に等しい顧客が顧客テーブルに存在することを確認するアクションを定義します。

```
final String CUSTOMER_TABLE_NAME = "Customers";
final String CUSTOMER_PARTITION_KEY = "CustomerId";
final String customerId = "09e8e9c8-ec48";
final HashMap<String, AttributeValue> customerItemKey = new HashMap<>();
customerItemKey.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));

ConditionCheck checkCustomerValid = new ConditionCheck()
    .withTableName(CUSTOMER_TABLE_NAME)
    .withKey(customerItemKey)
    .withConditionExpression("attribute_exists(" + CUSTOMER_PARTITION_KEY + ")");
```

製品のステータスを更新する

次に、製品ステータスが現在 IN_STOCK に設定されている条件が true の場合に、製品ステータスを SOLD に更新するアクションを定義します。項目の製品ステータス属性が IN_STOCK に等しくな

かった場合、ReturnValuesOnConditionCheckFailure パラメータを設定すると項目が返されます。

```
final String PRODUCT_TABLE_NAME = "ProductCatalog";
final String PRODUCT_PARTITION_KEY = "ProductId";
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
expressionAttributeValues.put(":new_status", new AttributeValue("SOLD"));
expressionAttributeValues.put(":expected_status", new AttributeValue("IN_STOCK"));

Update markItemSold = new Update()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey)
    .withUpdateExpression("SET ProductStatus = :new_status")
    .withExpressionAttributeValues(expressionAttributeValues)
    .withConditionExpression("ProductStatus = :expected_status")

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD);
```

注文を作成する

最後に、OrderId の注文がまだ存在しない場合に限って、注文を作成します。

```
final String ORDER_PARTITION_KEY = "OrderId";
final String ORDER_TABLE_NAME = "Orders";

HashMap<String, AttributeValue> orderItem = new HashMap<>();
orderItem.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));
orderItem.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));
orderItem.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));
orderItem.put("OrderStatus", new AttributeValue("CONFIRMED"));
orderItem.put("OrderTotal", new AttributeValue("100"));

Put createOrder = new Put()
    .withTableName(ORDER_TABLE_NAME)
    .withItem(orderItem)

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD)
    .withConditionExpression("attribute_not_exists(" + ORDER_PARTITION_KEY + ")");
```

トランザクションを実行する

次の例は、単一のオールオアナッシングオペレーションとして以前に定義されたアクションを実行する方法を示しています。

```
    Collection<TransactWriteItem> actions = Arrays.asList(
        new TransactWriteItem().withConditionCheck(checkCustomerValid),
        new TransactWriteItem().withUpdate(markItemSold),
        new TransactWriteItem().withPut(createOrder));

    TransactWriteItemsRequest placeOrderTransaction = new TransactWriteItemsRequest()
        .withTransactItems(actions)
        .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

    // Run the transaction and process the result.
    try {
        client.transactWriteItems(placeOrderTransaction);
        System.out.println("Transaction Successful");

    } catch (ResourceNotFoundException rnf) {
        System.err.println("One of the table involved in the transaction is not found"
 + rnf.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println("Internal Server Error" + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.out.println("Transaction Canceled " + tce.getMessage());
    }
```

注文詳細の読み込み

次の例は、完了した注文を Orders テーブルと ProductCatalog テーブルでのトランザクションから読み込む方法を示しています。

```
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

HashMap<String, AttributeValue> orderKey = new HashMap<>();
orderKey.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));

Get readProductSold = new Get()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey);
Get readCreatedOrder = new Get()
```

```
    .withTableName(ORDER_TABLE_NAME)
    .withKey(orderKey);

Collection<TransactGetItem> getActions = Arrays.asList(
    new TransactGetItem().withGet(readProductSold),
    new TransactGetItem().withGet(readCreatedOrder));

TransactGetItemsRequest readCompletedOrder = new TransactGetItemsRequest()
    .withTransactItems(getActions)
    .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

// Run the transaction and process the result.
try {
    TransactGetItemsResult result = client.transactGetItems(readCompletedOrder);
    System.out.println(result.getResponses());
} catch (ResourceNotFoundException rnf) {
    System.err.println("One of the table involved in the transaction is not found" +
 rnf.getMessage());
} catch (InternalServerErrorException ise) {
    System.err.println("Internal Server Error" + ise.getMessage());
} catch (TransactionCanceledException tce) {
    System.err.println("Transaction Canceled" + tce.getMessage());
}
```

その他の例

- DynamoDBMapper からのトランザクションの使用

Amazon DynamoDB の変更データキャプチャ

多くのアプリケーションでは、DynamoDB テーブルに保存された項目の変更を、変更の発生時にキャプチャすることで利点を利用できます。以下に示しているのは、いくつかのユースケースの例です。

- 人気のモバイルアプリケーションは、1 秒あたり数千件の更新速度で、DynamoDB テーブルのデータを変更します。別のアプリケーションは、これらの更新に関するデータをキャプチャして保存し、モバイルアプリの使用状況メトリクスをほぼリアルタイムで提供します。
- 金融アプリケーションは、DynamoDB テーブル内の株式市場データを変更します。並行して実行されるさまざまなアプリケーションは、これらの変化をリアルタイムで追跡し、リスクのある価値を計算し、株価の動きに基づいてポートフォリオを自動的にリバランスします。

- 輸送車両や産業機器のセンサーは、DynamoDB テーブルにデータを送信します。さまざまなアプリケーションがパフォーマンスをモニタリングし、問題が検出されたときにメッセージングアラートを送信し、機械学習アルゴリズムを適用して潜在的な欠陥を予測し、データを圧縮して Amazon Simple Storage Service (Amazon S3) にアーカイブします。
- アプリケーションは、友人の 1 人が新しい画像をアップロードするとすぐに、グループ内のすべての友人のモバイルデバイスに通知を自動送信します。
- 新しいお客様がデータを DynamoDB テーブルに追加します。このイベントにより、新しいお客様にようこそメールを送信する別のアプリケーションが起動されます。

DynamoDB は、項目レベルの変更データキャプチャレコードのストリーミングをほぼリアルタイムでサポートします。これらのストリーミングを使用し、内容に基づいてアクションを実行するアプリケーションを構築できます。

次の動画では、データキャプチャ変更の概念を紹介します。

テーブル容量モード

トピック

変更データキャプチャのストリーミングオプション

DynamoDB には、変更データキャプチャ用の 2 つのストリーミングモデルがあります。DynamoDB 用 Kinesis Data Streams と DynamoDB Streams です。

アプリケーションに適したソリューションを選択しやすくするために、次の表に、各ストリーミングモデルの特徴をまとめてあります。

プロパティ	DynamoDB 用 Kinesis Data Streams	DynamoDB Streams
データ保持期間	1 年まで。	24 時間。
Kinesis Client Library (KCL) サポート	KCL バージョン 1.X および 2.X サポート。	KCL バージョン 1.X サポート。

プロパティ	DynamoDB 用 Kinesis Data Streams	DynamoDB Streams
コンシューマー数	シャードごとに最大 5 つの同時コンシューマー、またはファンアウトが強化されたシャードごとに最大 20 の同時コンシューマー。	シャードごとに最大 2 つの同時コンシューマー。
スループットクォータ	無制限。	DynamoDB テーブルと AWS リージョンによるスループットクォータの対象となります。
レコードの配信モデル	GetRecords を使用して HTTP 経由でモデルをプルし、ファンアウトを強化すると、Kinesis Data Streams は SubscribeToShard を使用して HTTP/2 経由でレコードをプッシュします。	GetRecords を使用した HTTP 経由のプルモデル。
レコードの順序	各ストリーミングレコードのタイムスタンプ属性を使用して、DynamoDB テーブルで変更が発生した実際の順序を特定できます。	DynamoDB テーブルで変更された各項目について、ストリーミングレコードは項目に対する実際の変更と同じ順序で出現します。
重複レコード	重複レコードがストリーミングに表示される場合があります。	重複レコードがストリーミングに表示されません。

プロパティ	DynamoDB 用 Kinesis Data Streams	DynamoDB Streams
ストリーミング処理オプション	AWS Lambda、Amazon Managed Service for Apache Flink。Kinesis Data Firehose、または AWS Glue ストリーミング ETL を使用してストリームレコードを処理します。	AWS Lambda、DynamoDB Streams Kinesis adapter を使用してストリームレコードを処理します。
耐久性	アベイラビリティーゾーンにより、中断することなく自動的にフェイルオーバーできます。	アベイラビリティーゾーンにより、中断することなく自動的にフェイルオーバーできます。

同じ DynamoDB テーブルで両方のストリーミングモデルを有効にすることができます。

次の動画では、これら 2 つのオプションの違いを詳しく説明しています。

DynamoDB ストリームと Kinesis データストリームの比較

Kinesis Data Streams を使用して DynamoDB への変更をキャプチャする。

Amazon Kinesis Data Streams を使用して Amazon DynamoDB への変更をキャプチャできます。

Kinesis Data Streams は、DynamoDB テーブルの項目レベルの変更をキャプチャーし、それらを Kinesis Data Streams にレプリケートします。アプリケーションは このストリームにアクセスして、項目レベルの変更をほぼリアルタイムで表示できます。1 時間あたりテラバイトのデータを継続的に取り込み保存できます。より長いデータ保持時間を利用し、強化されたファンアウト機能により、2 つ以上のダウンストリームアプリケーションに同時にアクセスできます。その他のメリットには、追加の監査とセキュリティの透明性が含まれます。

Kinesis Data Streams を使用すると、Amazon Data Firehose および Amazon Managed Service for Apache Flink にもアクセスできます。これらのサービスは、リアルタイムでのダッシュボードの強化、アラートの生成、動的な料金設定、広告の実装、高度なデータ分析および機械学習アルゴリズムを実装するアプリケーションの構築に役立ちます。

> Note
>
> DynamoDB で Kinesis データストリームを使用すると、データストリームに対する Kinesis Data Streams 料金とソーステーブルに対する DynamoDB 料金の両方が適用されます。

Kinesis Data Streams の DynamoDB との連携について

DynamoDB テーブルで Kinesis データストリームが有効になっている場合、そのテーブルは、テーブルのデータに対するすべての変更をキャプチャしたデータレコードを送信します。このデータレコードは以下を含みます。

- 項目が最後に作成、更新、または削除された特定の時刻
- その項目のプライマリキー
- 変更前のレコードのスナップショット
- 変更後のレコードのスナップショット

これらのデータレコードは、ほぼリアルタイムに取り込まれ、公開されます。これらが Kinesis データストリームに書き込まれた後は、他のレコードと同じように読み込むことができます。Kinesis Client Library の使用、AWS Lambda の使用、Kinesis Data Streams API の呼び出し、およびその他接続サービスの利用が可能です。詳細については、Amazon Kinesis Data Streams デベロッパーガイドのAmazon Kinesis Data Streams からのデータの読み込みを参照してください。

これらのデータへの変更は、非同期的にキャプチャーされます。Kinesis は、ストリーミング元のテーブルに対するパフォーマンスに影響を与えません。Kinesis データストリームに保存されているストリームレコードも、保存時に暗号化されます。詳細については、「Amazon Kinesis Data Streams のデータ保護」を参照してください。

Kinesis データストリームレコードは、項目の変更が発生した順序とは異なる順序で表示される場合があります。同じ項目の通知がストリームに複数回表示される場合もあります。`ApproximateCreationDateTime` 属性をチェックして、項目の変更が発生した順序を特定したり、重複するレコードを特定したりできます。

Kinesis データストリームを DynamoDB テーブルのストリーミング先として有効にすると、`ApproximateCreationDateTime` 値の精度をミリ秒またはマイクロ秒単位で設定できます。デフォルトでは、`ApproximateCreationDateTime` は変更の時刻をミリ秒単位で示します。さらに、アクティブなストリーミング先でこの値を変更できます。このような変更後、Kinesis に書き込

まれるストリームレコードは、希望する精度の ApproximateCreationDateTime 値を持つようになります。

DynamoDB に書き込むバイナリ値は、base64 エンコード形式でエンコードする必要があります。ただし、データレコードを Kinesis データストリームに書き込む場合、これらのエンコードされたバイナリ値は、再度 base64 エンコーディングでエンコードされます。これらのレコードを Kinesis データストリームから読み取る場合、未加工のバイナリ値を取得するには、アプリケーションでこれらの値を 2 回デコードする必要があります。

DynamoDB では、Kinesis Data Streams の使用に対して変更データキャプチャ単位で課金されます。単一の項目あたり 1 KB の変更が、1 つの変更データキャプチャ単位としてカウントされます。各項目で変更したキロバイト数は、書き込み操作のキャパシティーユニット消費量と同じロジックを使用して、ストリームに書き込まれた項目の「前の」イメージと「後の」イメージの大きい方で計算されます。DynamoDB オンデマンドモードの動作と同様に、変更データキャプチャ単位のキャパシティースループットをプロビジョニングする必要はありません。

DynamoDB テーブルの Kinesis データストリームを有効にする

AWS Management Console、AWS SDK、または AWS Command Line Interface (AWS CLI) を使用して、既存の DynamoDB テーブルから Kinesis へのストリーミングを有効または無効にすることができます。

- テーブルと同じ AWS アカウントと AWS リージョンでのみ、DynamoDB から Kinesis Data Streams にデータをストリーミングできます。
- DynamoDB テーブルからのデータを 1 つの Kinesis データストリームにだけストリーミングできます。

DynamoDB テーブルの Kinesis Data Streams 送信先に変更を加える

デフォルトでは、すべての Kinesis データストリームレコードには ApproximateCreationDateTime 属性が含まれます。この属性は、各レコードが作成されたおおよその時刻のタイムスタンプをミリ秒単位で表します。この値の精度は、https://console.aws.amazon.com/kinesis、SDK、または AWS CLI を使用して変更できます。

Amazon DynamoDB 用 Kinesis Data Streams の開始方法

このセクションでは、Amazon DynamoDB コンソール、AWS Command Line Interface (AWS CLI)、および API を活用して Amazon DynamoDB 用の Amazon Kinesis Data Streams テーブルを使用する方法について説明します。

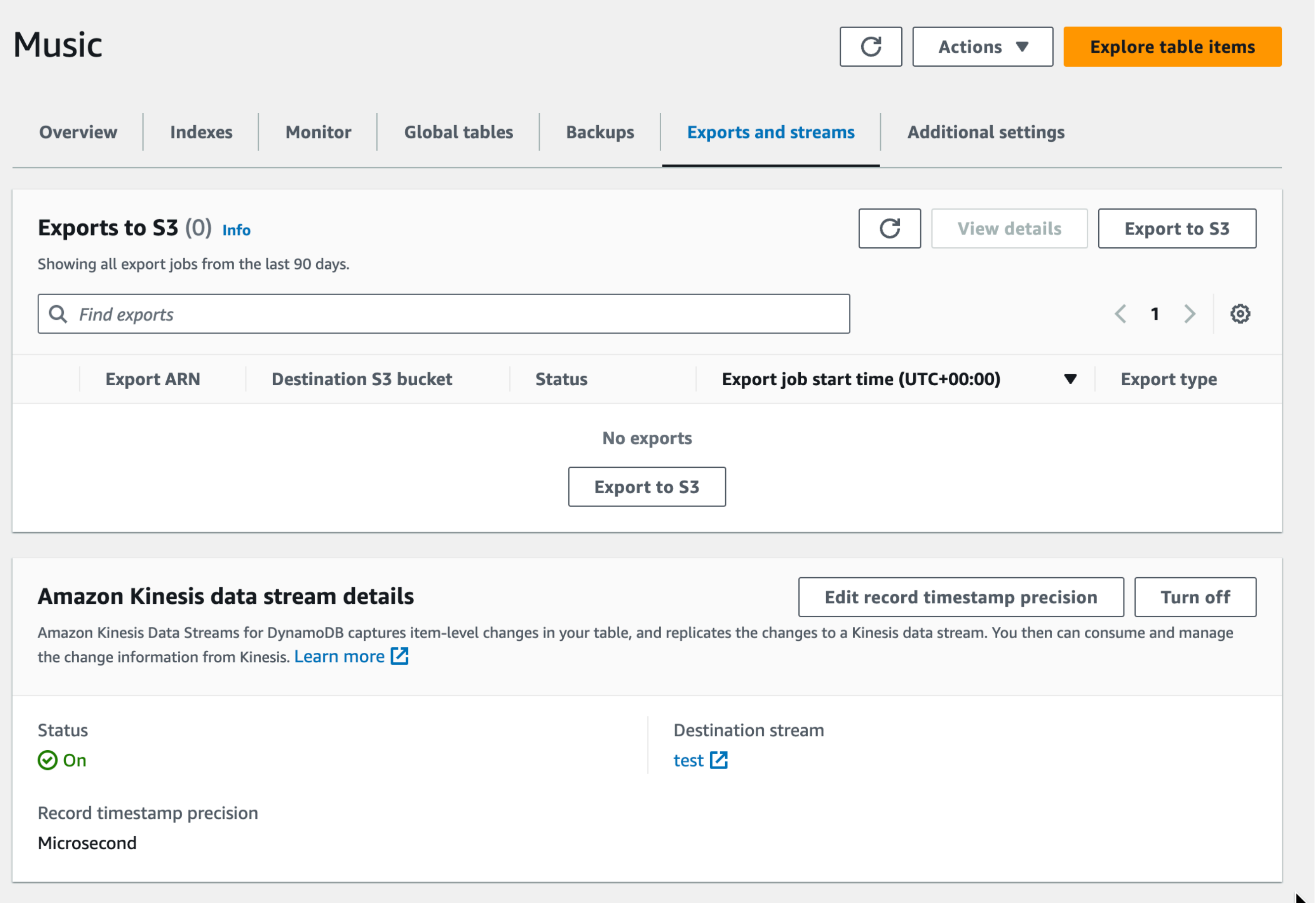
これらの例はすべて、DynamoDB の使用開始チュートリアルの一部として作成された `Music` DynamoDB テーブルを使用しています。

コンシューマーを構築し、Kinesis データストリームを他の AWS のサービスに接続する詳細方法については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Kinesis Data Streams からのデータの読み込み」を参照してください。

> **Note**
>
> KDS シャードを初めて使用するときは、使用パターンに合わせてシャードをスケールアップまたはスケールダウンするように設定することをお勧めします。使用パターンに関するデータをさらに蓄積したら、それに合わせてストリーム内のシャードを調整できます。

Console

1. AWS Management Console にサインインし、Kinesis コンソール (https://console.aws.amazon.com/kinesis/) を開きます。
2. [Create data stream (データストリーミングの作成)] を選択し、指示に従って `samplestream` というストリーミングを作成します。
3. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
4. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
5. [Music] テーブルを選択します。
6. [エクスポートとストリーム] タブを選択します。



Showing all export jobs from the last 90 days.

Amazon Kinesis Data Streams for DynamoDB captures item-level changes in your table, and replicates the changes to a Kinesis data stream. You then can consume and manage the change information from Kinesis. Learn more

Status
On

Destination stream
test

Record timestamp precision
Microsecond

7. (オプション) [Amazon Kinesis データストリームの詳細] で、レコードのタイムスタンプの精度をマイクロ秒 (デフォルト) からミリ秒に変更できます。

8. ドロップダウンリストから samplestream を選択します。

9. [オンにする] ボタンを選択します。

AWS CLI

1. create-stream コマンドを使用して、`samplestream` という名前の Kinesis データストリームを作成します。

```
aws kinesis create-stream --stream-name samplestream --shard-count 3
```

   Kinesis データストリームのシャード数を設定する前に「Kinesis Data Streams のシャード管理に関する考慮事項」を参照してください。

2. Kinesis ストリームがアクティブで、使用できる状態になっていることを確認するには、describe-stream コマンドを使用します。

```
aws kinesis describe-stream --stream-name samplestream
```

3. DynamoDB `enable-kinesis-streaming-destination` コマンドを使用して、DynamoDB テーブルで Kinesis ストリーミングを有効にします。`stream-arn` の値を、前のステップの `describe-stream` によって返された値で置き換えます。オプションで、各レコードに返されるタイムスタンプ値の精度をより細かく (マイクロ秒) したストリーミングを有効にします。

   マイクロ秒のタイムスタンプ精度でのストリーミングを有効にします。

   ```
   aws dynamodb enable-kinesis-streaming-destination \
     --table-name Music \
     --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
     --enable-kinesis-streaming-configuration
    ApproximateCreationDateTimePrecision=MICROSECOND
   ```

   または、デフォルトのタイムスタンプ精度 (ミリ秒) でのストリーミングを有効にします。

   ```
   aws dynamodb enable-kinesis-streaming-destination \
     --table-name Music \
     --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
   ```

4. DynamoDB `describe-kinesis-streaming-destination` コマンドを使用して、テーブルで Kinesis ストリーミングがアクティブかどうかを確認します。

   ```
   aws dynamodb describe-kinesis-streaming-destination --table-name Music
   ```

5. 「DynamoDB デベロッパーガイド」の説明通りに、`put-item` コマンドを使用して DynamoDB テーブルにデータを書き込みます。

   ```
   aws dynamodb put-item \
       --table-name Music  \
       --item \
           '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me
    Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

   aws dynamodb put-item \
       --table-name Music \
       --item \
   ```

```
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"} }'
```

6. Kinesis get-records CLI コマンドを使用して、Kinesis ストリームコンテンツを取得します。次に、以下のコードスニペットを使用して、ストリーミングコンテンツを逆シリアル化します。

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we want to fetch the value
     * of this attribute from the new item image. The following code fetches
 this value.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Java

1. 「Kinesis Data Streams デベロッパーガイド」の指示に従って、Java を使用し `samplestream` という名前の Kinesis データストリームを作成します。

   Kinesis データストリームのシャード数を設定する前に「Kinesis Data Streams のシャード管理に関する考慮事項」を参照してください。

2. 次のコードスニペットを使用して、DynamoDB テーブルで Kinesis ストリーミングを有効にします。オプションで、各レコードに返されるタイムスタンプ値の精度をより細かく (マイクロ秒) したストリーミングを有効にします。

   マイクロ秒のタイムスタンプ精度でのストリーミングを有効にします。

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

   または、デフォルトのタイムスタンプ精度 (ミリ秒) でのストリーミングを有効にします。

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. 「Kinesis Data Streams デベロッパーガイド」の指示に従って、作成したデータストリームから読み込みます。

4. 次のコードスニペットを使用して、ストリームコンテンツを逆シリアル化します。

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we wanted to fetch the value
     * of this attribute from the new item image, the below code would fetch
 this.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

アクティブな Amazon Kinesis データストリームに変更を加える

このセクションでは、コンソール、AWS CLI、API を使用して DynamoDB 用 Kinesis Data Streams のセットアップを変更する方法について説明します。

AWS Management Console

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. テーブルに移動します。
3. [エクスポートおよびストリーム] タブを選択します。

AWS CLI

1. `describe-kinesis-streaming-destination` を呼び出して、ストリームが `ACTIVE` であることを確認します。
2. 次の例のように、`UpdateKinesisStreamingDestination` を呼び出します。

   ```
   aws dynamodb update-kinesis-streaming-destination --table-name
    enable_test_table --stream-arn arn:aws:kinesis:us-east-1:12345678901:stream/
   enable_test_stream --update-kinesis-streaming-configuration
    ApproximateCreationDateTimePrecision=MICROSECOND
   ```
3. `describe-kinesis-streaming-destination` を呼び出して、ストリームが `UPDATING` であることを確認します。
4. ストリーミングステータスが再び `ACTIVE` になるまで `describe-kinesis-streaming-destination` を定期的に呼び出します。タイムスタンプの精度の更新が有効になるまで、最大 5 分かかることがあります。このステータスが更新されると、更新が完了したことを表し、新しい精度値が今後のレコードに適用されます。
5. `putItem` を使用してテーブルに書き込みます。
6. Kinesis `get-records` コマンドを使用して、Kinesis ストリームコンテンツを取得します。
7. 書き込みの `ApproximateCreationDateTime` の精度が希望どおりであることを確認します。

Java API

1. `UpdateKinesisStreamingDestination` リクエストと `UpdateKinesisStreamingDestination` レスポンスを構成するコードスニペットを提供します。

2. DescribeKinesisStreamingDestination リクエストと DescribeKinesisStreamingDestination response を構成するコードスニペットを提供します。
3. ストリーミングのステータスが、更新が完了し、将来のレコードに新しい精度値が適用されることを示す ACTIVE に戻るまで、describe-kinesis-streaming-destination を定期的に呼び出します。
4. テーブルへの書き込みを実行します。
5. ストリームから読み取り、ストリームコンテンツを逆シリアル化します。
6. 書き込みの ApproximateCreationDateTime の精度が希望どおりであることを確認します。

DynamoDB での Kinesis Data Streams を使用したシャードの設定および変更データキャプチャのモニタリング

Kinesis Data Streams のシャード管理に関する考慮事項

Kinesis データストリームは、シャードでスループットをカウントします。Amazon Kinesis Data Streams では、データストリームにオンデマンドモードとプロビジョンドモードのどちらかを選択できます。

DynamoDB の書き込みワークロードが大きく変動しやすく予測不可能な場合は、Kinesis データストリームにオンデマンドモードを使用することをお勧めします。オンデマンドモードでは、Kinesis Data Streams が必要なスループットを提供するためにシャードを自動的に管理するため、キャパシティプランニングは必要ありません。

予測可能なワークロードの場合は、Kinesis データストリームにプロビジョニングモードを使用できます。プロビジョンドモードでは、DynamoDB からの変更データ取得レコードを格納するためのデータストリームのシャード数を指定する必要があります。Kinesis データストリームが DynamoDB テーブルをサポートするために必要なシャードの数を決定するには、次の入力値が必要です。

- DynamoDB テーブルのレコードの平均サイズ (バイト単位、average_record_size_in_bytes)。
- DynamoDB テーブルで実行される 1 秒あたりの書き込み操作の最大数。これには、アプリケーションで実行される作成、削除、更新操作のほか、Time to Live で生成された削除操作 (write_throughput) などの自動生成された操作も含まれます。
- テーブルに対して実行する作成操作や削除操作と比較した、更新操作と上書き操作の割合 (percentage_of_updates)。更新操作と上書き操作では、変更された項目の古いイメージと

新しいイメージの両方がストリームにレプリケートされることに留意してください。これにより、DynamoDB 項目のサイズが 2 倍になります。

Kinesis データストリームに必要なシャードの数 (number_of_shards) を計算するには、入力値を以下の式にあてはめます。

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

例えば、書き込み操作の最大スループットが 1,040 回/秒で (write_throughput)、平均レコードサイズが 800 バイト (average_record_size_in_bytes) であるとします。 この書き込み操作の 25% が更新操作 (percentage_of_updates) である場合、DynamoDB ストリーミングスループットに対応するために 2 つのシャード (number_of_shards) が必要になります。

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

式を使用して Kinesis データストリームのプロビジョニングモードで必要なシャード数を計算する前に、次の点を考慮してください。

- この式では、DynamoDB 変更データレコードを格納するのに必要なシャードの数を見積もることができます。これは、追加の Kinesis データストリームのコンシューマーをサポートするために必要なシャードの数など、Kinesis データストリームで必要なシャードの総数を表すものではありません。
- ピークスループットを処理するようにデータストリームを設定しないと、プロビジョンドモードで読み取りおよび書き込みのスループットの例外が発生する可能性があります。この場合、データトラフィックに対応するようにデータストリームを手動でスケーリングする必要があります。
- この式では、変更ログデータレコードを Kinesis Data Stream にストリーミングする前に DynamoDB によって生成される追加の肥大化を考慮に入れています。

Kinesis データストリームのキャパシティモードの詳細については、「データストリームのキャパシティモードの選択」を参照してください。さまざまなキャパシティモード間の料金の違いについて詳しくは、「Amazon Kinesis Data Streams の料金」を参照してください。

Kinesis Data Streams を使用した変更データキャプチャのモニタリング

DynamoDB には、Kinesis への変更データキャプチャのレプリケーションをモニタリングするのに役立つ複数の Amazon CloudWatch メトリクスが用意されています。CloudWatch メトリクスの詳細な一覧については、「DynamoDB のメトリクスとディメンション」を参照してください。

ストリームに十分な容量があるかどうかを判断する場合は、ストリームの有効化時と実稼働時の両方で次の項目をモニタリングすることをお勧めします。

- `ThrottledPutRecordCount`: Kinesis データストリームのキャパシティが不足しているために、Kinesis データストリームによってスロットリングされたレコードの数。例外的な使用量のピーク時にスロットリングが発生する可能性がありますが、`ThrottledPutRecordCount` は可能な限り低く保つ必要があります。DynamoDB は、スロットリングされたレコードを Kinesis データストリームに再送信しますが、これによりレプリケーションのレイテンシーが高くなる可能性があります。

  過剰で定期的なスロットリングが発生した場合は、テーブルで観測された書き込みスループットに比例して Kinesis ストリーミングシャードの数を増やす必要があります。Kinesis Data Streams のサイズ決定の詳細については、「Kinesis Data Streams の初期サイズの決定」を参照してください。

- `AgeOfOldestUnreplicatedRecord`: Kinesis Data Streams にまだレプリケートされていない最も古い項目レベルの変更からの経過時間が DynamoDB テーブルに表示されました。通常のオペレーションでは、`AgeOfOldestUnreplicatedRecord` はミリ秒単位で順序を指定しなければなりません。この数字は、カスタマー管理設定上の選択が原因で失敗したレプリケーションの試行回数に基づいて増加します。

  `AgeOfOldestUnreplicatedRecord` メトリクスが 168 時間を超えると、DynamoDB テーブルから Kinesis データストリームへの項目レベルの変更のレプリケーションは自動的に無効になります。

  カスタマー管理設定でレプリケーション試行の失敗の原因になる例として、Kinesis データストリームキャパシティーのプロビジョニングが不足していたために過剰なスロットリングにつながった場合や、Kinesis データストリームのアクセスポリシーを手動で更新したために DynamoDB がデータストリームにデータを追加できなくなった場合が挙げられます。このメトリクスを可能な限り低く保つために、Kinesis データストリームキャパシティーを十分にプロビジョニングし、DynamoDB のアクセス許可が変更されていないことを確認する必要があります。

- `FailedToReplicateRecordCount`: DynamoDB が Kinesis データストリームにレプリケートできなかったレコードの数。34 KB を超える特定の項目はサイズが拡張されて、Kinesis Data

Streams の項目サイズ制限 1MB を超えるデータレコードが変更される場合があります。このサイズの拡張は、34 KB を超えるこれらの項目に多数のブール値や空の属性値が含まれる場合に発生します。ブール値と空の属性値は、DynamoDB に 1 バイトで格納されますが、Kinesis Data Streams レプリケーションで標準 JSON を使用してシリアル化すると、最大 5 バイトまで拡張されます。DynamoDB は、このような変更レコードを Kinesis データストリームにレプリケートできません。DynamoDB は、これらの変更データレコードをスキップし、後続のレコードを自動的にレプリケートします。

前述のメトリクスのいずれかが特定のしきい値を超えた場合に通知するために、Amazon Simple Notification Service (Amazon SNS) メッセージを送信する Amazon CloudWatch アラームを作成できます。

Amazon Kinesis Data Streams および Amazon DynamoDB の IAM ポリシーを使用する

Amazon DynamoDB 用 Amazon Kinesis Data Streams を初めて有効にすると、DynamoDB は AWS Identity and Access Management (IAM) サービスにリンクされたロールを自動的に作成します。このロール `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` を使用すると、DynamoDB はユーザーに代わって Kinesis Data Streams への項目レベルの変更のレプリケーションを管理できます。このサービスにリンクされたロールは削除しないでください。

サービスリンクロールの詳細については、「IAM ユーザーガイド」の「サービスリンクロールの使用」を参照してください。

Amazon DynamoDB 用 Amazon Kinesis Data Streams を有効にするには、テーブルに対する以下の許可が必要です。

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`

特定の DynamoDB テーブルに対する Amazon DynamoDB 用 Amazon Kinesis Data Streams を記述するには、テーブルに対する以下の許可が必要です。

- `dynamodb:DescribeKinesisStreamingDestination`

- kinesis:DescribeStreamSummary
- kinesis:DescribeStream

Amazon DynamoDB 用 Amazon Kinesis Data Streams を無効にするには、テーブルに対する以下の許可が必要です。

- dynamodb:DisableKinesisStreamingDestination

Amazon DynamoDB 用 Amazon Kinesis Data Streams を更新にするには、テーブルに対する以下の許可が必要です。

- dynamodb:UpdateKinesisStreamingDestination

以下の例は、IAM ポリシーを使用して Amazon DynamoDB 用 Amazon Kinesis Data Streams の許可を付与する方法を示しています。

例: Amazon DynamoDB 用 Amazon Kinesis Data Streams を有効にする

以下の IAM ポリシーは、`Music` テーブルに対して Amazon DynamoDB 用 Amazon Kinesis Data Streams を有効にする権限を付与します。`Music` テーブルに対して DynamoDB 用 Kinesis Data Streams を無効化、更新、記述する権限は付与されません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
kinesisreplication.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBKinesisDataStreamsReplication",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "kinesisreplication.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [

                "dynamodb:EnableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
        }
    ]
}
```

例: Amazon DynamoDB 用 Amazon Kinesis Data Streams を更新する

以下の IAM ポリシーは、`Music` テーブルに対して Amazon DynamoDB 用 Amazon Kinesis Data Streams を更新する権限を付与します。`Music` テーブルに対して DynamoDB 用 Kinesis Data Streams を有効化、無効化、記述する権限は付与されません。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
        }
    ]
}
```

例: Amazon DynamoDB 用 Amazon Kinesis Data Streams を無効にする

以下の IAM ポリシーは、`Music` テーブルに対して Amazon DynamoDB 用 Amazon Kinesis Data Streams を無効にする権限を付与します。`Music` テーブルに対して Amazon DynamoDB 用 Amazon Kinesis Data Streams を有効化、更新、記述する権限は付与されません。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
        }
    ]
}
```

例: リソースに基づいて、選択的に Amazon DynamoDB 用 Amazon Kinesis Data Streams に許可を適用する

以下の IAM ポリシーは、`Music` テーブルの Amazon DynamoDB 用 Amazon Kinesis Data Streams を有効にするか、記述する許可を付与しますが、`Orders` テーブルの Amazon DynamoDB 用 Amazon Kinesis Data Streams を無効にする許可は拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Music"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:12345678901:table/Orders"
        }

    ]
}
```

DynamoDB 用 Kinesis Data Streams に対するサービスにリンクされたロールの使用

Amazon DynamoDB 用 Amazon Kinesis Data Streams は、AWS Identity and Access Management (IAM) のサービスにリンクされたロールを使用します。サービスにリンクされたロールは、一意のタイプの IAM ロールで、DynamoDB 用 Kinesis Data Streams に直接リンクされます。サービスにリンクされたロールは、DynamoDB 用 Kinesis Data Streams によって事前定義されており、お客様の代わりにサービスから他の AWS のサービスを呼び出す必要のあるアクセス許可がすべて含まれています。

サービスにリンクされたロールを使用することで、必要なアクセス権限を手動で追加する必要がなくなるため、DynamoDB 用 Kinesis Data Streams の設定が簡単になります。DynamoDB 用 Kinesis

Data Streams は、サービスにリンクされたロールの許可を定義します。特に定義されている場合を除き、DynamoDB 用 Kinesis Data Streams のみがそのロールを引き受けることができます。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールをサポートする他のサービスについては、IAM と連携する AWS のサービスを参照して、[サービスにリンクされたロール] 列が [はい] になっているサービスを探してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

DynamoDB 用 Kinesis Data Streams に対するサービスにリンクされたロールの許可

DynamoDB 用 Kinesis Data Streams では、AWSServiceRoleForDynamoDBKinesisDataStreamsReplication という名前の、サービスにリンクされたロールを使用します。サービスにリンクされたロールの目的は、Amazon DynamoDB がお客様に代わって Kinesis Data Streams に対する項目レベルの変更のレプリケーションを管理できるように許可することです。

`AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` サービスリンクロールは、ロールの引き受けについて以下のサービスを信頼します。

- `kinesisreplication.dynamodb.amazonaws.com`

ロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを完了することを DynamoDB 用 Kinesis Data Streams に許可します。

- アクション: `Kinesis stream` 上で `Put records and describe`
- アクション: `AWS KMS` で `Generate data keys`、ユーザーが生成した AWS KMS キーを使用して暗号化された Kinesis ストリームにデータを格納します。

ポリシードキュメントの正確な内容については、「DynamoDBKinesisReplicationServiceRolePolicy」を参照してください。

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、アクセス許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの許可」を参照してください。

DynamoDB 用 Kinesis Data Streams のサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。AWS Management Console、AWS CLI または AWS API で DynamoDB 用 Kinesis Data Streams を有効にすると、DynamoDB 用 Kinesis Data Streams がサービスにリンクされたロールを作成します。

このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。DynamoDB 用 Kinesis Data Streams を有効にすると、DynamoDB 用 Kinesis Data Streams がサービスにリンクされたロールを再作成します。

DynamoDB 用 Kinesis Data Streams のサービスにリンクされたロールの編集

DynamoDB 用 Kinesis Data Streams では、`AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` サービスにリンクされたロールを編集することはできません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの編集」を参照してください。

DynamoDB 用 Kinesis Data Streams のサービスにリンクされたロールの削除

サービスにリンクされたロールは、IAM コンソール、AWS CLI 、または AWS API を使用して手動で削除することもできます。そのためにはまず、サービスにリンクされたロールのリソースをクリーンアップする必要があります。その後で、手動で削除できます。

> **Note**
>
> リソースを削除する際に、DynamoDB 用 Kinesis Data Streams サービスでロールが使用されている場合、削除は失敗することがあります。失敗した場合は、数分待ってから再度オペレーションを実行してください。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、`AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` サービスリンクロールを削除します。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

DynamoDB Streams の変更データキャプチャ

DynamoDB Streams は、DynamoDB テーブル内の項目レベルの変更に関するシーケンスを時間順にキャプチャし、その情報を最大 24 時間ログに保存します。アプリケーションは、このログにアクセスし、データ項目の変更前および変更後の内容をほぼリアルタイムで参照できます。

保管時の暗号化では、DynamoDB Streams のデータが暗号化されます。詳細については、「保管時の DynamoDB 暗号化」を参照してください。

DynamoDB Streams は、DynamoDB テーブル内の項目に加えられた変更に関する情報の順序付けされた情報です。テーブルでストリーミングを有効にすると、DynamoDB はテーブル内のデータ項目に加えられた各変更に関する情報をキャプチャします。

アプリケーションがテーブル内の項目を作成、更新、または削除するたびに、DynamoDB Streams は変更された項目のプライマリキー属性を付けてストリーミングレコードを書き込みます。ストリーミングレコードには、DynamoDB テーブル内の単一の項目に加えられたデータ変更についての情報が含まれています。ストリームレコードが追加情報 (変更された項目の前後のイメージ) をキャプチャするようにストリームを設定できます。

DynamoDB Streams を使用すれば、以下のことを確認できます。

- 各ストリームレコードは、ストリームに 1 回だけ出現します。
- DynamoDB テーブルで変更された各項目について、ストリーミングレコードは項目に対する実際の変更と同じ順序で出現します。

DynamoDB Streams は、ストリーミングレコードをほぼリアルタイムで書き込むため、これらのストリーミングを使用し、内容に基づいてアクションを実行するアプリケーションを構築できます。

トピック

DynamoDB Streams のエンドポイント

AWS では、DynamoDB と DynamoDB Streams 用に個別のエンドポイントを維持しています。データベースのテーブルとインデックスを使用するには、アプリケーションが DynamoDB エンドポイントにアクセスする必要があります。DynamoDB Streams レコードを読み込んで処理するには、アプリケーションが同じリージョンの DynamoDB Streams エンドポイントにアクセスする必要があります。

DynamoDB Streams エンドポイントの命名規則は `streams.dynamodb.<region>.amazonaws.com` です。たとえば、エンドポイント `dynamodb.us-west-2.amazonaws.com` を使用して DynamoDB にアクセスする場合は、エンドポイント `streams.dynamodb.us-west-2.amazonaws.com` を使用して DynamoDB Streams にアクセスします。

> Note
>
> DynamoDB および DynamoDB Streams のリージョンとエンドポイントの完全なリストについては、「AWS 全般のリファレンス」の「リージョンとエンドポイント」を参照してください。

AWS SDK は、DynamoDB と DynamoDB Streams 用に個別のクライアントを提供します。要件によっては、アプリケーションは、DynamoDB エンドポイント、DynamoDB Streams エンドポイント、または両方に同時にアクセスできます。両方のエンドポイントに接続するには、アプリケーションで 2 つのクライアントをインスタンス化する必要があります。1 つは DynamoDB 用、もう 1 つは DynamoDB Streams 用です。

ストリームの有効化

新しいテーブルでは、AWS CLI または AWS SDK 経由でそのテーブルの作成時にストリームを有効にできます。また、既存のテーブルでストリーミングを有効または無効にすることや、ストリーミングの設定を変更することができます。DynamoDB Streams は非同期的に動作するため、ストリーミングを有効にしてもテーブルのパフォーマンスに影響はありません。

DynamoDB Streams を管理する最も簡単な方法は、AWS Management Console を使用することです。

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。

2. DynamoDB コンソールのダッシュボードで、[Tables (テーブル)] を選択して既存テーブルを選びます。
3. [エクスポートとストリーム] タブを選択します。
4. [DynamoDB ストリームの詳細] セクションで、[オンにする] を選択します。
5. [DynamoDB ストリームをオンにする] ウィンドウで、テーブルのデータが変更されるたびにストリーミングに書き込まれる情報を選択します。

   - [キー属性のみ] - 変更された項目のキー属性のみ。
   - [New image] (新規イメージ) — 変更後に表示される項目全体。
   - [Old image] (古いイメージ) — 変更前に表示されていた項目全体。
   - [New and old images] (新規イメージおよび古いイメージ) — 項目の新しいイメージと古いイメージの両方。

   すべての設定が正しいことを確認したら、[ストリームをオンにする] を選択します。
6. (オプション) 既存のストリーミングを無効にするには、[DynamoDB ストリームの詳細] で [オフにする] を選択します。

`CreateTable` または `UpdateTable` API オペレーションを使用して、ストリームを有効にするか、変更することもできます。ストリームの設定内容は、`StreamSpecification` パラメータにより決まります。

- `StreamEnabled` — テーブルでストリーミングが有効 (`true`) か無効 (`false`) かを指定します。
- `StreamViewType` — テーブル内のデータが変更されるたびにストリーミングに書き込まれる情報を指定します。
  - `KEYS_ONLY` — 変更された項目のキー属性のみ。
  - `NEW_IMAGE` — 変更後に表示される項目全体。
  - `OLD_IMAGE` — 変更前に表示されていた項目全体。
  - `NEW_AND_OLD_IMAGES` — 項目の新しいイメージと古いイメージの両方。

ストリームはいつでも有効または無効にできます。ただし、既にストリームがあるテーブルでストリームを有効にしようとした場合、`ResourceInUseException` を受け取ります。ストリームのないテーブルでストリームを無効にしようとした場合、`ValidationException` を受け取ります。

StreamEnabled を true に設定すると、一意のストリーミング記述子が割り当てられた新しいストリーミングが DynamoDB で作成されます。テーブルでストリームを無効にして再度有効にすると、新しいストリームは異なるストリーム記述子で作成されます。

各ストリームは、Amazon リソースネーム (ARN) により一意に識別されます。次に、TestTable という名前の DynamoDB テーブルにあるストリーミングのサンプル ARN を示します。

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

テーブルの最新のストリーミング記述子を調べるには、DynamoDB DescribeTable リクエストを発行し、レスポンスで LatestStreamArn 要素を探します。

> ⓘ Note
>
> ストリームのセットアップ後は StreamViewType を編集できません。セットアップ後にストリームを変更する必要がある場合は、現在のストリームを無効にして新しいストリームを作成する必要があります。

ストリームの読み込みと処理

ストリームを読み取って処理するには、アプリケーションから DynamoDB Streams エンドポイントに接続して API リクエストを発行する必要があります。

ストリームは、ストリームレコードで構成されています。各ストリーミングレコードは、ストリーミングが属する DynamoDB テーブル内の 1 件のデータ変更を表しています。各ストリームレコードには、レコードがストリームに発行された順序を反映したシーケンス番号が割り当てられます。

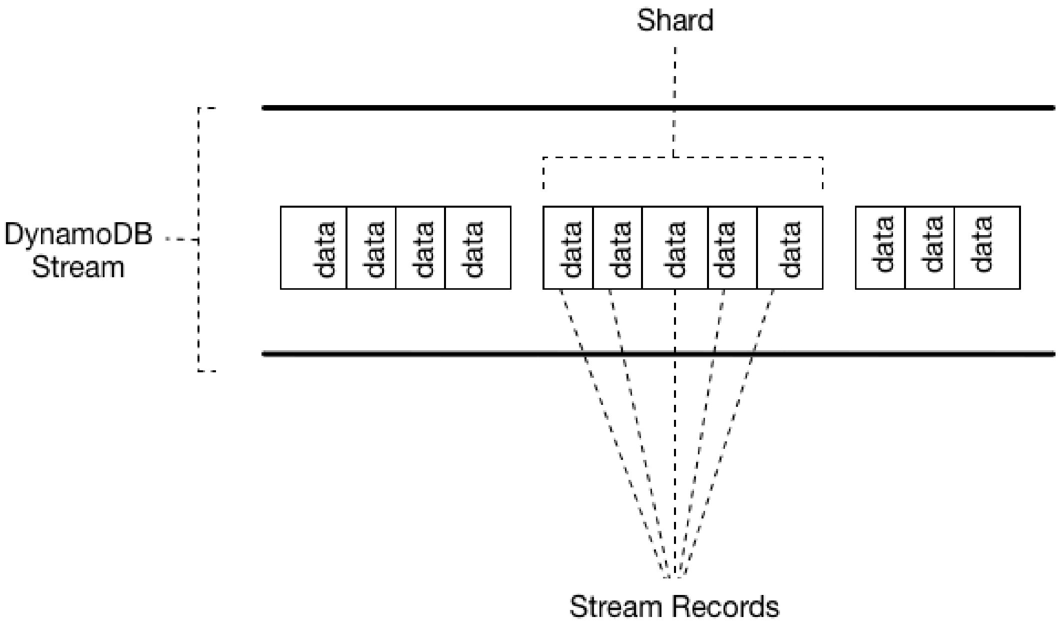
ストリームレコードは、グループ (つまり、シャード) に整理されます。各シャードは、複数のストリームレコードのコンテナとして機能し、これらのレコードへのアクセスと反復処理に必要な情報が含まれています。シャード内のストリームレコードは 24 時間後に自動的に削除されます。

シャードはエフェメラルであり、必要に応じて自動的に作成および削除されます。また、任意のシャードは複数の新しいシャードに分割できます。これもまた自動的に行われます (親シャードが 1 つの子シャードのみを持つ場合もあります)。アプリケーションが複数のシャードからレコードを並列処理できるように、シャードは親テーブルで高レベルな書き込みアクティビティに応じて分割される場合があります。

ストリームを無効にすると、開かれているシャードは閉じられます。ストリーミング内のデータは 24 時間読み込み可能な状態になります。

シャードには系列 (親と子) があるため、アプリケーションは子シャードを処理する前に、必ず親シャードを処理する必要があります。これにより、ストリームレコードも正しい順序で処理されるようになります。(DynamoDB Streams Kinesis Adapter を使用している場合、これは自動的に処理されます。アプリケーションは、シャードとストリーミングレコードを正しい順序で処理します。アプリケーションの実行中に分割されたシャードに加えて、新しいシャードまたは有効期限切れのシャードは自動的に処理されます。詳細については、「DynamoDB Streams Kinesis Adapter を使用したストリームレコードの処理」を参照してください。)

次の図は、ストリーム、ストリーム内のシャード、シャード内のストリームレコードの関係を示しています。



> Note
>
> 項目内のデータを何も変更しない `PutItem` または `UpdateItem` オペレーションを実行した場合、そのオペレーションのストリーミングレコードは DynamoDB Streams によって書き込まれません。

ストリームにアクセスしてその中のストリームレコードを処理するには、以下の操作を実行する必要があります。

- アクセスするストリームの一意の ARN を調べます。
- 目的のストリームレコードがストリーム内のどのシャードに含まれているかを調べます。
- シャードにアクセスし、目的のストリームレコードを取得します。

> Note
>
> 最大でも 2 つを超えるプロセスが、同時に同じストリームシャードから読み込みを行うことはできません。シャードごとに 2 つを超えるリーダーがあると、スロットリングが発生する場合があります。

DynamoDB Streams API は、アプリケーションプログラム用の以下のアクションを提供します。

- `ListStreams` — 現在のアカウントおよびエンドポイントのストリーミング記述子のリストを返します。必要に応じて、特定のテーブル名のストリーム記述子だけをリクエストできます。
- `DescribeStream` — 特定のストリーミングに関する詳細情報を返します。出力には、ストリームに関連付けられたシャードのリストが含まれています（シャード ID を含む）。
- `GetShardIterator` — シャード内の場所を表すシャードイテレーターを返します。イテレータがストリーム内の最も古いポイント、最も新しいポイント、特定のポイントへのアクセスを提供することをリクエストできます。
- `GetRecords` — 特定のシャード内からストリーミングレコードを返します。`GetShardIterator` リクエストから返されたシャードイテレーターを指定する必要があります。

リクエストやレスポンスの例など、これらの API オペレーションの詳細な説明については、「Amazon DynamoDB Streams API リファレンス」を参照してください。

DynamoDB Streams のデータ保持期限

DynamoDB Streams 内のすべてのデータは、24 時間保持されます。特定のテーブルの直近 24 時間のアクティビティを取得して分析できます。ただし、24 時間を超えたデータはすぐにトリミング (削除) される可能性があります。

テーブルのストリームを無効にした場合、ストリーム内のデータは 24 時間読み込み可能な状態になります。この時間が経過すると、データは期限切れになり、ストリームレコードは自動的に削除されます。既存のストリームを手動で削除するためのメカニズムはありません。保持期限 (24 時間) が切れ、すべてのストリームレコードが削除されるまで待つ必要があります。

DynamoDB Streams と有効期限 (TTL)

テーブルに対して Amazon DynamoDB Streams を有効にし、期限切れの項目のストリーミングレコードを処理することで、有効期限 (TTL) によって削除された項目をバックアップ (または処理) できます。詳細については、「ストリームの読み込みと処理」を参照してください。

ストリームレコードにはユーザー ID フィールド `Records[<index>].userIdentity` が含まれます。

有効期限切れの後に有効期限 (TTL) プロセスによって削除された項目には、次のフィールドが含まれています。

- `Records[<index>].userIdentity.type`

  `"Service"`
- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> Note
>
> TTL をグローバルテーブルで使用すると、TTL が実行されたリージョンに `userIdentity` フィールドが設定されます。削除が複製されても、このフィールドは他のリージョンには設定されません。

次の JSON は 1 つのストリームレコードの関連する部分を示しています。

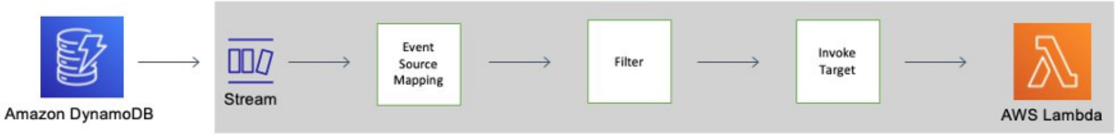
```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

DynamoDB Streams と Lambda を使用して TTL 削除済みアイテムをアーカイブする

DynamoDB 有効期限 (TTL)、DynamoDB Streams および AWS Lambda を組み合わせると、データのアーカイブを簡素化し、DynamoDB ストレージコストを削減し、コードの複雑さを軽減するのに役立ちます。ストリームコンシューマーとして Lambda を使用すると、Kinesis Client Library (KCL) などの他のコンシューマーと比較してコストが削減されるなど、多くの利点があります。Lambda を使用してイベントを消費する場合、DynamoDB ストリームの `GetRecords` API 呼び出しでは課金が発生しません。Lambda はストリームイベント内の JSON パターンを識別してイベントフィルタリングを提供できます。イベントパターンのコンテンツフィルタリングでは、最大 5 つの異なるフィルターを定義して、処理のために Lambda に送信されるイベントを制御できます。これにより、Lambda 関数の呼び出しを減らしてコードを簡素化し、全体的なコストを削減できます。

DynamoDB Streams には、`Create`、`Modify` および`Remove` アクションなどのすべてのデータ変更が含まれています。これは、アーカイブ Lambda 関数の不要な呼び出しを引き起こす可能性があります。例えば、1 時間あたり 200 万件のデータ変更がストリームに流れ込むテーブルがあり、そのうち 5% 未満が TTL プロセスによって期限切れになり、アーカイブする必要があるアイテム削除であるとします。Lambda イベントソースフィルターを使用すると、Lambda 関数は 1 時間あたり 100,000 回しか呼び出されません。イベントフィルタリングを使用した結果、イベントフィルタリングを行わなければ 200 万回の呼び出しが発生するところを、必要な呼び出しに対してのみ課金されることになります。

イベントフィルタリングは、Lambda イベントソースマッピングに適用されます。これは、選択されたイベント (DynamoDB ストリーム) から読み取り、Lambda 関数を呼び出すリソースです。次の図は、ストリームとイベントフィルターを使用して Lambda 関数によって有効期限 (TTL) 削除済みアイテムがどのように消費されるかを示しています。



DynamoDB の有効期限 (TTL) イベントフィルターパターン

イベントソースマッピングのフィルター条件に次の JSON を追加することで、TTL 削除済みアイテムに対してのみ Lambda 関数の呼び出しを許可します。

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

AWS Lambda イベントソースマッピングを作成します。

次のコードスニペットを使用して、テーブルの DynamoDB ストリームに接続できる、フィルター処理されたイベントソースマッピングを作成します。各コードブロックには、イベントフィルターパターンが含まれます。

AWS CLI

```
aws lambda create-event-source-mapping \
--event-source-arn 'arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000' \
--batch-size 10 \
--enabled \
--function-name test_func \
--starting-position LATEST \
--filter-criteria '{"Filters": [{"Pattern": "{\"userIdentity\":{\"type\":[\"Service
\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}"}]}'
```

Java

```
LambdaClient client = LambdaClient.builder()
        .region(Region.EU_WEST_1)
        .build();

Filter userIdentity = Filter.builder()
        .pattern("{\"userIdentity\":{\"type\":[\"Service\"],\"principalId\":
[\"dynamodb.amazonaws.com\"]}}")
        .build();

FilterCriteria filterCriteria = FilterCriteria.builder()
        .filters(userIdentity)
        .build();

CreateEventSourceMappingRequest mappingRequest =
 CreateEventSourceMappingRequest.builder()
```

```
        .eventSourceArn("arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000")
        .batchSize(10)
        .enabled(Boolean.TRUE)
        .functionName("test_func")
        .startingPosition("LATEST")
        .filterCriteria(filterCriteria)
        .build();

try{
    CreateEventSourceMappingResponse eventSourceMappingResponse =
 client.createEventSourceMapping(mappingRequest);
    System.out.println("The mapping ARN is
 "+eventSourceMappingResponse.eventSourceArn());

}catch (ServiceException e){
    System.out.println(e.getMessage());
}
```

Node

```
const client = new LambdaClient({ region: "eu-west-1" });

const input = {
    EventSourceArn: "arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000",
    BatchSize: 10,
    Enabled: true,
    FunctionName: "test_func",
    StartingPosition: "LATEST",
    FilterCriteria: { "Filters": [{ "Pattern": "{\"userIdentity\":{\"type\":
[\"Service\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}" }] }
}

const command = new CreateEventSourceMappingCommand(input);

try {
    const results = await client.send(command);
    console.log(results);
} catch (err) {
    console.error(err);
}
```

Python

```
session = boto3.session.Session(region_name = 'eu-west-1')
client = session.client('lambda')

try:
    response = client.create_event_source_mapping(
        EventSourceArn='arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000',
        BatchSize=10,
        Enabled=True,
        FunctionName='test_func',
        StartingPosition='LATEST',
        FilterCriteria={
            'Filters': [
                {
                    'Pattern': "{\"userIdentity\":{\"type\":[\"Service\"],
\"principalId\":[\"dynamodb.amazonaws.com\"]}}"
                },
            ]
        }
    )
    print(response)
except Exception as e:
    print(e)
```

JSON

```
{
  "userIdentity": {
     "type": ["Service"],
     "principalId": ["dynamodb.amazonaws.com"]
   }
}
```

DynamoDB Streams Kinesis Adapter を使用したストリームレコードの処理

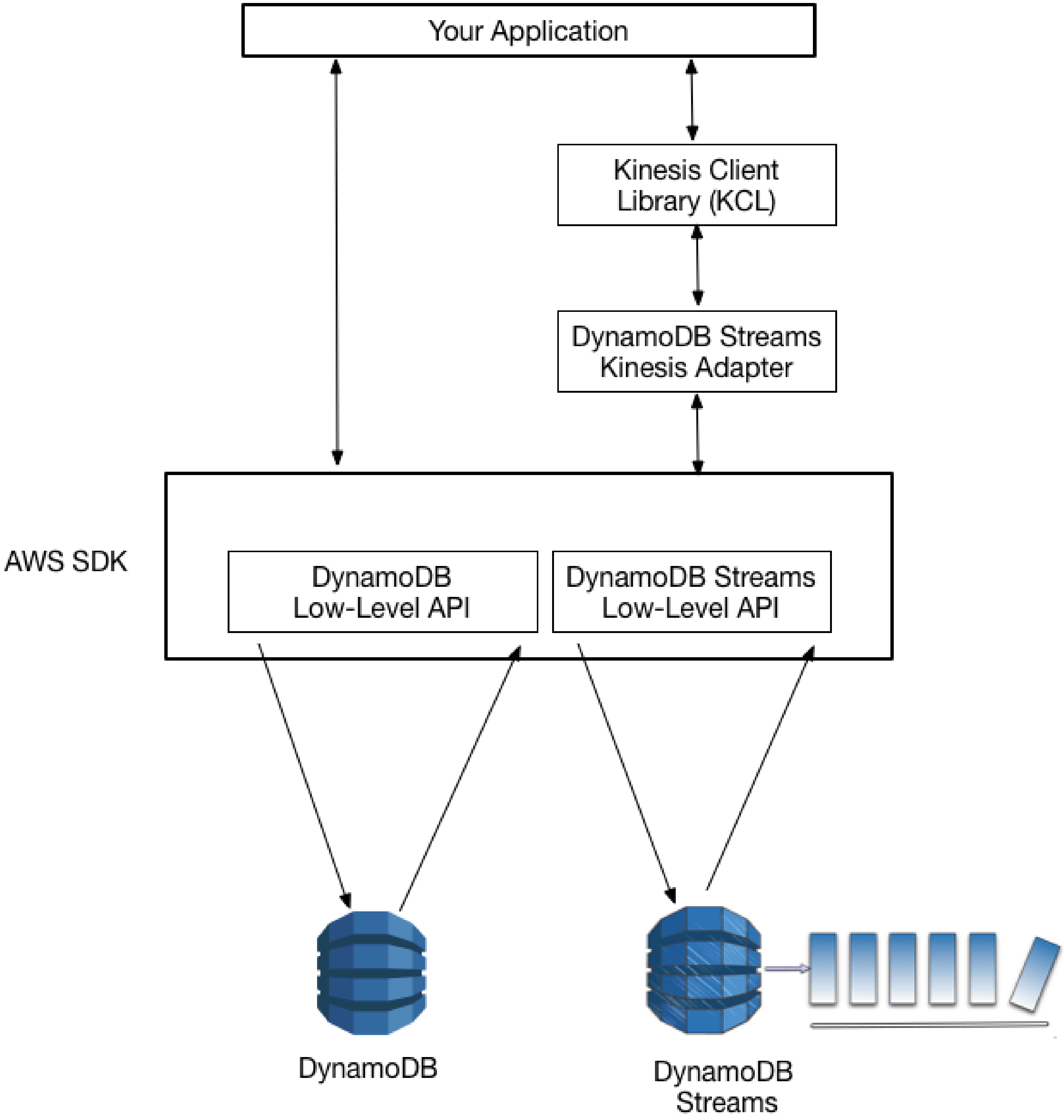
Amazon DynamoDB からのストリーミングを使用するには、Amazon Kinesis Adapter を使用することをお勧めします。DynamoDB Streams API は、Kinesis Data Streams (大規模なストリーミングデータをリアルタイムで処理するサービス) と意図的に似せています。どちらのサービスでも、データストリーミングは、ストリーミングレコードのコンテナであるシャードで構成されて

います。両方のサービスの API には `ListStreams`、`DescribeStream`、`GetShards`、および `GetShardIterator` オペレーションが含まれています。(これらの DynamoDB Streams アクションは、Kinesis Data Streams の対応するアクションと似ていますが、100% 同一ではありません)。

Kinesis Client Library (KCL) を使用して、Kinesis Data Streams のアプリケーションを書き込むことができます。KCL は、低レベルの Kinesis Data Streams API の上で役に立つ抽象化を提供することによりコーディングを簡素化します。KCL の詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Kinesis Client Library を使用したコンシューマーの開発」を参照してください。

DynamoDB Streams ユーザーは、KCL 内のデザインパターンを使用して DynamoDB Streams のシャードとストリーミングレコードを処理できます。これを行うには、DynamoDB Streams Kinesis Adapter を使用します。Kinesis Adapter は Kinesis Data Streams インターフェイスを実装しているため、KCL を使用して DynamoDB Streams からのレコードを消費および処理することができます。DynamoDB Streams Kinesis アダプターのセットアップとインストールの方法については、「GitHub リポジトリ」を参照してください。

次の図表は、これらのライブラリがどのように連携するかを示しています。


Your Application
Kinesis Client Library (KCL)
DynamoDB Streams Kinesis Adapter
AWS SDK
DynamoDB Low-Level API
DynamoDB Streams Low-Level API
DynamoDB
DynamoDB Streams

DynamoDB Streams Kinesis Adapter を使用すれば、DynamoDB Streams エンドポイントにシームレスに誘導された API コールを使用して、KCL インターフェイスに対して開発を開始できます。

アプリケーションは起動時に KCL をコールしてワーカーをインスタンス化します。ワーカーに、アプリケーションの構成情報 (ストリーミング記述子や AWS 認証情報、および提供するレコードプロセッサクラスの名前など) を提供する必要があります。レコードプロセッサでコードを実行すると、ワーカーは次のタスクを実行します。

- ストリームに接続する
- ストリーミング内のシャードを列挙します。
- シャードと他のワーカー (存在する場合) の関連付けを調整する
- レコードプロセッサで管理する各シャードのレコードプロセッサをインスタンス化する
- ストリーミングからレコードを取得します。
- 対応するレコードプロセッサにレコードを送信する
- 処理されたレコードのチェックポイントを作成する
- ワーカーのインスタンス数が変化したときに、シャードとワーカーの関連付けを調整する
- シャードが分割されたときに、シャードとワーカーの関連付けを調整します。

> Note
>
> こちらに記載されている KCL 概念の説明については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Kinesis Client Library を使用したコンシューマーの開発」を参照してください。
> AWS Lambda でのストリームの使用の詳細については、「DynamoDB Streams と AWS Lambda のトリガー」を参照してください。

チュートリアル: DynamoDB Streams Kinesis Adapter

このセクションは、Amazon Kinesis Client Library と Amazon DynamoDB Streams Kinesis Adapter を使用する Java アプリケーションのチュートリアルです。アプリケーションには、データレプリケーションの例が表示されます。データレプリケーションでは、1 つのテーブルからの書き込みアクティビティが 2 番目のテーブルに適用され、両方のテーブルの内容が同期されます。ソースコードについては、「完成したプログラム: DynamoDB Streams Kinesis Adapter」を参照してください。

このプログラムでは、次のような処理を実行します。

1. `KCL-Demo-src` と `KCL-Demo-dst` という 2 つの DynamoDB テーブルを作成します。これらの各テーブルでは、ストリームが有効になっています。

2. 項目を追加、更新、削除することで、ソーステーブルで更新アクティビティを生成します。これにより、データがテーブルのストリームに書き込まれます。
3. ストリーミングからレコードを読み込んで、DynamoDB リクエストとして再構築し、ターゲットテーブルにリクエストを適用します。
4. ソーステーブルとターゲットテーブルをスキャンし、内容が同じであることを確認します。
5. テーブルを削除してクリーンアップします。

これらのステップについては次のセクションで説明します。完成したアプリケーションは、チュートリアルの最後に示します。

トピック

ステップ 1: DynamoDB テーブルを作成する

最初のステップでは、2 つの DynamoDB テーブル (送信元テーブルと送信先テーブル) を作成します。ソーステーブルのストリームにある StreamViewType は NEW_IMAGE です。これは、このテーブルで項目が変更されると必ず、イメージの "後の" 項目がストリームに書き込まれることを意味します。このようにして、ストリームはテーブル内のすべての書き込みアクティビティを記録します。

次の例は、両方のテーブルを作成するためのコードを示しています。

```
java.util.List<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

java.util.List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition
```

```
  // key

ProvisionedThroughput provisionedThroughput = new
 ProvisionedThroughput().withReadCapacityUnits(2L)
    .withWriteCapacityUnits(2L);

StreamSpecification streamSpecification = new StreamSpecification();
streamSpecification.setStreamEnabled(true);
streamSpecification.setStreamViewType(StreamViewType.NEW_IMAGE);
CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions).withKeySchema(keySchema)

 .withProvisionedThroughput(provisionedThroughput).withStreamSpecification(streamSpecification)
```

ステップ 2: ソーステーブルに更新アクティビティを生成する

次のステップでは、ソーステーブルにいくつかの書き込みアクティビティを生成します。このアクティビティの実行中、ソーステーブルのストリームもほぼリアルタイムで更新されます。

アプリケーションは、データを書き込むための `PutItem`、`UpdateItem`、および `DeleteItem` API オペレーションを呼び出すメソッドを持つヘルパークラスを定義します。次の例は、これらのメソッドの使用方法を示しています。

```
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "101", "test1");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "101", "test2");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "101");
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "102", "demo3");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "102", "demo4");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "102");
```

ステップ 3: ストリームを処理する

ここでは、プログラムがストリームの処理を開始します。DynamoDB Streams Kinesis Adapter は、低レベルの DynamoDB Streams コールを行わなくてもコードが KCL を十分に使用できるように、KCL と DynamoDB Streams エンドポイントの間の透過的なレイヤーとして機能します。このプログラムでは次のタスクを実行しています。

- KCL インターフェイス定義に従ったメソッド（StreamsRecordProcessor、initialize、processRecords）を使用して、レコードプロセッサクラス shutdown を定義します。processRecords メソッドには、ソーステーブルのストリームからの読み込みとターゲットテーブルへの書き込みに必要なロジックが含まれています。
- レコードプロセッサクラスのクラスファクトリを定義します（StreamsRecordProcessorFactory）。これは、KCL を使用する Java プログラムに必要です。
- クラスファクトリに関連付けられた新しい KCL Worker をインスタンス化します。
- レコード処理が完了すると、Worker をシャットダウンします。

KCL インターフェイス定義の詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Kinesis Client Library を使用したコンシューマーの開発」を参照してください。

次の例は、StreamsRecordProcessor におけるメインループを示しています。case ステートメントは、ストリームレコードに出現する OperationType に基づいて、実行するアクションを決定します。

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
                com.amazonaws.services.dynamodbv2.model.Record streamRecord =
 ((RecordAdapter) record)
                    .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getNewImage());
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
    }
    checkpointCounter += 1;
    if (checkpointCounter % 10 == 0) {
        try {
            checkpointer.checkpoint();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ステップ 4: 両方のテーブルの内容が同じであることを確認する

この時点で、ソーステーブルとターゲットテーブルの内容が同期されています。アプリケーションは、両方のテーブルに対して Scan リクエストを発行し、内容が実際に同じであることを確認します。

DemoHelper クラスには、低レベルの ScanTable API を呼び出す Scan メソッドが含まれています。次の例は、その使用方法を示しています。

```
if (StreamsAdapterDemoHelper.scanTable(dynamoDBClient, srcTable).getItems()
    .equals(StreamsAdapterDemoHelper.scanTable(dynamoDBClient, destTable).getItems()))
 {
    System.out.println("Scan result is equal.");
}
else {
    System.out.println("Tables are different!");
}
```

ステップ 5：クリーンアップ

デモは完了したため、アプリケーションによりソーステーブルとターゲットテーブルが削除されます。次のコード例を参照してください。テーブルが削除されても、そのストリームは最大 24 時間使用可能です。その後、自動的に削除されます。

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

完成したプログラム: DynamoDB Streams Kinesis Adapter

次に、チュートリアル: DynamoDB Streams Kinesis Adapter で説明したタスクを実行する、完成した Java プログラムを次に示します。実行すると、次のような出力が表示されます。

```
Creating table KCL-Demo-src
Creating table KCL-Demo-dest
Table is active.
Creating worker for stream: arn:aws:dynamodb:us-west-2:111122223333:table/KCL-Demo-src/
stream/2015-05-19T22:48:56.601
Starting worker...
Scan result is equal.
Done.
```

> ⚠ Important
>
> このプログラムを実行するには、クライアントアプリケーションがポリシーを使用して DynamoDB および Amazon CloudWatch にアクセスできることを確認します。詳細については、「DynamoDB のアイデンティティベースのポリシー」を参照してください。

ソースコードは、4 つの .java ファイルから構成されています。

- StreamsAdapterDemo.java
- StreamsRecordProcessor.java
- StreamsRecordProcessorFactory.java
- StreamsAdapterDemoHelper.java

StreamsAdapterDemo.java

```

package com.amazonaws.codesamples;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreams;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreamsClientBuilder;
import com.amazonaws.services.dynamodbv2.model.DeleteTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
```

```
import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsWorkerFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;

public class StreamsAdapterDemo {
    private static Worker worker;
    private static KinesisClientLibConfiguration workerConfig;
    private static IRecordProcessorFactory recordProcessorFactory;

    private static AmazonDynamoDB dynamoDBClient;
    private static AmazonCloudWatch cloudWatchClient;
    private static AmazonDynamoDBStreams dynamoDBStreamsClient;
    private static AmazonDynamoDBStreamsAdapterClient adapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Regions awsRegion = Regions.US_EAST_2;

    private static AWSCredentialsProvider awsCredentialsProvider =
 DefaultAWSCredentialsProviderChain.getInstance();

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {
        System.out.println("Starting demo...");

        dynamoDBClient = AmazonDynamoDBClientBuilder.standard()
                .withRegion(awsRegion)
                .build();
        cloudWatchClient = AmazonCloudWatchClientBuilder.standard()
                .withRegion(awsRegion)
                .build();
        dynamoDBStreamsClient = AmazonDynamoDBStreamsClientBuilder.standard()
                .withRegion(awsRegion)
                .build();
        adapterClient = new AmazonDynamoDBStreamsAdapterClient(dynamoDBStreamsClient);
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        recordProcessorFactory = new StreamsRecordProcessorFactory(dynamoDBClient, 
destTable);

        setUpTables();

        workerConfig = new KinesisClientLibConfiguration("streams-adapter-demo",
                streamArn,
                awsCredentialsProvider,
                "streams-demo-worker")
                .withMaxRecords(1000)
                .withIdleTimeBetweenReadsInMillis(500)
                .withInitialPositionInStream(InitialPositionInStream.TRIM_HORIZON);

        System.out.println("Creating worker for stream: " + streamArn);
        worker = 
StreamsWorkerFactory.createDynamoDbStreamsWorker(recordProcessorFactory, workerConfig, 
adapterClient,
                dynamoDBClient, cloudWatchClient);
        System.out.println("Starting worker...");
        Thread t = new Thread(worker);
        t.start();

        Thread.sleep(25000);
        worker.shutdown();
        t.join();

        if (StreamsAdapterDemoHelper.scanTable(dynamoDBClient, srcTable).getItems()
                .equals(StreamsAdapterDemoHelper.scanTable(dynamoDBClient, 
destTable).getItems())) {
            System.out.println("Scan result is equal.");
        } else {
            System.out.println("Tables are different!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDBClient, srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDBClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
    }

    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResult result =
 StreamsAdapterDemoHelper.describeTable(dynamoDBClient, tableName);
            created = result.getTable().getTableStatus().equals("ACTIVE");
            if (created) {
                System.out.println("Table is active.");
                return;
            } else {
                retries++;
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // do nothing
                }
            }
        }
        System.out.println("Timeout after table creation. Exiting...");
        cleanupAndExit(1);
    }

    private static void performOps(String tableName) {
        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "101", "test1");
        StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "101", "test2");
        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "101");
        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "102", "demo3");
        StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "102", "demo4");
        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "102");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
        dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
        System.exit(returnValue);
    }
}
```

StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.streamsadapter.model.RecordAdapter;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;
import com.amazonaws.services.kinesis.model.Record;

import java.nio.charset.Charset;

public class StreamsRecordProcessor implements IRecordProcessor {
    private Integer checkpointCounter;

    private final AmazonDynamoDB dynamoDBClient;
    private final String tableName;

    public StreamsRecordProcessor(AmazonDynamoDB dynamoDBClient2, String tableName) {
        this.dynamoDBClient = dynamoDBClient2;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
        checkpointCounter = 0;
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (Record record : processRecordsInput.getRecords()) {
            String data = new String(record.getData().array(),
 Charset.forName("UTF-8"));
            System.out.println(data);
            if (record instanceof RecordAdapter) {
                com.amazonaws.services.dynamodbv2.model.Record streamRecord =
 ((RecordAdapter) record)
                        .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                                streamRecord.getDynamodb().getNewImage());
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                                streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
            }
            checkpointCounter += 1;
            if (checkpointCounter % 10 == 0) {
                try {
                    processRecordsInput.getCheckpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {
    private final String tableName;
    private final AmazonDynamoDB dynamoDBClient;

    public StreamsRecordProcessorFactory(AmazonDynamoDB dynamoDBClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDBClient = dynamoDBClient;
    }

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.CreateTableResult;
import com.amazonaws.services.dynamodbv2.model.DeleteItemRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.PutItemRequest;
import com.amazonaws.services.dynamodbv2.model.ResourceInUseException;
import com.amazonaws.services.dynamodbv2.model.ScanRequest;
import com.amazonaws.services.dynamodbv2.model.ScanResult;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.model.UpdateItemRequest;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(AmazonDynamoDB client, String tableName) {
        java.util.List<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
        attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

        java.util.List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
        keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

          // key

        ProvisionedThroughput provisionedThroughput = new
 ProvisionedThroughput().withReadCapacityUnits(2L)
                .withWriteCapacityUnits(2L);

        StreamSpecification streamSpecification = new StreamSpecification();
        streamSpecification.setStreamEnabled(true);
        streamSpecification.setStreamViewType(StreamViewType.NEW_IMAGE);
        CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)

 .withAttributeDefinitions(attributeDefinitions).withKeySchema(keySchema)

 .withProvisionedThroughput(provisionedThroughput).withStreamSpecification(streamSpecification)
```

```
       try {
           System.out.println("Creating table " + tableName);
           CreateTableResult result = client.createTable(createTableRequest);
           return result.getTableDescription().getLatestStreamArn();
       } catch (ResourceInUseException e) {
           System.out.println("Table already exists.");
           return describeTable(client, tableName).getTable().getLatestStreamArn();
       }
   }

   public static DescribeTableResult describeTable(AmazonDynamoDB client, String
tableName) {
       return client.describeTable(new
DescribeTableRequest().withTableName(tableName));
   }

   public static ScanResult scanTable(AmazonDynamoDB dynamoDBClient, String tableName)
{
       return dynamoDBClient.scan(new ScanRequest().withTableName(tableName));
   }

   public static void putItem(AmazonDynamoDB dynamoDBClient, String tableName, String
id, String val) {
       java.util.Map<String, AttributeValue> item = new HashMap<String,
AttributeValue>();
       item.put("Id", new AttributeValue().withN(id));
       item.put("attribute-1", new AttributeValue().withS(val));

       PutItemRequest putItemRequest = new
PutItemRequest().withTableName(tableName).withItem(item);
       dynamoDBClient.putItem(putItemRequest);
   }

   public static void putItem(AmazonDynamoDB dynamoDBClient, String tableName,
           java.util.Map<String, AttributeValue> items) {
       PutItemRequest putItemRequest = new
PutItemRequest().withTableName(tableName).withItem(items);
       dynamoDBClient.putItem(putItemRequest);
   }

   public static void updateItem(AmazonDynamoDB dynamoDBClient, String tableName,
String id, String val) {
       java.util.Map<String, AttributeValue> key = new HashMap<String,
AttributeValue>();
```

```
        key.put("Id", new AttributeValue().withN(id));

        Map<String, AttributeValueUpdate> attributeUpdates = new HashMap<String,
 AttributeValueUpdate>();
        AttributeValueUpdate update = new
 AttributeValueUpdate().withAction(AttributeAction.PUT)
                .withValue(new AttributeValue().withS(val));
        attributeUpdates.put("attribute-2", update);

        UpdateItemRequest updateItemRequest = new
 UpdateItemRequest().withTableName(tableName).withKey(key)
                .withAttributeUpdates(attributeUpdates);
        dynamoDBClient.updateItem(updateItemRequest);
    }

    public static void deleteItem(AmazonDynamoDB dynamoDBClient, String tableName,
 String id) {
        java.util.Map<String, AttributeValue> key = new HashMap<String,
 AttributeValue>();
        key.put("Id", new AttributeValue().withN(id));

        DeleteItemRequest deleteItemRequest = new
 DeleteItemRequest().withTableName(tableName).withKey(key);
        dynamoDBClient.deleteItem(deleteItemRequest);
    }

}
```

DynamoDB Streams 低レベル API: Java の例

> **Note**
>
> このページのコードはすべてを網羅していないため、Amazon DynamoDB Streams を使用するすべてのシナリオに対応しているわけではありません。DynamoDB からストリーミングレコードを使用する場合は、DynamoDB Streams Kinesis Adapter を使用したストリームレコードの処理 で説明されているとおり、Kinesis Client Library (KCL) を使用し、Amazon Kinesis Adapter を介して行うことを推奨します。

このセクションには、動作中の DynamoDB Streams を示す Java プログラムが含まれています。このプログラムでは、次のような処理を実行します。

1. ストリーミングが有効になった DynamoDB テーブルを作成します。
2. このテーブルのストリーム設定を記述します。
3. テーブル内のデータを変更します。
4. ストリーム内のシャードを記述します。
5. シャードからストリームレコードを読み込みます。
6. クリーンアップします。

プログラムを実行すると、以下のような出力が表示されます。

```
Issuing CreateTable request for TestTableForStreams
Waiting for TestTableForStreams to be created...
Current stream ARN for TestTableForStreams: arn:aws:dynamodb:us-
east-2:123456789012:table/TestTableForStreams/stream/2018-03-20T16:49:55.208
Stream enabled: true
Update view type: NEW_AND_OLD_IMAGES

Performing write activities on TestTableForStreams
Processing item 1 of 100
Processing item 2 of 100
Processing item 3 of 100
...
Processing item 100 of 100

Shard: {ShardId: shardId-1234567890-...,SequenceNumberRange: {StartingSequenceNumber:
 01234567890...,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys:
 {Id={N: 1,}},NewImage: {Message={S: New item!,}, Id={N: 1,}},SequenceNumber:
 100000000003218256368,SizeBytes: 24,StreamViewType: NEW_AND_OLD_IMAGES}
         {ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys: {Id={N:
 1,}},NewImage: {Message={S: This item has changed,}, Id={N: 1,}},OldImage:
 {Message={S: New item!,}, Id={N: 1,}},SequenceNumber: 200000000003218256412,SizeBytes:
 56,StreamViewType: NEW_AND_OLD_IMAGES}
         {ApproximateCreationDateTime: Tue Mar 20 09:50:00 PDT 2018,Keys: {Id={N:
 1,}},OldImage: {Message={S: This item has changed,}, Id={N: 1,}},SequenceNumber:
 300000000003218256413,SizeBytes: 36,StreamViewType: NEW_AND_OLD_IMAGES}
...
Deleting the table...
Demo complete
```

Example

```
package com.amazon.codesamples;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreams;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBStreamsClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeAction;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.AttributeValueUpdate;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamRequest;
import com.amazonaws.services.dynamodbv2.model.DescribeStreamResult;
import com.amazonaws.services.dynamodbv2.model.DescribeTableResult;
import com.amazonaws.services.dynamodbv2.model.GetRecordsRequest;
import com.amazonaws.services.dynamodbv2.model.GetRecordsResult;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorRequest;
import com.amazonaws.services.dynamodbv2.model.GetShardIteratorResult;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.Record;
import com.amazonaws.services.dynamodbv2.model.Shard;
import com.amazonaws.services.dynamodbv2.model.ShardIteratorType;
import com.amazonaws.services.dynamodbv2.model.StreamSpecification;
import com.amazonaws.services.dynamodbv2.model.StreamViewType;
import com.amazonaws.services.dynamodbv2.util.TableUtils;

public class StreamsLowLevelDemo {

        public static void main(String args[]) throws InterruptedException {
                AmazonDynamoDB dynamoDBClient = AmazonDynamoDBClientBuilder
                                .standard()
                                .withRegion(Regions.US_EAST_2)
                                .withCredentials(new
 DefaultAWSCredentialsProviderChain())
                                .build();

                AmazonDynamoDBStreams streamsClient =
 AmazonDynamoDBStreamsClientBuilder
                                .standard()
                                .withRegion(Regions.US_EAST_2)
                                .withCredentials(new
 DefaultAWSCredentialsProviderChain())
                                .build();

                // Create a table, with a stream enabled
                String tableName = "TestTableForStreams";

                ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>(
                                Arrays.asList(new AttributeDefinition()
                                                .withAttributeName("Id")
                                                .withAttributeType("N")));

                ArrayList<KeySchemaElement> keySchema = new ArrayList<>(
                                Arrays.asList(new KeySchemaElement()
                                                .withAttributeName("Id")
                                                .withKeyType(KeyType.HASH))); //
 Partition key

                StreamSpecification streamSpecification = new StreamSpecification()
                                .withStreamEnabled(true)
                                .withStreamViewType(StreamViewType.NEW_AND_OLD_IMAGES);

                CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)

 .withKeySchema(keySchema).withAttributeDefinitions(attributeDefinitions)
                                .withProvisionedThroughput(new ProvisionedThroughput()
                                                .withReadCapacityUnits(10L)
                                                .withWriteCapacityUnits(10L))
                                .withStreamSpecification(streamSpecification);

                System.out.println("Issuing CreateTable request for " + tableName);
                dynamoDBClient.createTable(createTableRequest);
                System.out.println("Waiting for " + tableName + " to be created...");

                try {
                        TableUtils.waitUntilActive(dynamoDBClient, tableName);
                } catch (AmazonClientException e) {
                        e.printStackTrace();
                }

                // Print the stream settings for the table
                DescribeTableResult describeTableResult =
 dynamoDBClient.describeTable(tableName);
                String streamArn = describeTableResult.getTable().getLatestStreamArn();
                System.out.println("Current stream ARN for " + tableName + ": " +
                                describeTableResult.getTable().getLatestStreamArn());
                StreamSpecification streamSpec =
 describeTableResult.getTable().getStreamSpecification();
                System.out.println("Stream enabled: " + streamSpec.getStreamEnabled());
                System.out.println("Update view type: " +
 streamSpec.getStreamViewType());
                System.out.println();

                // Generate write activity in the table

                System.out.println("Performing write activities on " + tableName);
                int maxItemCount = 100;
                for (Integer i = 1; i <= maxItemCount; i++) {
                        System.out.println("Processing item " + i + " of " +
 maxItemCount);

                        // Write a new item
                        Map<String, AttributeValue> item = new HashMap<>();
                        item.put("Id", new AttributeValue().withN(i.toString()));
                        item.put("Message", new AttributeValue().withS("New item!"));
                        dynamoDBClient.putItem(tableName, item);

                        // Update the item
                        Map<String, AttributeValue> key = new HashMap<>();
                        key.put("Id", new AttributeValue().withN(i.toString()));
                        Map<String, AttributeValueUpdate> attributeUpdates = new
 HashMap<>();
                        attributeUpdates.put("Message", new AttributeValueUpdate()
                                        .withAction(AttributeAction.PUT)
                                        .withValue(new AttributeValue()
                                                        .withS("This item has 
 changed")));
                        dynamoDBClient.updateItem(tableName, key, attributeUpdates);

                        // Delete the item
                        dynamoDBClient.deleteItem(tableName, key);
                }

                // Get all the shard IDs from the stream.  Note that DescribeStream 
 returns
                // the shard IDs one page at a time.
                String lastEvaluatedShardId = null;

                do {
                        DescribeStreamResult describeStreamResult = 
 streamsClient.describeStream(
                                        new DescribeStreamRequest()
                                                        .withStreamArn(streamArn)
                                                       
 .withExclusiveStartShardId(lastEvaluatedShardId));
                        List<Shard> shards = 
 describeStreamResult.getStreamDescription().getShards();

                        // Process each shard on this page

                        for (Shard shard : shards) {
                                String shardId = shard.getShardId();
                                System.out.println("Shard: " + shard);

                                // Get an iterator for the current shard

                                GetShardIteratorRequest getShardIteratorRequest = new 
 GetShardIteratorRequest()
                                                .withStreamArn(streamArn)
                                                .withShardId(shardId)
                                               
 .withShardIteratorType(ShardIteratorType.TRIM_HORIZON);
                                GetShardIteratorResult getShardIteratorResult = 
 streamsClient
                                               
 .getShardIterator(getShardIteratorRequest);
                                String currentShardIter = 
 getShardIteratorResult.getShardIterator();
```

```
                               // Shard iterator is not null until the Shard is sealed
(marked as READ_ONLY).
                               // To prevent running the loop until the Shard is
sealed, which will be on
                               // average
                               // 4 hours, we process only the items that were written
into DynamoDB and then
                               // exit.
                               int processedRecordCount = 0;
                               while (currentShardIter != null && processedRecordCount
< maxItemCount) {
                                       System.out.println("    Shard iterator: " +
currentShardIter.substring(380));

                                       // Use the shard iterator to read the stream
records

                                       GetRecordsResult getRecordsResult =
streamsClient
                                                       .getRecords(new
GetRecordsRequest()

.withShardIterator(currentShardIter));
                                       List<Record> records =
getRecordsResult.getRecords();
                                       for (Record record : records) {
                                               System.out.println("        " +
record.getDynamodb());
                                       }
                                       processedRecordCount += records.size();
                                       currentShardIter =
getRecordsResult.getNextShardIterator();
                               }
                       }

                       // If LastEvaluatedShardId is set, then there is
                       // at least one more page of shard IDs to retrieve
                       lastEvaluatedShardId =
describeStreamResult.getStreamDescription().getLastEvaluatedShardId();

               } while (lastEvaluatedShardId != null);

               // Delete the table
               System.out.println("Deleting the table...");
```

```
                dynamoDBClient.deleteTable(tableName);

                System.out.println("Demo complete");

        }
}
```

DynamoDB Streams と AWS Lambda のトリガー

トピック

- チュートリアル #1: AWS CLI を使用した Amazon DynamoDB と AWS Lambda での、フィルターを使ったすべてのイベントの処理
- チュートリアル #2: Amazon DynamoDB と Lambda での、フィルターを使用したいくつかのイベントの処理。
- Lambda を使用したベストプラクティス

Amazon DynamoDB は AWS Lambda と統合されているため、トリガー (DynamoDB Streams 内のイベントに自動的に応答するコードの一部) を作成できます。トリガーを使用すると、DynamoDB テーブル内のデータ変更に対応するアプリケーションを構築できます。

テーブルで DynamoDB Streams を有効にした場合、書き込む AWS Lambda 関数にストリーミングの Amazon リソースネーム (ARN) を関連付けることができます。その後、その DynamoDB テーブルに対するすべてのミューテーションアクションをストリーム上の項目としてキャプチャできます。例えば、テーブル内の項目が変更されたときに、そのテーブルのストリームに新しいレコードがすぐに表示されるようにトリガーを設定できます。

> **Note**
>
> 3 つ以上の Lambda 関数をサブスクライブできます。3 つ以上の Lambda 関数を 1 つの DynamoDB ストリームにサブスクライブすると、読み込みスロットリングが発生する可能性があります。

AWS Lambda サービスは、1 秒に 4 回、ストリームをポーリングして新しいレコードを探します。新しいストリームレコードが利用可能になると、Lambda 関数が同期的に呼び出されます。同じ DynamoDB ストリームに最大 2 つの Lambda 関数をサブスクライブできます。同じ DynamoDB ストリームに 3 つ以上の Lambda 関数をサブスクライブすると、読み込みスロットリングが発生する可能性があります。

Lambda 関数は、通知の送信やワークフローの開始など、指定した他の多くのアクションを実行できます。各ストリームレコードを Amazon S3 File Gateway (Amazon S3) などの永続的ストレージにコピーするだけで、書き込みアクティビティの永続的な監査追跡をテーブルに作成する Lambda 関数を記述できます。または、GameScores テーブルに書き込みを行うモバイルゲームアプリがあるとします。TopScore テーブルの GameScores 属性が更新されるたびに、対応するストリームレコードがテーブルのストリームに書き込まれます。その後、このイベントによって Lambda 関数をトリガーし、ソーシャルメディアネットワークにおめでとうメッセージを投稿できます この関数は、GameScores の更新でないストリームレコードや、TopScore 属性を変更しないストリームレコードを無視するように記述できます。

関数がエラーを返した場合、処理が正常に完了するか、データの有効期限が切れるまで Lambda はバッチを再試行します。Lambda では、より小さなバッチで再試行したり、再試行回数を制限したり、古すぎるレコードを破棄したりするよう設定できます。

パフォーマンスのベストプラクティスとして、Lambda 関数は存続期間を短くする必要があります。また、不必要な処理の遅延が発生するのを防ぐため、複雑なロジックは実行しないでください。特に高速ストリームの場合は、同期的に長時間実行する Lambda よりも、非同期的な後処理ステップ関数ワークフローをトリガーすることをお勧めします。

同じ Lambda トリガーを複数の異なる AWS アカウント間で使用することはできません。DynamoDB テーブルと Lambda 関数の両方が、同じ AWS アカウントに属している必要があります。

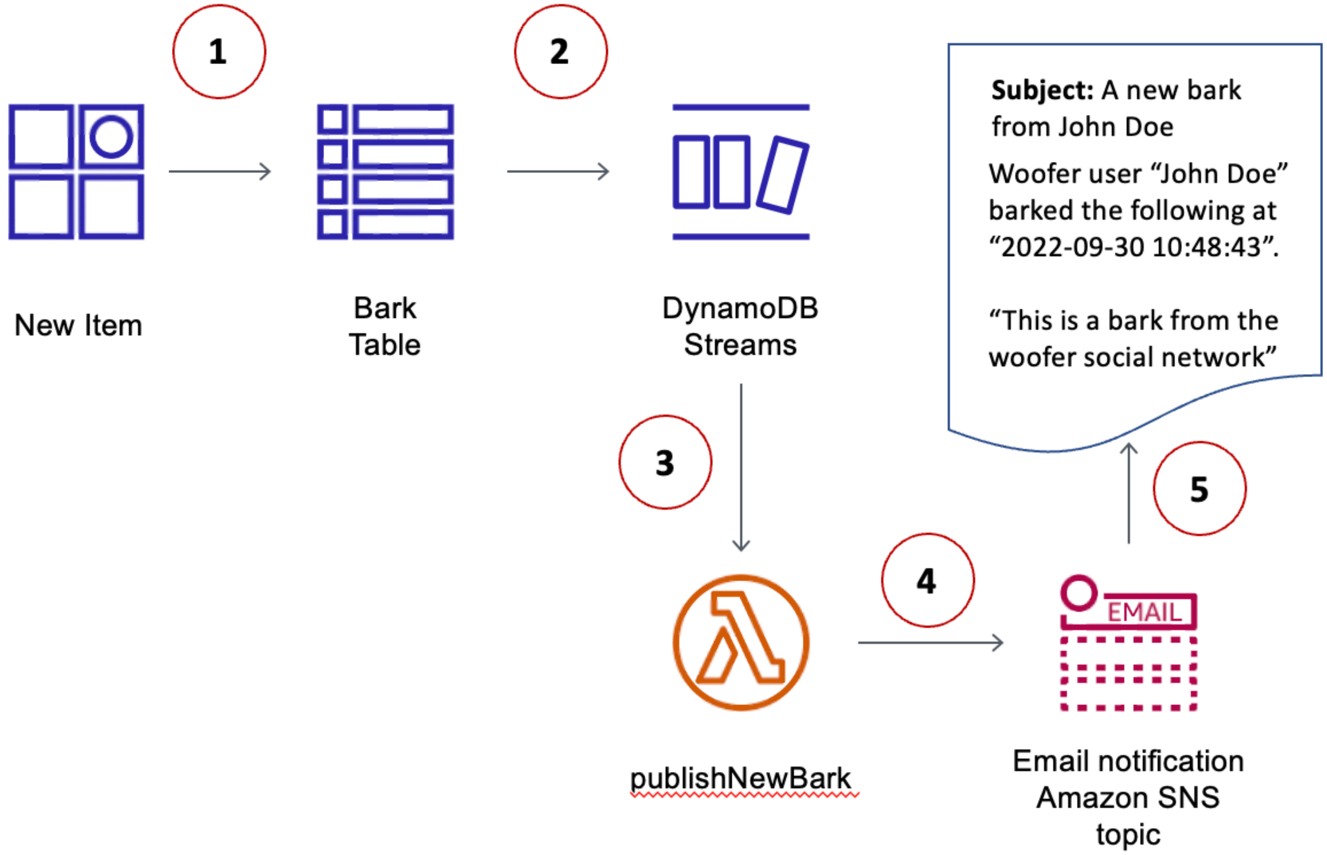
AWS Lambda の詳細については、「AWS Lambda デベロッパーガイド」を参照してください。

チュートリアル #1: AWS CLI を使用した Amazon DynamoDB と AWS Lambda での、フィルターを使ったすべてのイベントの処理

トピック

このチュートリアルでは、AWS Lambda トリガーを作成して、DynamoDB テーブルからのストリーミングを処理します。

このチュートリアルのシナリオは、シンプルなソーシャルネットワークである Woofer です。Woofer ユーザーは、他の Woofer ユーザーに送信される bark (短いテキストメッセージ) を使用して通信します。次の図は、このアプリケーションのコンポーネントとワークフローを示しています。


1
2
New Item
Bark Table
DynamoDB Streams
Subject: A new bark from John Doe
Woofer user "John Doe" barked the following at "2022-09-30 10:48:43".
"This is a bark from the woofer social network"
3
4
5
EMAIL
publishNewBark
Email notification Amazon SNS topic

1. ユーザーは DynamoDB テーブル (`BarkTable`) に項目を書き込みます。テーブルの各項目は bark を表します。
2. 新しいストリームレコードが書き込まれ、新しい項目が `BarkTable` に追加されたことを反映します。
3. 新しいストリームレコードは AWS Lambda 関数 (`publishNewBark`) をトリガーします。
4. ストリーミングレコードに、新しい項目が `BarkTable` に追加されたことが示された場合、Lambda 関数はストリーミングレコードからデータを読み込み、Amazon Simple Notification Service (Amazon SNS) のトピックにメッセージを発行します。
5. メッセージは Amazon SNS トピックの受信者によって受信されます (このチュートリアルでは、唯一の受信者は E メールアドレスです)。

開始する前に

このチュートリアルでは AWS Command Line Interface AWS CLI を使用します。まだ行っていない場合は、「AWS Command Line Interface ユーザーガイド」の手順に従って、AWS CLI をインストールおよび設定します。

ステップ 1: ストリーミングが有効になった DynamoDB テーブルを作成する

このステップでは、Woofer ユーザーからのすべての bark を保存する DynamoDB テーブル (`BarkTable`) を作成します。プライマリキーは、`Username` (パーティションキー) と `Timestamp` (ソートキー) で構成されます。これらの属性は両方とも文字列型になります。

`BarkTable` ではストリームが有効になっています。このチュートリアルの後半では、AWS Lambda 関数をストリームと関連付けてトリガーを作成します。

1. 次のコマンドを入力して、テーブルを作成します。

```
aws dynamodb create-table \
    --table-name BarkTable \
    --attribute-definitions AttributeName=Username,AttributeType=S
 AttributeName=Timestamp,AttributeType=S \
    --key-schema AttributeName=Username,KeyType=HASH
 AttributeName=Timestamp,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

2. 出力で、`LatestStreamArn` を探します。

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

   `region` と *`accountID`* をメモしておきます。これらは、このチュートリアルの他のステップで必要になります。

ステップ 2: Lambda 実行ロールを作成する

このステップでは、AWS Identity and Access Management (IAM) ロール (`WooferLambdaRole`) を作成し、それにアクセス権限を割り当てます。このロールは、ステップ 4: Lambda 関数を作成してテストする で作成する Lambda 関数で使用されます。

また、ロールのポリシーを作成します。このポリシーには、Lambda 関数がランタイム時に必要とするすべてのアクセス許可が含まれます。

1. 次の内容で、`trust-relationship.json` というファイルを作成します。

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Principal": {
         "Service": "lambda.amazonaws.com"
       },
       "Action": "sts:AssumeRole"
     }
   ]
 }
```

2. `WooferLambdaRole` を作成するため、以下のコマンドを入力します。

```
aws iam create-role --role-name WooferLambdaRole \
    --path "/service-role/" \
    --assume-role-policy-document file://trust-relationship.json
```

3. 次の内容で、`role-policy.json` というファイルを作成します。(*region* および *accountID* をお客様の AWS リージョンとアカウント ID に置き換えます。)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:region:accountID:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:region:accountID:table/BarkTable/stream/
*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

ポリシーには 4 つのステートメントがあり、これにより `WooferLambdaRole` は以下を実行できます。

- Lambda 関数 (`publishNewBark`) を実行します。このチュートリアルの後半で、この関数を作成します。
- Amazon CloudWatch Logs のアクセス。Lambda 関数はランタイム時に診断を CloudWatch Logs に書き込みます。
- `BarkTable` の DynamoDB Streams からデータを読み込みます。
- Amazon SNS にメッセージを公開します。

4. 次のコマンドを入力して、`WooferLambdaRole` にポリシーをアタッチします。

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

ステップ 3: Amazon SNS トピックを作成する

このステップでは、Amazon SNSトピック (`wooferTopic`) を作成し、そのトピックに E メールアドレスをサブスクライブします。Lambda 関数はこのトピックを使用して、Woofer ユーザーからの新しい bark を公開します。

1. 次のコマンドを入力して、新しい Amazon SNS トピックを作成します。

```
aws sns create-topic --name wooferTopic
```

2. 次のコマンドを入力して、wooferTopic に E メールアドレスをサブスクライブします (*region* および *accountID* は AWS リージョンとアカウント ID に置き換え、*example@example.com* は有効な E メールアドレスと置き換えます)。

```
aws sns subscribe \
    --topic-arn arn:aws:sns:region:accountID:wooferTopic \
    --protocol email \
    --notification-endpoint example@example.com
```

3. Amazon SNS は E メールアドレスに確認メッセージを送信します。そのメッセージの [サブスクリプションを確認] リンクを選択して、サブスクリプションプロセスを完了します。

ステップ 4: Lambda 関数を作成してテストする

このステップでは、AWS Lambda 関数 (publishNewBark) を作成して BarkTable からのストリームレコードを処理します。

publishNewBark 関数は、BarkTable の新しい項目に対応するストリームイベントのみを処理します。この関数は、そのようなイベントからデータを読み取ってから、Amazon SNS をコールしてデータを公開します。

1. 次の内容で、publishNewBark.js というファイルを作成します。*region* および *accountID* をお客様の AWS リージョンとアカウント ID に置き換えます。

```
'use strict';
var AWS = require("aws-sdk");
var sns = new AWS.SNS();

exports.handler = (event, context, callback) => {

    event.Records.forEach((record) => {
        console.log('Stream record: ', JSON.stringify(record, null, 2));

        if (record.eventName == 'INSERT') {
            var who = JSON.stringify(record.dynamodb.NewImage.Username.S);
            var when = JSON.stringify(record.dynamodb.NewImage.Timestamp.S);
            var what = JSON.stringify(record.dynamodb.NewImage.Message.S);
            var params = {
                Subject: 'A new bark from ' + who,
                Message: 'Woofer user ' + who + ' barked the following at ' + when
 + ':\n\n ' + what,
                TopicArn: 'arn:aws:sns:region:accountID:wooferTopic'
            };
            sns.publish(params, function(err, data) {
                if (err) {
                    console.error("Unable to send message. Error JSON:",
 JSON.stringify(err, null, 2));
                } else {
                    console.log("Results from sending message: ",
 JSON.stringify(data, null, 2));
                }
            });
        }
    });
    callback(null, `Successfully processed ${event.Records.length} records.`);
};
```

2. publishNewBark.js を含める zip ファイルを作成します。zip コマンドラインユーティリティがある場合は、次のコマンドを入力してこれを行うことができます。

```
zip publishNewBark.zip publishNewBark.js
```

3. Lambda 関数を作成する場合は、ステップ 2: Lambda 実行ロールを作成する で作成した WooferLambdaRole の Amazon リソースネーム (ARN) を指定します。この ARN を取得するには、次のコマンドを入力します。

```
aws iam get-role --role-name WooferLambdaRole
```

出力で、WooferLambdaRole の ARN を探します。

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
...
```

次のコマンドを入力して、Lambda 関数を作成します。*roleARN* を WooferLambdaRole の ARN に置き換えます。

```
aws lambda create-function \
    --region region \
    --function-name publishNewBark \
    --zip-file fileb://publishNewBark.zip \
    --role roleARN \
    --handler publishNewBark.handler \
    --timeout 5 \
    --runtime nodejs16.x
```

4. ここで、publishNewBark をテストして、これが動作することを確認します。これを行うには、DynamoDB Streams の実際のレコードに似た情報を入力します。

   次の内容で、payload.json というファイルを作成します。*region* および *accountID* をお客様の AWS リージョンとアカウント ID に置き換えます。

```
{
    "Records": [
        {
            "eventID": "7de3041dd709b024af6f29e4fa13d34c",
            "eventName": "INSERT",
            "eventVersion": "1.1",
            "eventSource": "aws:dynamodb",
            "awsRegion": "region",
            "dynamodb": {
                "ApproximateCreationDateTime": 1479499740,
                "Keys": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "NewImage": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Message": {
                        "S": "This is a bark from the Woofer social network"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "SequenceNumber": "1302160000000001596893679",
                "SizeBytes": 112,
                "StreamViewType": "NEW_IMAGE"
            },
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

次のコマンドを入力して、publishNewBark 関数をテストします。

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

テストが成功すると、次の出力が表示されます。

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

さらに、output.txt ファイルには次のテキストが含まれます。

```
"Successfully processed 1 records."
```

また、数分以内に新しい E メールメッセージが届きます。

> **Note**
>
> AWS Lambda は、Amazon CloudWatch Logs に診断情報を書き込みます。Lambda 関数でエラーが発生した場合、この診断情報をトラブルシューティングに使用できます。
>
> 1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
> 2. ナビゲーションペインで [ログ] を選択します。
> 3. 次のロググループを選択: /aws/lambda/publishNewBark

4. 最新のログストリーミングを選択して、関数からの出力 (およびエラー) を表示します。

ステップ 5: トリガーを作成してテストする

ステップ 4: Lambda 関数を作成してテストする で、Lambda 関数をテストして、正しく実行されたことを確認しました。このステップでは、Lambda 関数 (publishNewBark) をイベントソース (BarkTable ストリーミング) に関連付けることでトリガーを作成します。

1. トリガーを作成する場合、BarkTable ストリーム用の ARN を指定する必要があります。この ARN を取得するには、次のコマンドを入力します。

```
aws dynamodb describe-table --table-name BarkTable
```

   出力で、LatestStreamArn を探します。

```
...
 "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

2. 次のコマンドを入力してトリガーを作成します *streamARN* を実際のストリーム ARN に置き換えます。

```
aws lambda create-event-source-mapping \
    --region region \
    --function-name publishNewBark \
    --event-source streamARN  \
    --batch-size 1 \
    --starting-position TRIM_HORIZON
```

3. トリガーをテストします。次のコマンドを入力して、BarkTable に項目を追加します。

```
aws dynamodb put-item \
    --table-name BarkTable \
    --item Username={S="Jane
 Doe"},Timestamp={S="2016-11-18:14:32:17"},Message={S="Testing...1...2...3"}
```

   数分以内に新しい E メールメッセージが届きます。

4. DynamoDB コンソールを開き、さらにいくつかの項目を `BarkTable` に追加します。`Username` および `Timestamp` 属性の値を指定する必要があります (必須ではないものの、`Message` の値を指定する必要があります)。`BarkTable` に追加した各項目について、新しい E メールメッセージが届きます。

   Lambda 関数は、`BarkTable` に追加した新しい項目のみを処理します。テーブル内の項目を更新または削除すると、この関数は何も行いません。

> Note
>
> AWS Lambda は、Amazon CloudWatch Logs に診断情報を書き込みます。Lambda 関数でエラーが発生した場合、この診断情報をトラブルシューティングに使用できます。
>
> 1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
> 2. ナビゲーションペインで [ログ] を選択します。
> 3. 次のロググループを選択: `/aws/lambda/publishNewBark`
> 4. 最新のログストリーミングを選択して、関数からの出力 (およびエラー) を表示します。

チュートリアル #2: Amazon DynamoDB と Lambda での、フィルターを使用したいくつかのイベントの処理。

トピック

このチュートリアルでは、AWS Lambda トリガーを作成して、DynamoDB テーブルからのストリームの一部のイベントのみを処理します。

Lambda イベントフィルタリングでは、フィルター式を使用して、処理のために Lambda が関数に送信するイベントを制御できます。DynamoDB ストリームごとに最大 5 つの異なるフィルターを設定できます。バッチ処理ウィンドウを使用している場合、Lambda は新しいイベントそれぞれにフィルター条件を適用して、現在のバッチに含めるかどうかを確認します。

フィルターは `FilterCriteria` と呼ばれる構造を介して適用されます。`FilterCriteria` の 3 つの主な属性は `metadata properties`、`data properties`、および `filter patterns` です。

DynamoDB Streams イベントの構造の例を次に示します。

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

`metadata properties` は、イベントオブジェクトのフィールドです。DynamoDB Streams の場合、`metadata properties` は `dynamodb` や `eventName` のようなフィールドです。

`data properties` は、イベント本文のフィールドです。`data properties` をフィルタリングするには、それらを適切なキー内の `FilterCriteria` に含めるようにしてください。DynamoDB イベントソースのデータキーは `NewImage` または `OldImage` です。

最後に、フィルタールールは、特定のプロパティに適用するフィルター式を定義します。次に例を示します。

Comparison operator (比較演算子)	例	ルール構文 (一部)
Null	製品タイプは NULL	`{ "product_type": { "S": null } }`
空	製品名は空白	`{ "product_name": { "S": [ ""] } }`
等しい	州はフロリダに等しい	`{ "state": { "S": ["FL"] } }`
And	製品州はフロリダ、製品カテゴリはチョコレート	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
または	製品州はフロリダまたはカリフォルニア	`{ "state": { "S": ["FL","CA"] } }`
Not	製品州はフロリダ州ではない	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
存在する	地産品は存在する	`{"homemade": {"S": [{"exists": true}]}}`
存在しない	地産品は存在しない	`{"homemade": {"S": [{"exists": false}]}}`
で始まる	PK は COMPANY から始まる	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

Lambda 関数には、最大 5 つのイベントフィルタリングパターンを指定できます。これら 5 つのイベントのそれぞれが論理 OR として評価されることに注意してください。そのため、`Filter_One` および `Filter_Two` という名前の 2 つのフィルターを設定すると、Lambda 関数は `Filter_One` OR `Filter_Two` を実行します。

> Note
>
> Lambda イベントのフィルタリングページには、数値をフィルタリングして比較するオプションがいくつかありますが、DynamoDB のフィルタイベントの場合、DynamoDB の数値は文字列として保存されるため、適用されません。例えば `"quantity": { "N": "50" }` の場合は、`"N"` プロパティのおかげでそれが数字だとわかります。

すべてをまとめる - AWS CloudFormation

実際のイベントフィルタリング機能を見ていただくために、CloudFormation テンプレートのサンプルを以下に示します。このテンプレートは、Amazon DynamoDB Streams を有効にしたパーティションキー PK とソートキー SK を含むシンプルな DynamoDB テーブルを生成します。これにより、Amazon Cloudwatch へのログの書き込みと Amazon DynamoDB Streams からのイベントの読み取りを許可する Lambda 関数とシンプルな Lambda 実行ロールが作成されます。また、DynamoDB Streams と Lambda 関数間にイベントソースマッピングも追加されるため、Amazon DynamoDB Streams にイベントが発生するたびに関数を実行できます。

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Sample application that presents AWS Lambda event source filtering
with Amazon DynamoDB Streams.

Resources:
  StreamsSampleDDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: "PK"
          AttributeType: "S"
        - AttributeName: "SK"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "PK"
          KeyType: "HASH"
        - AttributeName: "SK"
          KeyType: "RANGE"
      StreamSpecification:
        StreamViewType: "NEW_AND_OLD_IMAGES"
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5

  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
        - PolicyName: root
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - logs:CreateLogGroup
                  - logs:CreateLogStream
                  - logs:PutLogEvents
                Resource: arn:aws:logs:*:*:*
              - Effect: Allow
                Action:
                  - dynamodb:DescribeStream
                  - dynamodb:GetRecords
                  - dynamodb:GetShardIterator
                  - dynamodb:ListStreams
                Resource: !GetAtt StreamsSampleDDBTable.StreamArn

  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST

  ProcessEventLambda:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      Timeout: 300
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import logging

          LOGGER = logging.getLogger()
          LOGGER.setLevel(logging.INFO)

          def handler(event, context):
            LOGGER.info('Received Event: %s', event)
            for rec in event['Records']:
              LOGGER.info('Record: %s', rec)

Outputs:
  StreamsSampleDDBTable:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.Arn
  StreamARN:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.StreamArn
```

この CloudFormation テンプレートをデプロイすると、次の Amazon DynamoDB 項目を挿入できます。

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

この CloudFormation テンプレートにインラインで組み込まれているシンプルな Lambda 関数により、Lambda 関数の Amazon CloudWatch ロググループのイベントは次のように表示されます。

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

フィルター例

- 特定の州に一致する商品のみ

この例では、CloudFormation テンプレートを変更して、フロリダで生産されたすべての製品 (略称「FL」) と一致するフィルターを含めます。

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

スタックを再デプロイしたら、テーブルに次の DynamoDB 項目を追加できます。この例での製品はカリフォルニア産なので、Lambda 関数ログには表示されないことに注意してください。

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK#1000",
 "company_id": "1000",
 "fabric": "Florida Chocolates",
 "price": 15,
 "product_id": "1000",
 "quantity": 50,
 "state": "CA",
 "stores": 5,
 "type": ""
}
```

- PK と SK のうち、ある値で始まるアイテムのみ

この例では、CloudFormation テンプレートを変更して次の条件を含めます。

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
 "COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

AND 条件では、条件がパターン内になければならないことに注意してください。ここで、キー PK と SK は同じ式内にあり、カンマで区切られます。

PK と SK がある値から始まるか、特定の州産かのどちらかです。

この例では、CloudFormation テンプレートを変更して次の条件を含めます。

```
  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix": 
"COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

フィルターセクションに新しいパターンを導入することで、OR 条件が追加されます。

すべてをまとめる - CDK

次のサンプル CDK プロジェクト形成テンプレートでは、イベントフィルタリング機能について説明します。この CDK プロジェクトに取り組む前に、準備スクリプトの実行を含む前提条件をインストールする必要があります。

CDK プロジェクトを作成する

まず、空のディレクトリで cdk init を呼び出して、新しい AWS CDK プロジェクトを作成します。

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

この cdk init コマンドは、プロジェクトフォルダの名前を使用して、クラス、サブフォルダ、ファイルなど、プロジェクトのさまざまな要素に名前を付けます。フォルダ名に含まれるハイフンはすべてアンダースコアに変換されます。それ以外の場合、名前は Python 識別子の形式に従う必要があります。例えば、数字で始めたり、スペースを含めたりはしないでください。

新しいプロジェクトで作業するには、その仮想環境を有効にします。これにより、プロジェクトの依存関係をグローバルにインストールするのではなく、プロジェクトフォルダにローカルにインストールできます。

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> これは、仮想環境をアクティブにする Mac/Linux コマンドとして認識されるかもしれません。Python テンプレートには、同じコマンドを Windows で使用できるようにするバッチファイル、`source.bat` が含まれています。従来の Windows コマンド、`.venv\Scripts\activate.bat` も機能します。AWS CDK Toolkit v1.70.0 以前を使用して AWS CDK プロジェクトを初期化した場合、仮想環境は `.venv` ではなく `.env` ディレクトリにあります。

基本インフラストラクチャ

任意のテキストエディタでファイル `./ddb_filters/ddb_filters_stack.py` を開きます。このファイルは、AWS CDK プロジェクトを作成したときに自動生成されました。

次に、`_create_ddb_table` および `_set_ddb_trigger_function` 関数を追加します。これらの関数は、プロビジョニングモードのオンデマンドモードでパーティションキー PK とソートキー SK を含む DynamoDB テーブルを作成します。Amazon DynamoDB Streams をデフォルトで有効にして、新しいイメージと古いイメージを表示できます。

Lambda 関数はファイル `app.py` の下のフォルダ `lambda` に保存されます。このファイルは後で作成されます。これには環境変数 `APP_TABLE_NAME` が含まれます。この変数は、このスタックによって作成される Amazon DynamoDB テーブルの名前になります。同じ関数で、Lambda 関数にストリーム読み取り権限を付与します。最後に、Lambda 関数のイベントソースとして DynamoDB Streams にサブスクライブします。

`__init__` メソッド内のファイルの最後で、それぞれの構成を呼び出してスタック内で初期化します。追加のコンポーネントやサービスを必要とする大規模なプロジェクトでは、これらの構成を基本スタックの外部で定義するのが最適な場合があります。

```
import os
import json

import aws_cdk as cdk
```

```
from aws_cdk import (
    Stack,
    aws_lambda as _lambda,
    aws_dynamodb as dynamodb,
)
from constructs import Construct


class DdbFiltersStack(Stack):

    def _create_ddb_table(self):
        dynamodb_table = dynamodb.Table(
            self,
            "AppTable",
            partition_key=dynamodb.Attribute(
                name="PK", type=dynamodb.AttributeType.STRING
            ),
            sort_key=dynamodb.Attribute(
                name="SK", type=dynamodb.AttributeType.STRING),
            billing_mode=dynamodb.BillingMode.PAY_PER_REQUEST,
            stream=dynamodb.StreamViewType.NEW_AND_OLD_IMAGES,
            removal_policy=cdk.RemovalPolicy.DESTROY,
        )

        cdk.CfnOutput(self, "AppTableName", value=dynamodb_table.table_name)
        return dynamodb_table

    def _set_ddb_trigger_function(self, ddb_table):
        events_lambda = _lambda.Function(
            self,
            "LambdaHandler",
            runtime=_lambda.Runtime.PYTHON_3_9,
            code=_lambda.Code.from_asset("lambda"),
            handler="app.handler",
            environment={
                "APP_TABLE_NAME": ddb_table.table_name,
            },
        )

        ddb_table.grant_stream_read(events_lambda)

        event_subscription = _lambda.CfnEventSourceMapping(
            scope=self,
            id="companyInsertsOnlyEventSourceMapping",
            function_name=events_lambda.function_name,
            event_source_arn=ddb_table.table_stream_arn,
            maximum_batching_window_in_seconds=1,
            starting_position="LATEST",
            batch_size=1,
        )

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ddb_table = self._create_ddb_table()
        self._set_ddb_trigger_function(ddb_table)
```

次に、Amazon CloudWatch にログを出力する、非常にシンプルな Lambda 関数を作成します。それには、`lambda` という新しいフォルダを作成します。

```
mkdir lambda
touch app.py
```

任意のテキストエディタを使用して、次の内容を `app.py` ファイルに追加します。

```
import logging

LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)


def handler(event, context):
    LOGGER.info('Received Event: %s', event)
    for rec in event['Records']:
        LOGGER.info('Record: %s', rec)
```

`/ddb_filters/` フォルダにいることを確認し、次のコマンドを入力してサンプルアプリケーションを作成します。

```
cdk deploy
```

ある時点で、ソリューションをデプロイするかどうか確認する画面が表示されます。Y を入力して変更を確定します。

```
###############################################################################################
```

```
# + # ${LambdaHandler/ServiceRole} # arn:${AWS::Partition}:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole #
################################################################################################

Do you wish to deploy these changes (y/n)? y

...

#  Deployment time: 67.73s

Outputs:
DdbFiltersStack.AppTableName = DdbFiltersStack-AppTable815C50BC-1M1W7209V5YPP
Stack ARN:
arn:aws:cloudformation:us-east-2:111122223333:stack/
DdbFiltersStack/66873140-40f3-11ed-8e93-0a74f296a8f6
```

変更がデプロイされたら、AWS コンソールを開いてテーブルに項目を 1 つ追加します。

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

これで CloudWatch ログには、このエントリの情報がすべて含まれているはずです。

フィルター例

- 特定の州に一致する商品のみ

ファイル ddb_filters/ddb_filters/ddb_filters_stack.py を開き、「FL」に等しいすべての製品と一致するフィルターを含めるように変更します。これは 45 行目の event_subscription のすぐ下で修正できます。

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

- PK と SK のうち、ある値で始まる項目のみ

Python スクリプトを次の条件を含むように変更します。

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- PK と SK がある値から始まるか、特定の州産かのどちらかです。

Python スクリプトを次の条件を含むように変更します。

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

Filters 配列に要素を追加することで OR 条件が追加されることに注意してください。

クリーンアップ

作業ディレクトリのベースにあるフィルタースタックを見つけて、cdk destroy を実行します。リソースの削除を確認するメッセージが表示されます。

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Lambda を使用したベストプラクティス

AWS Lambda 関数は、他の関数から分離された実行環境であるコンテナ内で実行されます。この関数を初めて実行すると、AWS Lambda は新しいコンテナを作成し、関数のコードを実行し始めます。

Lambda 関数には、コールごとに 1 回実行されるハンドラがあります。ハンドラには、関数用の主要なビジネスロジックが含まれます。たとえば、ステップ 4: Lambda 関数を作成してテストする に示す Lambda 関数には、DynamoDB Streams のレコードを処理できるハンドラがあります。

コンテナの作成後、AWS Lambda が初めてハンドラーを実行する前に、1 回だけ実行される初期化コードを提供することもできます。ステップ 4: Lambda 関数を作成してテストする に示す Lambda 関数には、SDK for JavaScript in Node をインポートし、Amazon SNS 用のクライアントを作成する初期化コードがあります。これらのオブジェクトはハンドラの外部で 1 回のみ定義します。

関数の実行後、AWS Lambda は関数のそれ以降の呼び出しに対してコンテナを再利用する場合があります。この場合、関数ハンドラは、初期化コードで定義したリソースを再利用できる可能性があります (AWS Lambda がコンテナを保持する期間や、コンテナを再利用するかどうかを制御することはできません)。

AWS Lambda を使用した DynamoDB トリガーの場合は、次のことをお勧めします。

- AWS のサービスのクライアントは、ハンドラではなく初期化コードでインスタンス化する必要があります。これにより、AWS Lambda コンテナは、コンテナの有効期間中は既存の接続を再利用することができます。
- 通常、お客様が明示的に接続を管理したり、接続プールを実装したりする必要はありません。これは AWS Lambda によって自動的に管理されます。

DynamoDB ストリームの Lambda コンシューマーは、正確に一度だけ配信されることを保証するものではなく、時折重複が発生する可能性があります。重複処理が原因で予期しない問題が発生しないように、Lambda 関数コードは必ずべき等性にしてください。

詳細については、「AWS Lambda デベロッパーガイド」の「AWS Lambda 関数を操作するためのベストプラクティス」を参照してください。

DynamoDB のオンデマンドバックアップおよび復元の使用

DynamoDB オンデマンドバックアップ機能を使用し、テーブルの完全なバックアップを作成して、長期保存し、規制コンプライアンスの要件に合わせてアーカイブすることができます。テーブルのデータは、AWS Management Console でワンクリックするか、単一の API コールを使用するだけで、バックアップおよび復元することができます。バックアップや復元のアクションがテーブルのパフォーマンスや可用性に影響を及ぼすことはありません。

次の動画では、バックアップと復元の概念を紹介します。

バックアップと復元

DynamoDB オンデマンドバックアップの作成と管理に使用できるオプションは 2 つあります。

- AWS Backup サービス
- DynamoDB

AWS Backup を使用すると、バックアップポリシーを設定し、AWS リソースとオンプレミスワークロードのアクティビティを 1 か所でモニタリングできます。DynamoDBをAWS Backup で使用すると、AWS アカウントとリージョン間でオンデマンドバックアップをコピーし、オンデマンドバックアップにコスト分配タグを追加し、オンデマンドバックアップをコールドストレージに移行してコストを削減できます。これらの高度な機能を使用するには、AWS Backup に オプトインします。オプトインの選択は特定のアカウントと AWS リージョンに適用されるため、同じアカウントを使用して複数のリージョンにオプトインする必要がある場合があります。詳細については、AWS Backup デベロッパーガイドを参照してください。

オンデマンドバックアップおよび復元プロセスは、アプリケーションのパフォーマンスや可用性を低下させずにスケーリングすることができます。新しい独自の分散型技術を使用しており、テーブルのサイズに関係なく、数秒でバックアップを完了することができます。スケジュールや、長時間稼働のバックアッププロセスについて心配することなく、数千ものパーティションにわたって一貫性のあるバックアップを数秒以内に作成することができます。オンデマンドバックアップはすべて、カタログ化され、検出可能で、明示的に削除されるまで維持されます。

また、オンデマンドバックおよび復元オペレーションが、パフォーマンスや API レイテンシーに影響を及ぼすことはありません。テーブルが削除されても、バックアップは保持されます。詳細については、「DynamoDB バックアップと復元の使用」を参照してください。

DynamoDB のオンデマンドバックアップは、バックアップストレージサイズに関連付けられた通常料金でご利用になれます。追加料金はかかりません。DynamoDB オンデマンドバックアップを別のアカウントまたはリージョンにコピーすることはできません。AWS アカウントおよびリージョン間でバックアップコピーを作成したり、その他の高度な機能を利用したりする場合は、AWS Backup を使用する必要があります。AWS Backup の機能を使用する場合、AWS Backup から請求されることになります。使用可能な AWS リージョンおよび料金に関する詳細については、Amazon DynamoDB 料金表を参照してください。

トピック

DynamoDB での AWS Backup の使用

Amazon DynamoDB は、AWS Backup の強化されたバックアップ機能を通じて、規制コンプライアンスとビジネス継続性の要件を満たすのに役立ちます。AWS Backup は、フルマネージドのデータ保護サービスであり、AWS サービス間、クラウド内、およびオンプレミスでのバックアップの一元化と自動化を容易にします。このサービスを使用すると、バックアップポリシーを設定し、AWS リソースのアクティビティを 1 か所でモニタリングできます。AWS Backup を使用するには、肯定的にする必要があります。積極的にオプトインする必要があります。オプトインの選択は特定のアカウントと AWS リージョンに適用されるため、同じアカウントを使用して複数のリージョンにオプトインする必要がある場合があります。詳細については、AWSBackup デベロッパーガイドを参照してください。

Amazon DynamoDBは、AWS Backup とネイティブに統合されています。AWS Backup を使用して、DynamoDB オンデマンドバックアップを自動的にスケジュール、コピー、タグ付け、ライフサイクルを実行することができます。これらのバックアップは DynamoDB コンソールから引き続き表示および復元できます。DynamoDB コンソール、API、および AWS コマンドラインインターフェイス (AWS CLI) を使用して、DynamoDB テーブルの自動バックアップを有効にします。

> Note
>
> DynamoDB を介して作成されたバックアップは変更されません。バックアップを作成するのに、現在の DynamoDB ワークフローをまだ使用できます。

AWS Backup で利用できる拡張バックアップ機能は次のとおりです。

スケジュールバックアップ - バックアッププランを使用して、DynamoDB テーブルの定期的にスケジュールするバックアップを設定できます。

クロスアカウントとクロスリージョンのコピー - バックアップを異なる AWS リージョンまたはアカウントの別のバックアップボールトに自動的にコピーできます。これにより、データ保護要件をサポートできるようになります。

コールドストレージ階層化 - バックアップを削除したり、古いストレージに移行したりするライフサイクルルールを実施するようにバックアップを設定できます。これにより、バックアップコストを最適化できます。

タグ - 請求およびコスト配分のために、バックアップに自動的にタグを付けることができます。

暗号化 — AWS Backup で管理されている DynamoDB オンデマンドバックアップが AWS Backup ボールトに保存されるようになりました。これにより、DynamoDB テーブルの暗号化キーから独立した AWS KMS key を使用してバックアップを暗号化して保護できます。

バックアップの監査 — AWS Backup Audit Manager を使用して、AWS Backup ポリシーのコンプライアンスを監査し、定義したコントロールにまだ準拠していないバックアップアクティビティとリソースを検索できます。また、バックアップガバナンスのために、日次レポートとオンデマンドレポートの監査証跡を自動的に生成することもできます。

WORM モデルを使用した安全なバックアップ — AWS Backup Vault Lock を使用して、バックアップのWrite-Once-Read-Many（WORM）設定を有効にすることができます。AWS Backup Vault Lock を使用すると、不注意または悪意のある削除操作、バックアップ保持期間の変更、ライフサイクル設定の更新からバックアップを保護する保護レイヤーを追加できます。詳細については、AWS BackupVault Lock を参照してください。

これらの拡張バックアップ機能は、すべての AWS リージョンで利用できます。これらの機能の詳細については、AWS Backup デベロッパーガイドを参照してください。

トピック

AWS Backup を使用した DynamoDB テーブルのバックアップと復元: 仕組み

オンデマンドバックアップといった特徴を使用して、Amazon DynamoDB テーブルの完全バックアップを作成できます。このセクションでは、バックアップおよび復元プロセス中の概要について示します。

バックアップ

AWS Backup を使用してオンデマンドバックアップを作成する場合、リクエストのタイムマーカーがカタログ化されます。このバックアップは、前回のテーブル全体のスナップショットへのリクエスト時間までにすべての変更を適用して、非同期的に作成されます。

オンデマンドバックアップを作成する度に、テーブルデータ全体がバックアップされます。オンデマンドバックアップの実行可能数に制限はありません。

> Note
>
> DynamoDB バックアップとは異なり、AWS Backup で作成されるバックアップは即時に行われません。

バックアップ進行中は、以下を行うことができません。

- バックアップオペレーションの一時停止またはキャンセル。
- バックアップのソーステーブルの削除。
- テーブルのバックアップ中におけるテーブルのバックアップの無効化。

AWS Backup は、自動バックアップスケジュール、保持管理、およびライフサイクル管理を提供します。これにより、カスタムスクリプトや手動プロセスが不要になります。 AWS Backup はバックアップを行い、期限がくるとそれを削除します。詳細については、AWS Backup デベロッパーガイドを参照してください。

コンソールを使用している場合、AWS Backup を使用して作成されたバックアップは、[Backup type (バックアップタイプ)] が `AWS_BACKUP` に設定された状態で [Backups (バックアップ)] タブに一覧表示されます。

> Note
>
> DynamoDB コンソールを使用して、[Backup type (バックアップタイプ)] が `AWS_BACKUP` でマークされたバックアップを削除することはできません。これらのバックアップを管理するには、AWS Backup コンソールを使用します。

バックアップを実行する方法については、「DynamoDB テーブルのバックアップ」を参照してください。

復元

テーブルは、テーブルのプロビジョニングされたスループットを消費することなく復元します。DynamoDB バックアップからテーブル全体を復元することも、送信先テーブルの設定を構成することもできます。復元を実行するときに、次のテーブル設定を変更できます。

- グローバルセカンダリインデックス (GSI)
- ローカルセカンダリインデックス (LSI)
- 請求モード
- プロビジョニングされた読み込みおよび書き込みキャパシティ
- 暗号化設定

> ⚠ Important
>
> テーブル全体の復元を実行すると、送信先テーブルには、バックアップが要求されたときに送信元テーブルが持っていたのと同じプロビジョニングされた読み込みキャパシティユニットと書き込みキャパシティユニットが設定されます。復元プロセスでは、ローカルセカンダリインデックスおよびグローバルセカンダリインデックスも復元されます。

DynamoDB テーブルデータのバックアップを別の AWS リージョンにコピーしてから、その新しいリージョンで復元することができます。AWS 商用リージョン、AWS 中国リージョン、および AWS GovCloud (米国) リージョン間でバックアップをコピーして復元できます。送信元リージョンからコピーしたデータと、送信先リージョンの新しいテーブルに復元したデータに対してのみ料金が発生します。

AWS Backupは、元のインデックスをすべて含むテーブルを復元します。

以下は、復元されたテーブルで手動で設定する必要があります。

- Auto Scaling ポリシー
- AWS Identity and Access Management (IAM) ポリシー
- Amazon CloudWatch メトリクスおよびアラーム
- タグ
- ストリーム設定
- 有効期限 (TTL) 設定
- 削除保護設定
- ポイントインタイムリカバリ (PITR) 設定

テーブルデータ全体は、バックアップから新しいテーブルにのみ復元することができます。復元されたテーブルに書き込むことができるのは、アクティブになってからです。

> Note
>
> AWS Backup 復元は非破壊的です。復元オペレーション中は既存のテーブルを上書きすることはできません。

サービスメトリクスでは、お客様のテーブル復元の 95% が 1 時間未満で完了していることを示しています。ただし、復元時間は、テーブルの構成 (テーブルのサイズ、基礎となるパーティションの数など) およびその他の関連する変数に直接関係しています。災害対策を計画する際のベストプラクティスは、平均復元完了時間を定期的に記録し、これらの時間が目標復旧時間全体にどのように影響するかを確認することです。

復元を実行する方法については、「バックアップからの DynamoDB テーブルの復元」を参照してください。

IAM ポリシーを使用してアクセスコントロールできます。詳細については、「DynamoDB バックアップおよび復元での IAM の使用」を参照してください。

バックアップおよび復元を行うコンソールと API のアクションはすべて、AWS CloudTrail にキャプチャおよび記録され、ログ記録、継続的モニタリング、監査に使用されます。

AWS Backup Backup を使用した DynamoDB テーブルのバックアップの作成

このセクションでは、AWS Backup をオンにし DynamoDB テーブルからオンデマンドおよびスケジュールでバックアップを作成する方法について説明します。

トピック

- AWS Backup 機能をオンにする
- オンデマンドバックアップ
- スケジュールバックアップ

AWS Backup 機能をオンにする

DynamoDB で使用するには、AWS Backup をオンにする必要があります。

AWS Backup をオンにするには、次の手順を実行します。

1. AWS マネジメントコンソールにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。

2. コンソールの左側のナビゲーションペインで、[バックアップ] を選択します。
3. [Backup Settings] (バックアップの設定) ウィンドウで、[Turn on] (オンにする) を選択します。
4. 確認画面が表示されます。[Turn on features] (オンにする) を選択します。

DynamoDB テーブルで AWS Backup 機能を使用できるようになりました。

オンにした AWS Backup 機能をオフにするには、次の手順を実行します。

1. AWS マネジメントコンソールにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[バックアップ] を選択します。
3. [Backup Settings] (バックアップの設定) ウィンドウで、[Turn off] (オフにする) を選択します。
4. 確認画面が表示されます。[Turn off features] (オフにする) を選択します。

AWS Backup 機能をオンまたはオフにできない場合は、AWS 管理者がこれらのアクションを実行する必要があります。

オンデマンドバックアップ

DynamoDB テーブルのオンデマンドバックアップを作成するには、次の手順を実行します。

1. AWS マネジメントコンソールにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[バックアップ] を選択します。
3. [Create backup] (バックアップの作成) を選択します。
4. ドロップダウンメニューから [Create an on-demand backup] (オンデマンドバックアップを作成) を選択します。
5. ウォームストレージおよびその他の基本機能を備えた AWS Backup によって管理されるバックアップを作成するには、[Default Settings] (デフォルト設定) を選択します。コールドストレージに移行できるバックアップを作成するか、またはAWS Backup の代わりに DynamoDB 機能を使用してバックアップを作成するには、[Customize settings] (設定のカスタマイズ) を選択します。

   代わりに以前の DynamoDB 機能を使用してバックアップを作成するには、[Customize settings] (設定のカスタマイズ) を選択し、次に [Backup with DynamoDB] (DynamoDB でバックアップ) を選択します。
6. 設定が完了したら、[Create backup] (バックアップを作成) を選択します。

スケジュールバックアップ

バックアップをスケジュールするには、次の手順を実行します。

1. AWS マネジメントコンソールにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[バックアップ] を選択します。
3. 表示されるドロップダウンメニューから [AWS Backup を使用してバックアップをスケジュール] を選択します。
4. AWS Backup に移動して、バックアップ計画を作成します。

AWS Backup を使用した DynamoDB テーブルのバックアップのコピー

現在のバックアップのコピーを作成できます。バックアップは、オンデマンドで複数の AWSアカウントAWSまたはリージョンにコピーすることも、スケジュールしたバックアッププランの一部としてあるいは自動的にコピーすることもできます。また、Amazon DynamoDB Encryption Client のクロスアカウントのコピーやクロスリージョンのコピーのシーケンスを自動化することもできます。

クロスリージョンレプリケーションは、ビジネス継続性またはコンプライアンス要件があり、本番稼働用データから最小限の距離だけ離してバックアップを保存する場合に特に役立ちます。

クロスアカウントバックアップは、運用上またはセキュリティ上の理由からバックアップを組織内の 1 つまたは複数の AWS アカウントに確実にコピーする必要がある場合に便利です。元のバックアップを誤って削除してしまった場合は、コピー先のアカウントからコピー元のアカウントにバックアップをコピーし復元できます。これを行うには、Organizations サービスで同じ組織に属する 2 つのアカウントが必要です。

コピーでは、特に指定しない限り、ソースバックアップの設定を継承します。ただし、新しいコピーの「Never」の期限切れを指定した場合は例外があります。この設定では、新しいコピーはソースの有効期限を継承します。新しいバックアップコピーを永続的なコピーにする場合は、ソースバックアップを期限切れにならないように設定するか、新しいコピーの作成から 100 年後を有効期限に指定します。

> **Note**
>
> 別のアカウントにコピーする場合は、まずそのアカウントからのアクセス許可が必要です。

バックアップをコピーするには、以下の操作を行います。

1. AWS マネジメントコンソールにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[Backups] (バックアップ) を選択します。
3. コピーするバックアップの横にあるチェックボックスを選択します。
   - コピーするバックアップがグレーアウトしている場合は、AWS Backup で高度な機能を有効にする必要があります。次に、新しいバックアップを作成します。これで、この新しいバックアップを他のリージョンおよびアカウントにコピーし、今後、他の新しいバックアップをコピーできます。
4. [Copy] (コピー) を選択します。
5. バックアップを別のアカウントまたはリージョンにコピーする場合は、[Copy the recovery point to another destination] (リカバリポイントを別の送信先にコピーする) の横にあるチェックボックスをオンにします。次に、アカウント内の別のリージョンにコピーするか、別のリージョンの別のアカウントにコピーするかを選択します。

> Note
>
> バックアップを別のリージョンまたはアカウントに復元するには、まずそのリージョンまたはアカウントにバックアップをコピーしておく必要があります。

6. ファイルをコピーする目的のボールドを選択します。必要に応じて、新しいバックアップボールトを作成することもできます。
7. [Copy backup] (バックアップをコピー) を選択します。

AWS Backup からの DynamoDB テーブルのバックアップの復元。

このセクションでは、AWS Backup から DynamoDB テーブルのバックアップを復元する方法について説明します。

トピック

- AWS Backup から DynamoDB テーブルの復元
- DynamoDB テーブルを別のリージョンまたはアカウントに復元

AWS Backup から DynamoDB テーブルの復元

AWS Backup から DynamoDB テーブルを復元するには、次の手順を実行します。

1. AWS マネジメントコンソールにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[Tables] (テーブル) を選択します。
3. [Backups] (バックアップ) タブを選択します。
4. 復元したい以前のバックアップの横にあるチェックボックスを選択します。
5. [復元] を選択します。[Restore table from backup] (バックアップからのテーブルの復元) 画面が表示されます。
6. 新しく復元されたテーブルの名前、この新しいテーブルの暗号化、復元の暗号化に使用するキー、およびその他のオプションを入力します。
7. 完了したら、[Restore] (復元) を選択します。

DynamoDB テーブルを別のリージョンまたはアカウントに復元

DynamoDB テーブルを別のリージョンまたはアカウントに復元するには、まずその新しいリージョンまたはアカウントにバックアップをコピーする必要があります。別のアカウントにコピーするには、まずそのアカウントからアクセス許可を付与される必要があります。DynamoDB バックアップを新しいリージョンまたはアカウントにコピーした後、前のセクションのプロセスを使用して復元できます。

AWS Backup を使用して DynamoDB テーブルのバックアップを削除

このセクションでは、AWS Backup を使用して DynamoDB テーブルのバックアップを削除する方法について説明します。

AWS Backup の機能で作成された DynamoDB バックアップは、AWS Backup ボールトに保存されます。

この種のバックアップを削除するには、次の手順を実行します。

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[Backups (バックアップ)] を選択します。
3. 次の画面で、[AWS Backup にアクセス] を選択します。

   AWS Backup コンソール に自動的に移動します。AWS Backup コンソール でバックアップを削除する方法の詳細については、「バックアップの削除」を参照してください。

AWS Backup の詳細については、「AWS 規範ガイダンス」の「AWS Backup を使用したバックアップとリカバリ」を参照してください。

使用に関する注意事項

このセクションでは、AWS Backup が管理するオンデマンドバックアップと DynamoDB の技術的な違いについて説明します。

AWS Backup には DynamoDB とは異なるワークフローや動作があります。具体的には次のとおりです。

暗号化 - AWS Backup プランで作成されたバックアップは、AWS Backup サービスが管理するキーを使用して暗号化されたボールトに保存されます。ボールトには、セキュリティを強化するためのアクセス制御ポリシーがあります。

Backup ARN - AWS Backup が作成したバックアップファイルには AWS Backup ARN が含まれるようになり、ユーザーアクセス許可モデルに影響を与える可能性があります。バックアップリソース名 (ARN) が `arn:aws:dynamodb` から `arn:aws:backup` に変更されます。

バックアップの削除 - AWS Backup で作成したバックアップは、AWS Backup ボールトからのみ削除できます。DynamoDB コンソールから AWS Backup ファイルを削除することはできません。

バックアッププロセス - DynamoDB バックアップとは異なり、AWS Backup で作成されるバックアップは即時に行われません。

請求 - AWS Backup 機能を使用した DynamoDB テーブルのバックアップは、AWS Backup から請求されます。

IAM ロール - IAM ロールを介してアクセスを管理する場合は、次の新しいアクセス許可を使用して新しい IAM ロールを設定する必要もあります。

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob` は AWS Backup 機能を使用してバックアップを完了させるために必要で、`dynamodb:RestoreTableFromAwsBackup` は AWS Backup 機能を使用して作成されたバックアップから復元するために必要です。

完全な IAM ポリシーでこれらのアクセス許可を確認するには、「DynamoDB バックアップおよび復元での IAM の使用」の例 8 を参照してください。

DynamoDB バックアップと復元の使用

Amazon DynamoDB では、スタンドアロンのオンデマンドバックアップと復元機能をサポートしています。これらの機能は、AWS Backup を使用するかどうかに関係なく利用できます。

DynamoDB オンデマンドバックアップ機能を使用し、テーブルの完全なバックアップを作成して、規制コンプライアンスの要件を満たすために長期間の保存とアーカイブを行うことができます。テーブルのデータは、AWS マネジメントコンソールでワンクリックするか、1 回の API 呼び出しで、バックアップおよび復元することができます。バックアップおよび復元アクションを実行しても、テーブルのパフォーマンスや可用性に影響を及ぼすことはありません。

コンソール、AWS コマンドラインインターフェース (AWS CLI)、または DynamoDB API を使用してテーブルバックアップを作成できます。詳細については、「DynamoDB テーブルのバックアップ」を参照してください。

バックアップからテーブル復元する方法については、「バックアップからの DynamoDB テーブルの復元」を参照してください。

DynamoDB を使用した DynamoDB テーブルのバックアップと復元の仕組み

DynamoDB オンデマンドバックアップ機能を使用して、Amazon DynamoDB テーブルの完全バックアップを作成できます。この機能は、AWS Backup とは関係なく利用できます。このセクションでは、DynamoDB のバックアップおよび復元プロセス中に発生することの概要について説明します。

バックアップ

DynamoDB を使用してオンデマンドバックアップを作成すると、リクエストのタイムマーカーがカタログ化されます。このバックアップは、前回のテーブル全体のスナップショットへのリクエスト時間までにすべての変更を適用して、非同期的に作成されます。DynamoDB のバックアップリクエストは瞬時に処理され、数分以内に復元できる状態になります。

> Note
>
> オンデマンドバックアップを作成する度に、テーブルデータ全体がバックアップされます。オンデマンドバックアップの実行可能数に制限はありません。

DynamoDB では、テーブルのプロビジョンされたスループットを消費することなく、バックアップすることができます。

DynamoDB バックアップでは、項目間の因果整合性は保証されません。ただし、バックアップの更新間のスキューは、通常 1 秒未満です。

バックアップ進行中は、以下を行うことができません。

- バックアップオペレーションの一時停止またはキャンセル。
- バックアップのソーステーブルの削除。
- テーブルのバックアップ中におけるテーブルのバックアップの無効化。

スケジューリングスクリプトとクリーンアップジョブを作成したくない場合は、AWS Backup を使用して DynamoDB テーブルのスケジュールと保存ポリシーを含むバックアップ計画を作成できます。AWS Backup はバックアップを実行し、有効期限が切れると削除します。詳細については、「AWS Backup デベロッパーガイド」を参照してください。

AWS Backup に加えて、AWS Lambda 関数を使用して、定期的または以降のバックアップをスケジュールできます。詳細については、ブログ記事「Amazon DynamoDB のオンデマンドバックアップをスケジュールするサーバーレスソリューション」を参照してください。

コンソールを使用している場合、AWS Backup を使用して作成されたバックアップは、[Backup type (バックアップタイプ)] が `AWS` に設定された状態で [Backups (バックアップ)] タブに一覧表示されます。

> **Note**
>
> DynamoDB コンソールを使用して、[Backup type (バックアップタイプ)] が AWS でマークされたバックアップを削除することはできません。これらのバックアップを管理するには、AWS Backup コンソールを使用します。

バックアップを実行する方法については、「DynamoDB テーブルのバックアップ」を参照してください。

復元

テーブルは、テーブルのプロビジョニングされたスループットを消費することなく復元します。DynamoDB バックアップからテーブル全体を復元することも、送信先テーブルの設定を構成することもできます。復元を実行するときに、次のテーブル設定を変更できます。

- グローバルセカンダリインデックス (GSI)

- ローカルセカンダリインデックス (LSI)
- 請求モード
- プロビジョニングされた読み込みおよび書き込みキャパシティ
- 暗号化設定

> ⚠ Important
>
> テーブル全体を復元する場合、送信先テーブルはバックアップがリクエストされた時間に記録されたように、送信元テーブルと同じプロビジョニングされた読み込みキャパシティユニットおよびプロビジョニングされた書き込みキャパシティユニットを使用して設定されます。復元プロセスでは、ローカルセカンダリインデックスおよびグローバルセカンダリインデックスも復元されます。

また、バックアップが存在する別のリージョンに復元済みテーブルが作成されるように、AWS リージョン全体で DynamoDB テーブルデータを復元することもできます。AWS 商用リージョン、AWS 中国リージョン、および AWS GovCloud (米国) リージョン間でクロスリージョン復元を実行できます。送信元リージョンから転送したデータと、送信先リージョンの新しいテーブルの復元に対してのみ料金が発生します。

新しく復元されるテーブルで、一部またはすべてのセカンダリインデックスの作成を除外すると、復元はより高速でコスト効率が高くなります。

以下は、復元されたテーブルで手動で設定する必要があります。

- Auto Scaling ポリシー
- AWS Identity and Access Management (IAM) ポリシー
- Amazon CloudWatch メトリクスおよびアラーム
- タグ
- ストリーム設定
- 有効期限 (TTL) 設定
- 削除保護設定
- ポイントインタイムリカバリ (PITR) 設定

テーブルデータ全体は、バックアップから新しいテーブルにのみ復元することができます。復元されたテーブルに書き込むことができるのは、アクティブになってからです。

> **Note**
>
> 復元オペレーション中は既存のテーブルを上書きすることはできません。

サービスメトリクスでは、お客様のテーブル復元の 95% が 1 時間未満で完了していることを示しています。ただし、復元時間は、テーブルの構成 (テーブルのサイズ、基礎となるパーティションの数など) およびその他の関連する変数に直接関係しています。災害対策を計画する際のベストプラクティスは、平均復元完了時間を定期的に記録し、これらの時間が目標復旧時間全体にどのように影響するかを確認することです。

復元を実行する方法については、「バックアップからの DynamoDB テーブルの復元」を参照してください。

IAM ポリシーを使用してアクセスコントロールできます。詳細については、「DynamoDB バックアップおよび復元での IAM の使用」を参照してください。

バックアップおよび復元を行うコンソールと API のアクションはすべて、AWS CloudTrail にキャプチャおよび記録され、ログ記録、継続的モニタリング、監査に使用されます。

DynamoDB テーブルのバックアップ

このセクションでは、Amazon DynamoDB コンソールまたは AWS Command Line Interface を使用してテーブルをバックアップする方法について説明します。

トピック

- テーブルバックアップの作成 (コンソール)
- テーブルバックアップの作成 (AWS CLI)

テーブルバックアップの作成 (コンソール)

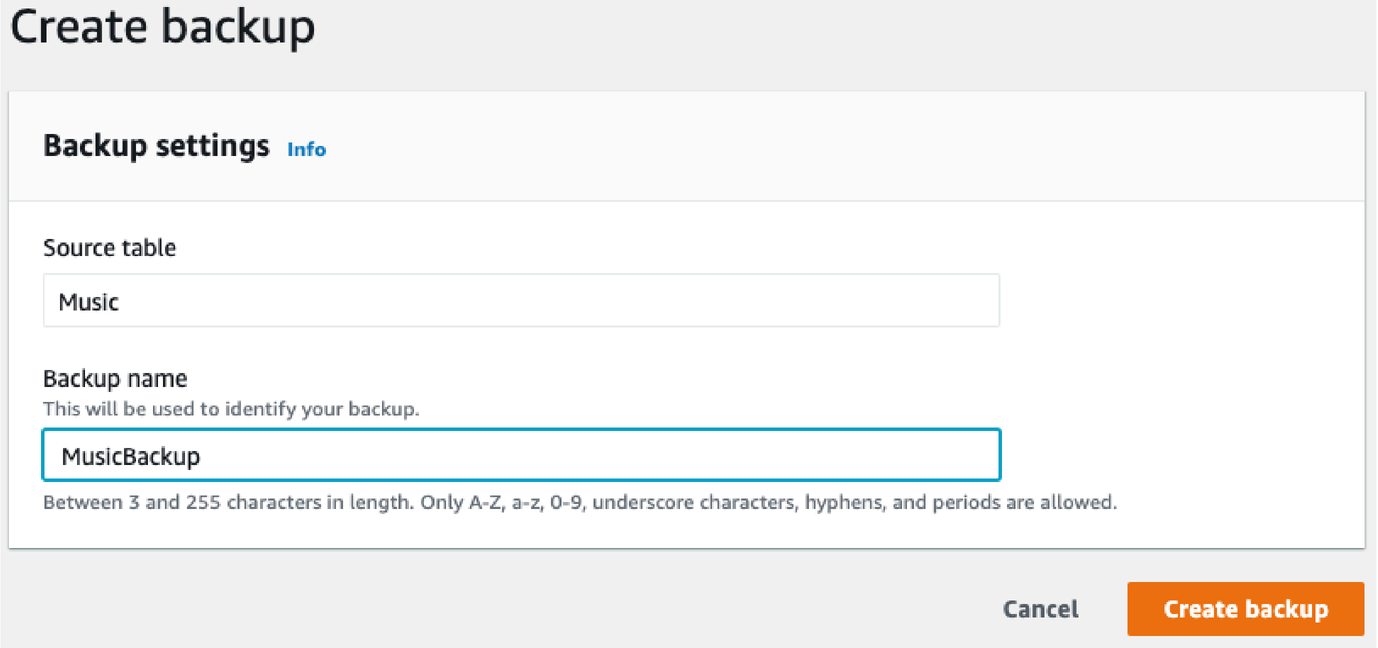
`MusicBackup` を使用して既存のテーブル `Music` のバックアップ AWS Management Console を作成するには、以下のステップに従います。

テーブルのバックアップを作成するには

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. バックアップを作成するには、次のいずれかを行います。

    - `Music` テーブルの [Backups] (バックアップ) タブで、[Create backup] (バックアップの作成) を選択します。
    - コンソールの左側のナビゲーションペインで、[Backups (バックアップ)] を選択します。続いて、[Create backup (バックアップを作成)] を選択します。

3. `Music` がテーブル名であることを確認し、バックアップ名として **`MusicBackup`** を入力します。続いて、[作成] を選択してバックアップを作成します。



Create backup

Backup settings Info

Source table

Music

Backup name

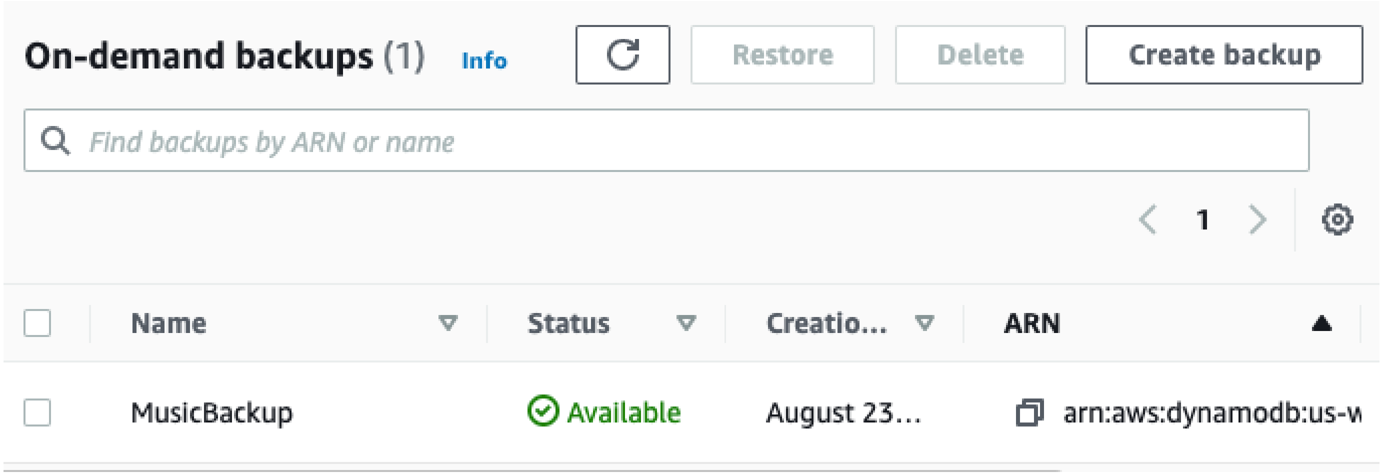
This will be used to identify your backup.

MusicBackup

Between 3 and 255 characters in length. Only A-Z, a-z, 0-9, underscore characters, hyphens, and periods are allowed.

> Note
>
> ナビゲーションペインの [Backups (バックアップ)] セクションでバックアップを作成した場合、テーブルは事前に選択されていません。バックアップ用のソーステーブルは、手動で選択する必要があります。

バックアップ作成中、バックアップステータスは [作成中] に設定されます。バックアップが完了すると、バックアップステータスは [利用可能] に変わります。



□	Name	Status	Creatio...	ARN ▲
□	MusicBackup	✓ Available	August 23...	arn:aws:dynamodb:us-w

テーブルバックアップの作成 (AWS CLI)

Music を使用して既存のテーブル AWS CLI のバックアップを作成するには、以下のステップに従います。

テーブルのバックアップを作成するには

- MusicBackup テーブルのバックアップ (Music) を作成します。

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

バックアップ作成中、バックアップステータスは CREATING に設定されます。

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

バックアップが完了すると、その BackupStatus が AVAILABLE に変更されます。これを確認するには、describe-backup コマンドを使用します。入力値 (backup-arn) は、前のステップの出力から、または list-backups コマンドを使用して取得できます。

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

バックアップを追跡するには、list-backups コマンドを使用できます。これにより、CREATING ステータスまたは AVAILABLE ステータスのバックアップがすべて表示されます。

```
aws dynamodb list-backups
```

バックアップのソーステーブルに関する情報を確認するには、list-backups コマンドおよび describe-backup コマンドが便利です。

バックアップからの DynamoDB テーブルの復元

このセクションでは、 Amazon DynamoDB コンソールまたは AWS Command Line Interface (AWS CLI) を使用してバックアップからテーブルを復元する方法について説明します。

> **Note**
>
> AWS CLI を使用する場合は、最初に設定する必要があります。詳細については、「DynamoDB にアクセスする」を参照してください。

トピック

- バックアップからのテーブルの復元 (コンソール)
- バックアップからのテーブルの復元 (AWS CLI)

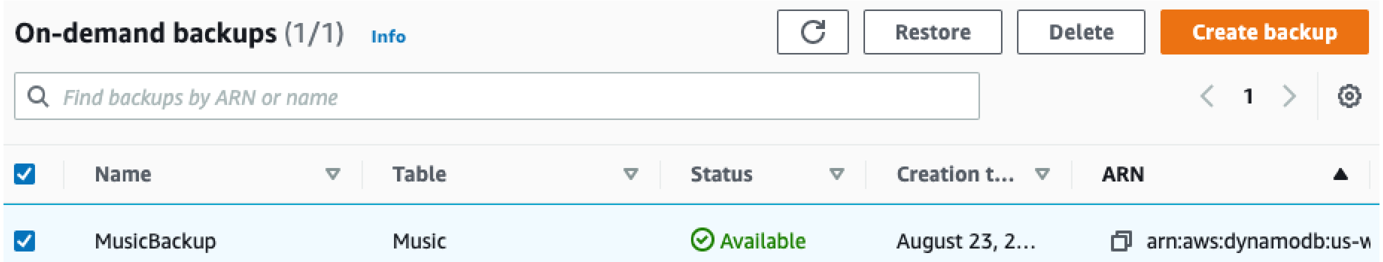
バックアップからのテーブルの復元 (コンソール)

以下の手順では、Music チュートリアルで作成された MusicBackup ファイルを使用して DynamoDB テーブルのバックアップ を復元する方法について説明します。

> **Note**
>
> この手順では、Music が既に存在していないことを前提として、これを MusicBackup ファイルを使用してリストアする方法を示します。

バックアップからテーブルを復元するには

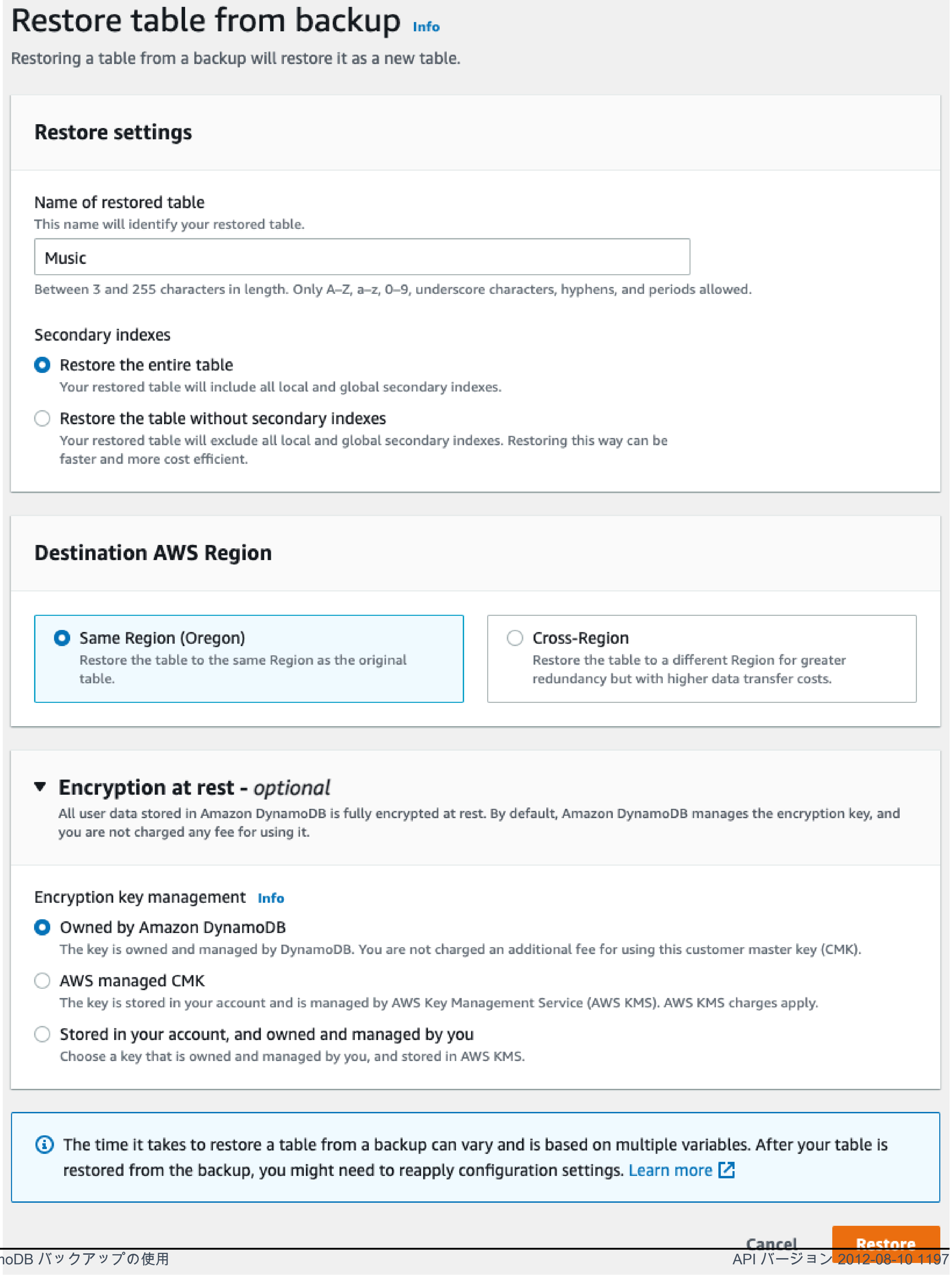
1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[Backups (バックアップ)] を選択します。
3. バックアップのリストで、[`MusicBackup`] を選択します。



☑	Name	Table	Status	Creation t...	ARN
☑	MusicBackup	Music	Available	August 23, 2...	arn:aws:dynamodb:us-w

4. [復元] を選択します。
5. 新しいテーブルの名前を **`Music`** と入力します。バックアップ名およびその他のバックアップの詳細を確認します。続いて、[Restore table (テーブルの復元)] を選択して復元プロセスを開始します。

> **Note**
>
> テーブルは、同じ AWS リージョンまたはバックアップが存在する別のリージョンに復元できます。新しく復元されるテーブルで、セカンダリインデックスの作成を除外することもできます。また、別の暗号化モードを指定することもできます。
> バックアップから復元されたテーブルは、常に DynamoDB 標準テーブルクラスを使用して作成されます。



Restore table from backup Info

Restoring a table from a backup will restore it as a new table.

Restore settings

Name of restored table

This name will identify your restored table.

Music

Between 3 and 255 characters in length. Only A–Z, a–z, 0–9, underscore characters, hyphens, and periods allowed.

Secondary indexes

● Restore the entire table
Your restored table will include all local and global secondary indexes.

○ Restore the table without secondary indexes
Your restored table will exclude all local and global secondary indexes. Restoring this way can be faster and more cost efficient.

Destination AWS Region

● Same Region (Oregon)
Restore the table to the same Region as the original table.

○ Cross-Region
Restore the table to a different Region for greater redundancy but with higher data transfer costs.

▼ Encryption at rest - *optional*

All user data stored in Amazon DynamoDB is fully encrypted at rest. By default, Amazon DynamoDB manages the encryption key, and you are not charged any fee for using it.

Encryption key management Info

● Owned by Amazon DynamoDB
The key is owned and managed by DynamoDB. You are not charged an additional fee for using this customer master key (CMK).

○ AWS managed CMK
The key is stored in your account and is managed by AWS Key Management Service (AWS KMS). AWS KMS charges apply.

○ Stored in your account, and owned and managed by you
Choose a key that is owned and managed by you, and stored in AWS KMS.

ⓘ The time it takes to restore a table from a backup can vary and is based on multiple variables. After your table is restored from the backup, you might need to reapply configuration settings. Learn more

復元中のテーブルのステータスは、[Creating (作成中)] と表示されます。復元プロセスが終了すると、Music テーブルのステータスは [アクティブ] に変わります。

バックアップからのテーブルの復元 (AWS CLI)

AWS CLI チュートリアルで作成されている Music を使用して、MusicBackup で DynamoDB テーブルのバックアップ テーブルを復元するには、以下の手順を実行します。

バックアップからテーブルを復元するには

1. list-backups コマンドを使用して復元するバックアップを確認します。この例では MusicBackup を使用します。

   ```
   aws dynamodb list-backups
   ```

   バックアップの詳細をさらに取得するには、describe-backup コマンドを使用します。前のステップから入力の backup-arn を取得できます。

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. バックアップからテーブルを復元します。この場合、MusicBackup は Music テーブルを同じ AWS リージョンに復元します。

   ```
   aws dynamodb restore-table-from-backup \
   --target-table-name Music \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

3. カスタムテーブル設定で、バックアップからテーブルを復元します。この場合、MusicBackup は Music テーブルを復元し、復元したテーブルの暗号化モードを指定します。

   > ⓘ Note
   >
   > sse-specification-override パラメータは、sse-specification-override コマンドで使用される CreateTable パラメータと同じ値を使用します。詳細については、「DynamoDB での暗号化テーブルの管理」を参照してください。

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

テーブルは、バックアップが存在する別の AWS リージョンに復元できます。

> **Note**
>
> - `sse-specification-override` パラメータは、クロスリージョン復元には必須ですが、送信元テーブルと同じリージョンに復元する場合はオプションです。
> - コマンドラインからクロスリージョン復元を実行する場合は、デフォルトの AWS リージョンを希望のリージョンに設定する必要があります。詳細については、「AWS Command Line Interface のユーザーガイド」の「コマンドラインオプション」を参照してください。

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS
```

復元済みテーブルの請求モードとプロビジョニングされたスループットをオーバーライドできます。

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d \
--billing-mode-override PAY_PER_REQUEST
```

復元済みテーブルで、一部またはすべてのセカンダリインデックスの作成を除外できます。

Note

復元済みテーブルで、一部またはすべてのセカンダリインデックスの作成を除外すると、復元はより高速でコスト効率が高くなります。

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581081403719-db9c1f91 \
--global-secondary-index-override '[]' \
--sse-specification-override Enabled=true,SSEType=KMS
```

Note

提供されたセカンダリインデックスは、既存のインデックスに一致します。復元時に新しいインデックスを作成することはできません。

さまざまなオーバーライドの組み合わせを使用できます。たとえば、次のように単一のグローバルセカンダリインデックスを使用すると同時に、プロビジョニングされたスループットを変更できます。

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:eu-west-1:123456789012:table/Music/
backup/01581082594992-303b6239 \
--billing-mode-override PROVISIONED \
--provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100 \
--global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEYS_
 \
--sse-specification-override Enabled=true,SSEType=KMS
```

復元を確認するには、describe-table コマンドで Music テーブルを指定します。

```
aws dynamodb describe-table --table-name Music
```

バックアップから復元中のテーブルのステータスは、[Creating (作成中)] と表示されます。復元プロセスが終了すると、`Music` テーブルのステータスは [アクティブ] に変わります。

> ⚠ Important
>
> 復元中は、IAM ロールのポリシーを変更または削除しないでください。行った場合、予期しない動作が発生する場合があります。たとえば、テーブルの復元中にテーブルの書き込み権限を削除したとします。この場合、基本となる `RestoreTableFromBackup` オペレーションを使用しても、復元されたデータをテーブルに書き込むことはできません。
> 復元オペレーションが完了したら、IAM ロールポリシーを変更または削除できます。
> 復元先のターゲットテーブルにアクセスするための送信元 IP の制限を含む IAM ポリシーでは、プリンシパルによって直接行われたリクエストにのみその制限が適用されるように、`aws:ViaAWSService` キーを `false` に設定する必要があります。そうしないと、復元はキャンセルされます。
> バックアップが AWS マネージドキー またはカスタマーマネージドキーで暗号化されている場合、復元中にキーを無効にしたり削除したりすると復元が失敗します。
> 復元操作が完了したら、復元されたテーブルの暗号化キーを変更し、古いキーを無効化または削除できます。

DynamoDB テーブルのバックアップの削除

このセクションでは、AWS Management Console または AWS Command Line Interface (AWS CLI) を使用して Amazon DynamoDB テーブルバックアップを削除する方法について説明します。

> ⓘ Note
>
> AWS CLI を使用する場合は、最初に設定する必要があります。詳細については、「AWS CLI の使用」を参照してください。

トピック

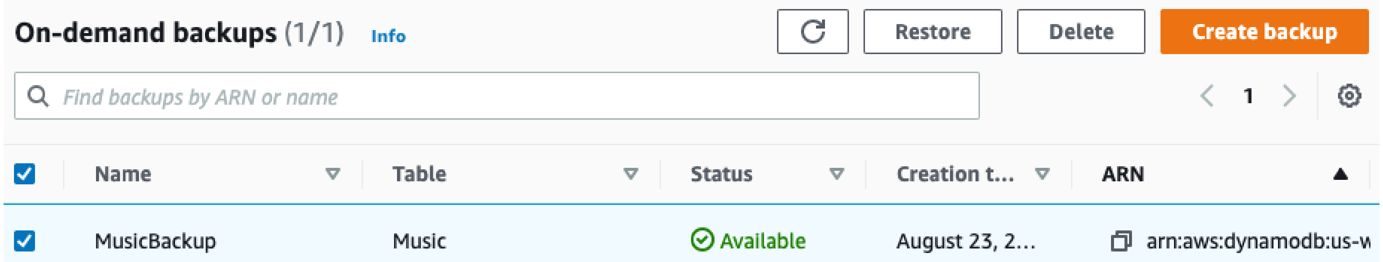
- テーブルバックアップの削除 (コンソール)
- テーブルバックアップの削除 (AWS CLI)

テーブルバックアップの削除 (コンソール)

次の手順では、コンソールを使用し、`MusicBackup` チュートリアルで作成した DynamoDB テーブルのバックアップ を削除する方法を示します。

バックアップを削除するには

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[Backups (バックアップ)] を選択します。
3. バックアップのリストで、[`MusicBackup`] を選択します。



☑	Name	Table	Status	Creation t...	ARN
☑	MusicBackup	Music	✓ Available	August 23, 2...	arn:aws:dynamodb:us-w

4. [削除] を選択します。「**`delete`**」と入力してから、[削除] をクリックして、バックアップの削除を確認します。

テーブルバックアップの削除 (AWS CLI)

次の例では、`Music` を使用して、既存のテーブルである AWS CLI テーブルのバックアップを削除します。

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

DynamoDB バックアップおよび復元での IAM の使用

AWS Identity and Access Management (IAM) を使用すれば、一部リソースの Amazon DynamoDB バックアップアクションと復元アクションを制限することができます。`CreateBackup` および `RestoreTableFromBackup` API はテーブルごとにオペレーションを行います。

DynamoDB での IAM ポリシーの詳細な使用については、「DynamoDB のアイデンティティベースのポリシー」を参照してください。

次に、DynamoDB で特定のバックアップ機能と復元機能を設定するために使用できる IAM ポリシーの例を示します。

例 1: CreateBackup および RestoreTableFromBackup アクションを許可する

以下の IAM ポリシーは、すべてのテーブルで `CreateBackup` および `RestoreTableFromBackup` DynamoDB アクションを適用する許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

> ⚠ Important
>
> DynamoDB RestoreTableFromBackup アクセス許可はソースバックアップに必要であり、復元機能にはターゲットテーブルに対する DynamoDB の読み取りおよび書き込みアクセス許可が必要です。
> RestoreTableToPointInTime アクセス許可はソーステーブルに必要であり、復元機能にはターゲットテーブルに対する DynamoDB の読み取りおよび書き込みアクセス許可が必要です。

例 2: CreateBackup アクションを許可し、RestoreTableFromBackup アクションを拒否する

以下の IAM ポリシーは、`CreateBackup` アクションの許可を付与し、`RestoreTableFromBackup` アクションを拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

例 3: ListBackups アクションを許可し、CreateBackup および RestoreTableFromBackup アクションを拒否する

以下の IAM ポリシーは、`ListBackups` アクションの許可を付与し、`CreateBackup` アクションおよび `RestoreTableFromBackup` アクションを拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

例 4: ListBackups を許可し、DeleteBackup を拒否する

以下の IAM ポリシーは、ListBackups アクションの許可を付与し、DeleteBackup アクションを拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

例 5: すべてのリソースに対する RestoreTableFromBackup および DescribeBackup を許可し、特定のバックアップに対する DeleteBackup を拒否する

以下の IAM ポリシーは、RestoreTableFromBackup および DescribeBackup アクションの許可を付与し、特定のバックアップリソースに対する DeleteBackup アクションを拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeBackup",
                "dynamodb:RestoreTableFromBackup",
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        },
        {
             "Effect": "Allow",
             "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DeleteBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        }
    ]
}
```

> ⚠ Important
>
> DynamoDB RestoreTableFromBackup アクセス許可はソースバックアップに必要であり、復元機能にはターゲットテーブルに対する DynamoDB の読み取りおよび書き込みアクセス許可が必要です。
> RestoreTableToPointInTime アクセス許可はソーステーブルに必要であり、復元機能にはターゲットテーブルに対する DynamoDB の読み取りおよび書き込みアクセス許可が必要です。

例 6: 特定のテーブルに対する CreateBackup を許可する

以下の IAM ポリシーは、`Movies` テーブルに対する `CreateBackup` アクションの許可のみを付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

例 7: ListBackups を許可する

以下の IAM ポリシーは、`ListBackups` アクションに対する許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
}
```

> ⚠ Important
>
> 特定のテーブルに対する `ListBackups` アクションの許可を付与することはできません。

例 8: AWS Backup 機能へのアクセス許可

高度な機能を備えたバックアップを完了するための `StartAwsBackupJob` アクションと、そのバックアップを正常に復元するための `dynamodb:RestoreTableFromAwsBackup` アクションに対する API アクセス許可が必要になります。

次の IAM ポリシーでは、高度な機能と復元を使用してバックアップをトリガーするためのアクセス許可を AWS Backup に付与します。また、テーブルが暗号化されている場合、ポリシーは AWS KMS キーにアクセスする必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:StartAwsBackupJob",
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:account-id:table/Books"
        },
        {
            "Sid": "AllowRestoreFromAwsBackup",
            "Effect": "Allow",
            "Action": ["dynamodb:RestoreTableFromAwsBackup"],
            "Resource": "*"
        },

    ]
}
```

例 9: 特定のソーステーブルの RestoreTableToPointInTime を拒否する

以下の IAM ポリシーは、特定のソーステーブルに対する `RestoreTableToPointInTime` アクションの許可を拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

例 10: 特定のソーステーブルのすべてのバックアップに対して RestoreTableFromBackup を拒否する

以下の IAM ポリシーは、特定のソーステーブルのすべてのバックアップに対する `RestoreTableToPointInTime` アクションの許可を拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/*"
        }
    ]
}
```

DynamoDB のポイントインタイムリカバリ

Amazon DynamoDB テーブルのオンデマンドバックアップを作成するか、ポイントインタイムリカバリを使用して連続バックアップを有効にすることができます。オンデマンドバックアップの詳細については、「DynamoDB のオンデマンドバックアップおよび復元の使用」を参照してください。

ポイントインタイムリカバリを使用することで、DynamoDB テーブルを、偶発的な書き込みや削除のオペレーションから保護できます。ポイントインタイムリカバリを有効化すれば、オンデマンドバックアップの作成、維持、スケジュールを心配する必要はありません。たとえば、テストスクリプトで、誤って本稼働環境の DynamoDB テーブルに書き込みを行ったとします。ポイントインタイムリカバリを使用すれば、過去 35 日間の任意の時点にテーブルを復元することができます。ポイントインタイムリカバリを有効にしたら、現在時刻の 5 分前から 35 日前までの任意の時点に復元できます。DynamoDB では、テーブルの増分バックアップが維持されます。

また、ポイントインタイムリカバリオペレーションが、パフォーマンスや API レイテンシーに影響を及ぼすことはありません。詳細については、「ポイントインタイムリカバリ: 仕組み」を参照してください。

DynamoDB テーブルをポイントインタイムに復元するには、AWS Management Console、 AWS Command Line Interface (AWS CLI)、または DynamoDB API を使用します。ポイントインタイムリカバリプロセスは、新しいテーブルに復元されます。詳細については、「DynamoDB テーブルを特定の時点に復元」を参照してください。

次の動画では、バックアップと復元の概念を紹介し、ポイントインタイムリカバリについて詳しく説明します。

バックアップと復元

使用可能な AWS リージョンおよび料金に関する詳細については、Amazon DynamoDB 料金表を参照してください。

トピック

ポイントインタイムリカバリ: 仕組み

Amazon DynamoDB ポイントインタイムリカバリ (PITR) は、DynamoDB テーブルデータを自動バックアップする機能です。このセクションでは、DynamoDB でのプロセスの動作の概要を説明します。

ポイントインタイムリカバリの有効化

ポイントインタイムリカバリは、AWS Management Console、AWS Command Line Interface (AWS CLI)、または DynamoDB API を使用して有効にできます。有効化されると、ポイントインタイムリカバリは、明示的に無効化するまで、連続バックアップを行います。詳細については、「DynamoDB テーブルを特定の時点に復元」を参照してください。

ポイントインタイムリカバリを有効にしたら、`EarliestRestorableDateTime` および `LatestRestorableDateTime` 内の任意の時点に復元できます。`LatestRestorableDateTime` は、通常、現在時刻の 5 分前です。

> Note
>
> ポイントインタイムリカバリプロセスは、必ず新しいテーブルに復元されます。

ポイントインタイムリカバリを使用したテーブルの復元

`EarliestRestorableDateTime` を使用すると、過去 35 日間の任意の時点にテーブルを復元できます。保持期間は、35 日間 (5 週間) に固定されており、変更することはできません。任意の数のユーザーが、特定のアカウントで最大 50 の同時復元 (任意のタイプの復元) を実行できます。

> **Important**
>
> ポイントインタイムリカバリを無効にし、後にテーブル上で再度有効にした場合は、そのテーブルで復元可能な開始時間をリセットします。そのため、`LatestRestorableDateTime` を使用して、すぐにそのテーブルを復元できます。

ポイントインタイムリカバリを使用して復元すると、DynamoDB は、選択された日時 (`day:hour:minute:second`) に基づく状態でテーブルデータを新しいテーブルに復元します。

テーブルは、テーブルのプロビジョニングされたスループットを消費することなく復元します。ポイントインタイムリカバリを使用してテーブル全体を復元するか、送信先テーブルの設定を構成することができます。復元済みテーブルで次のテーブル設定を変更できます。

- グローバルセカンダリインデックス (GSI)
- ローカルセカンダリインデックス (LSI)
- 請求モード
- プロビジョニングされた読み込みおよび書き込みキャパシティ
- 暗号化設定

> **Important**
>
> テーブル全体の復元を実行すると、送信先テーブルには、バックアップが要求されたときに送信元テーブルが持っていたのと同じプロビジョニングされた読み込みキャパシティユニットと書き込みキャパシティユニットが設定されます。たとえば、テーブルのプロビジョニングされたスループットが 50 読み込みキャパシティーユニットおよび 50 書き込みキャパシティーユニットに最近下げられたとします。その後、このテーブルの状態を 3 週間前に復元し、その時点で、プロビジョニングされたスループットは 100 読み込みキャパシティーユニットと 100 書き込みキャパシティーユニットに設定されました。この場合、DynamoDB

はテーブルデータをその時点からのプロビジョンされたスループット (100 読み込み容量単位および 100 書き込み容量単位) を利用してその時点のものに復元します。

また、送信元テーブルが存在する別のリージョンに復元済みテーブルが作成されるように、AWS リージョン全体で DynamoDB テーブルデータを復元することもできます。AWS 商用リージョン、AWS 中国リージョン、および AWS GovCloud (米国) リージョン間でクロスリージョン復元を実行できます。送信元リージョンから転送したデータと、送信先リージョンの新しいテーブルの復元に対してのみ料金が発生します。

Note

送信元または送信先リージョンがアジアパシフィック (香港) または中東 (バーレーン) の場合、クロスリージョン復元はサポートされません。

復元済みテーブルで、一部またはすべてのインデックスの作成を除外すると、復元はより高速でコスト効率が高くなります。

以下は、復元されたテーブルで手動で設定する必要があります。

- Auto Scaling ポリシー
- AWS Identity and Access Management (IAM) ポリシー
- Amazon CloudWatch メトリクスおよびアラーム
- タグ
- ストリーム設定
- 有効期限 (TTL) 設定
- ポイントインタイムリカバリ設定
- 削除保護設定

テーブルの復元にかかる時間は複数の要因によって異なります。ポイントインタイム復元の時間は必ずしもテーブルのサイズに直接関連していません。詳細については、「復元」を参照してください。

ポイントインタイムリカバリが有効になっているテーブルの削除

ポイントインタイムリカバリが有効になっているテーブルを削除すると、DynamoDB はシステムバックアップと呼ばれるバックアップスナップショットを自動的に作成し、それを 35 日間保持しま

す (追加コストは発生しません)。システムバックアップを使用して、削除されたテーブルを削除の直前の状態に復元できます。システムバックアップはすべて、*table-name* $DeletedTableBackup という標準的な命名規則に従います。

> Note
>
> ポイントインタイムリカバリが有効になっているテーブルが削除されたら、システムの復元を使用してそのテーブルを特定の時点、つまり削除直前の時点に復元できます。削除したテーブルを過去 35 日間の他の時点に復元することはできません。

ポイントインタイムリカバリを使用して開始する前に

Amazon DynamoDB テーブルでポイントインタイムリカバリ (PITR) を有効にする前に、以下の点を考慮します。

- ポイントインタイムリカバリを無効にし、後にテーブル上で再度有効にした場合は、そのテーブルで復元可能な開始時間をリセットします。そのため、LatestRestorableDateTime を使用して、すぐにそのテーブルを復元できます。
- ポイントインタイムリカバリは、グローバルテーブルのローカルレプリカごとに有効にすることができます。テーブルを復元すると、バックアップは、グローバルテーブルに含まれていない独立したテーブルに復元されます。グローバルテーブルバージョン 2019.11.21 (現行) を使用している場合、復元されたテーブルから新しいグローバルテーブルを作成できます。詳細については、「グローバルテーブル: 仕組み」を参照してください。
- また、送信元テーブルが存在する別のリージョンに復元済みテーブルが作成されるように、AWS リージョン全体で DynamoDB テーブルデータを復元することもできます。AWS 商用リージョン、AWS 中国リージョン、および AWS GovCloud (米国) リージョン間でクロスリージョン復元を実行できます。送信元リージョンから転送したデータと、送信先リージョンの新しいテーブルの復元に対してのみ料金が発生します。
- AWS CloudTrailポイントインタイムリカバリのコンソールと API のアクションはすべて、 に記録され、ログ記録、継続的モニタリング、監査に使用されます。詳細については、「AWS CloudTrail を使用して DynamoDB オペレーションをログに記録する」を参照してください。

DynamoDB テーブルを特定の時点に復元

Amazon DynamoDB ポイントインタイムリカバリ (PITR) は、DynamoDB テーブルデータを連続バックアップする機能です。テーブルをポイントインタイムに復元するには、DynamoDB コンソー

ル、または AWS Command Line Interface (AWS CLI) を使用します。ポイントインタイムリカバリプロセスは、新しいテーブルに復元されます。

AWS CLI を使用する場合は、最初に設定する必要があります。詳細については、「DynamoDB にアクセスする」を参照してください。

トピック

- DynamoDB テーブルを特定の時点に復元 (コンソール)
- テーブルを特定の時点に復元する (AWS CLI)

DynamoDB テーブルを特定の時点に復元 (コンソール)

DynamoDB コンソールを使用して、`Music` という既存のテーブルを特定の時点に復元する方法を次の例に示します。

> **Note**
>
> この手順は、ポイントインタイムリカバリを有効にしていることを前提としています。`Music` テーブルで有効にするには、[バックアップ] タブの [ポイントインタイムリカバリ (PITR)] セクションで、[編集] を選択してから、[ポイントインタイムリカバリを有効にする] の横にあるチェックボックスをオンにします。

テーブルをポイントインタイムに復元するには

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
3. テーブルのリストで、[`Music`] テーブルを選択します。
4. `Music` テーブルの [バックアップ] タブの [ポイントインタイムリカバリ (PITR)] セクションで、[復元] を選択します。
5. 新しいテーブルの名前に **`MusicMinutesAgo`** と入力します。

> Note
>
> テーブルは、同じ AWS リージョンまたは送信元テーブルが存在する別のリージョンに復元できます。復元済みテーブルで、セカンダリインデックスの作成を除外することもできます。また、別の暗号化モードを指定することもできます。

6. 復元可能な時刻を確認するには、[復元日時] を [最も早い] に設定します。次に、[復元] を選択して復元プロセスを開始します。

   復元中のテーブルのステータスは、[Restoring (復元中)] と表示されます。復元プロセスが終了すると、`MusicMinutesAgo` テーブルのステータスは [アクティブ] に変わります。

テーブルを特定の時点に復元する (AWS CLI)

AWS CLI を使用して、`Music` という既存のテーブルをポイントインタイムに復元する方法を次の手順に示します。

> Note
>
> この手順は、ポイントインタイムリカバリを有効にしていることを前提としています。この機能を `Music` テーブルに対して有効にするには、次のコマンドを実行します。
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

テーブルをポイントインタイムに復元するには

1. `Music` でポイントインタイムリカバリが有効になっていることを確認するには、`describe-continuous-backups` コマンドを使用します。

   ```
   aws dynamodb describe-continuous-backups \
       --table-name Music
   ```

   連続バックアップ (テーブル作成時は自動的に有効になる) とポイントインタイムリカバリは有効化されています。

```
{
    "ContinuousBackupsDescription": {
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": 1519257118.0,
            "LatestRestorableDateTime": 1520018653.01
        },
        "ContinuousBackupsStatus": "ENABLED"
    }
}
```

2. テーブルをポイントインタイムに復元します。この場合、Music テーブルは同じ AWS リージョンの LatestRestorableDateTime (最大 5 分前) に復元されます。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time
```

> **Note**
>
> 特定の時点に復元することもできます。そのためには、--restore-date-time 引数を使用してコマンドを実行し、タイムスタンプを指定します。過去 35 日間の任意の時点を指定できます。たとえば、次のコマンドでは EarliestRestorableDateTime にテーブルを復元します。
>
> ```
> aws dynamodb restore-table-to-point-in-time \
>     --source-table-name Music \
>     --target-table-name MusicEarliestRestorableDateTime \
>     --no-use-latest-restorable-time \
>     --restore-date-time 1519257118.0
> ```
>
> 特定の時点に復元する場合の --no-use-latest-restorable-time 引数の指定はオプションです。

3. カスタムテーブル設定で、テーブルを特定時点に復元します。この場合、Music テーブルは、LatestRestorableDateTime に復元されます (5 分前まで)。

   復元済みテーブルに対して、次のように別の暗号化モードを指定できます。

Note

sse-specification-override パラメータは、sse-specification-override コマンドで使用される CreateTable パラメータと同じ値を使用します。詳細については、「DynamoDB での暗号化テーブルの管理」を参照してください。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

テーブルは、送信元テーブルが存在する別の AWS リージョンから復元できます。

Note

- sse-specification-override パラメータは、クロスリージョン復元には必須ですが、送信元テーブルと同じリージョンに復元する場合はオプションです。
- クロスリージョン復元には、source-table-arn パラメータを指定する必要があります。
- コマンドラインからクロスリージョン復元を実行する場合は、デフォルトの AWS リージョンを希望のリージョンに設定する必要があります。詳細については、「AWS Command Line Interface のユーザーガイド」の「コマンドラインオプション」を参照してください。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

復元済みテーブルの請求モードとプロビジョニングされたスループットをオーバーライドできます。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --billing-mode-override PAY_PER_REQUEST
```

復元済みテーブルで、一部またはすべてのセカンダリインデックスの作成を除外できます。

> **Note**
>
> 新しく復元されるテーブルで、一部またはすべてのセカンダリインデックスの作成を除外すると、復元はより高速でコスト効率が高くなります。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

さまざまなオーバーライドの組み合わせを使用できます。たとえば、次のように単一のグローバルセカンダリインデックスを使用すると同時に、プロビジョニングされたスループットを変更できます。

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --billing-mode-override PROVISIONED \
    --provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100
 \
    --global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEY
 \
    --sse-specification-override Enabled=true,SSEType=KMS \
    --use-latest-restorable-time
```

復元を確認するには、describe-table コマンドで MusicEarliestRestorableDateTime テーブルを指定します。

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

復元中のテーブルは、ステータスが [Creating (作成中)]、復元中に [true] として表示されています。復元プロセスが終了すると、MusicEarliestRestorableDateTime テーブルのステータスは [アクティブ] に変わります。

> ⚠ Important
>
> 復元中は、復元を目的とした、IAM エンティティ (例: ユーザー、グループ、ロール) を付与する AWS Identity and Access Management (IAM) ポリシーの変更や削除を行わないでください。行った場合、予期しない動作が発生する場合があります。たとえば、テーブルの復元中にテーブルの書き込み権限を削除したとします。この場合、基本となる RestoreTableToPointInTime オペレーションを使用しても、復元されたデータをテーブルに書き込むことはできません。復元先のターゲットテーブルにアクセスするための送信元 IP 制限を含む IAM ポリシーでも、同様の問題が発生する可能性があります。
> 復元オペレーション完了後は、アクセス権限の変更または削除のみ行うことができます。

DynamoDB Accelerator (DAX) とインメモリアクセラレーション

Amazon DynamoDB はスケールとパフォーマンスのために設計されています。ほとんどの場合、DynamoDB の応答時間は 1 桁台のミリ秒単位で測定できます。ただし、マイクロ秒の応答時間を必要とする一部のユースケースがあります。これらのユースケースでは DynamoDB Accelerator (DAX) が整合性データへのアクセス時に素早い対応を実現します。

DAX は、DynamoDB と互換性のあるキャッシュサービスで、条件の厳しいアプリケーションで高速なインメモリパフォーマンスを可能にします。DAX は、次の 3 つのコアシナリオに対応します。

1. インメモリキャッシュとしての DAX は、1 桁台のミリ秒単位からマイクロ秒単位まで、結果整合性のある読み込みワークロードの応答時間を短縮します。
2. DAX は、DynamoDB と API の互換性のあるマネージドサービスを提供することにより、オペレーションとアプリケーションの複雑さを軽減します。したがって、既存のアプリケーションで使用するために必要なのは最小限の機能変更だけです。
3. 読み込みの多いワークロードや急激に増大するワークロードにおいて、DAX はスループットを強化することや、読み込み容量ユニットを必要以上にプロビジョニングしないようにすることで運用コストの節約を可能にします。個々のキーで繰り返し読み込みが必要なアプリケーションにおいては特にメリットがあります。

DAX はサーバー側の暗号化をサポートします。保存時の暗号化では、ディスク上で DAX によって保持されるデータが暗号化されます。DAX は、プライマリノードからリードレプリカへの変更の伝播の一部として、データをディスクに書き込みます。詳細については、「」を参照してください保管時の DAX 暗号化

DAX は転送中の暗号化もサポートしており、アプリケーションとクラスター間のすべてのリクエストとレスポンスがトランスポートレベルセキュリティ (TLS) によって暗号化され、クラスターへの接続はクラスター x509 証明書の検証によって認証されます。詳細については、「」を参照してください転送時の DAX 暗号化

トピック

DAX のユースケース

DAX は、マイクロ秒のレイテンシーで DynamoDB テーブルからの結果整合性データへのアクセスを提供します。マルチ AZ DAX クラスターは、1 秒間に数百万件のリクエストを処理できます。

DAX は、次のタイプのアプリケーションに最適です。

- 可能な限り迅速な読み込み応答時間を必要とするアプリケーション。たとえば、リアルタイム入札、ソーシャルゲーム、トレーディングアプリケーションなどです。DAX はこれらのユースケースに対して、高速なインメモリ読み取りパフォーマンスを提供します。
- 少数の項目を他のものよりも頻繁に読み込むアプリケーション。たとえば、人気商品の 1 日限りのセールを行う e コマースシステムを考えてみます。セール中は、その商品 (および DynamoDB 内のそのデータ) に対する需要が他の商品全般に比べて急増します。「ホット」キーおよび不均一なトラフィックディストリビューションの影響を緩和するために、1 日限りのセールが終了するまで、読み込みアクティビティを DAX キャッシュにオフロードできます。
- 読み込み負荷が高いが、コストも重要なアプリケーション。DynamoDB を使用する場合、アプリケーションが要求する読み込み数を 1 秒ごとにプロビジョニングします。読み込みアクティビティが上昇すれば、テーブルにプロビジョニングされた読み込みスループットも増加し (追加コ

ストがかかり) ます。または、アプリケーションからのアクティビティを DAX クラスターにオフロードして、そうしない場合に購入する必要がある読み込みキャパシティユニットの数を削減できます。

- 大規模データセットに対して繰り返し読み込みが必要なアプリケーション。このようなアプリケーションは、データベースリソースを他のアプリケーションから引き離してしまう可能性があります。たとえば、リージョンの天気データの長期分析では、DynamoDB テーブルのすべての読み込みキャパシティが一時的に消費される可能性があります。この状況は、同じデータにアクセスする必要がある他のアプリケーションに悪影響を及ぼします。DAX を使用することで、代わりにキャッシュデータに対して気象分析を実行できます。

DAX は、次のタイプのアプリケーションには最適ではありません。

- 強整合性のある読み込みを必要とする (または結果整合性のある読み込みを許容できない) アプリケーション。
- 読み込みでマイクロ秒の応答時間を要求しない、または基礎となるテーブルから繰り返される読み込みアクティビティをオフロードする必要がないアプリケーション。
- 書き込み負荷が高い、またはそれほど多くの読み込みアクティビティを実行しないアプリケーション。
- DynamoDB ですでに別のキャッシングソリューションを使用しており、そのキャッシングソリューションと連携するために独自のクライアント側ロジックを使用しているアプリケーション。

DAX の使用に関する注意事項

- DAX を使用できる AWS リージョンのリストについては、「Amazon DynamoDB の料金表」を参照してください。
- DAX は、Go、Java、Node.js、Python、および .NET で記述されたアプリケーションをサポートします (これらのプログラミング言語用に AWS が提供するクライアントを使用します)。
- DAX は EC2-VPC プラットフォームでのみ使用できます。
- DAX クラスターサービスロールポリシーは、DynamoDB テーブルに関するメタデータを維持するために `dynamodb:DescribeTable` アクションを許可する必要があります。
- DAX クラスターは、格納する項目の属性名に関するメタデータを保持します。そのメタデータは (項目の有効期限が切れた後、またはキャッシュから削除された後でも) 無期限に保持されます。大量の属性名を使用するアプリケーションは、時間の経過につれて DAX クラスターのメモリ消耗を引き起こすことがあります。この制限は最上位の属性名のみに適用され、ネスト属性名には適用さ

れません。問題が発生しやすい最上位の属性名の例としては、タイムスタンプ、UUID や ID があります。

この制限は属性名のみに適用され、その値には適用されません。以下のような項目に問題はありません。

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "CreationDate": "2017-10-24T01:02:03+00:00"
}
```

ただし、以下のような項目は、十分な数があり、それぞれ別のタイムスタンプがある場合は、問題になります。

```
{
    "Id": 123,
    "Title": "Bicycle 123",
    "2017-10-24T01:02:03+00:00": "created"
}
```

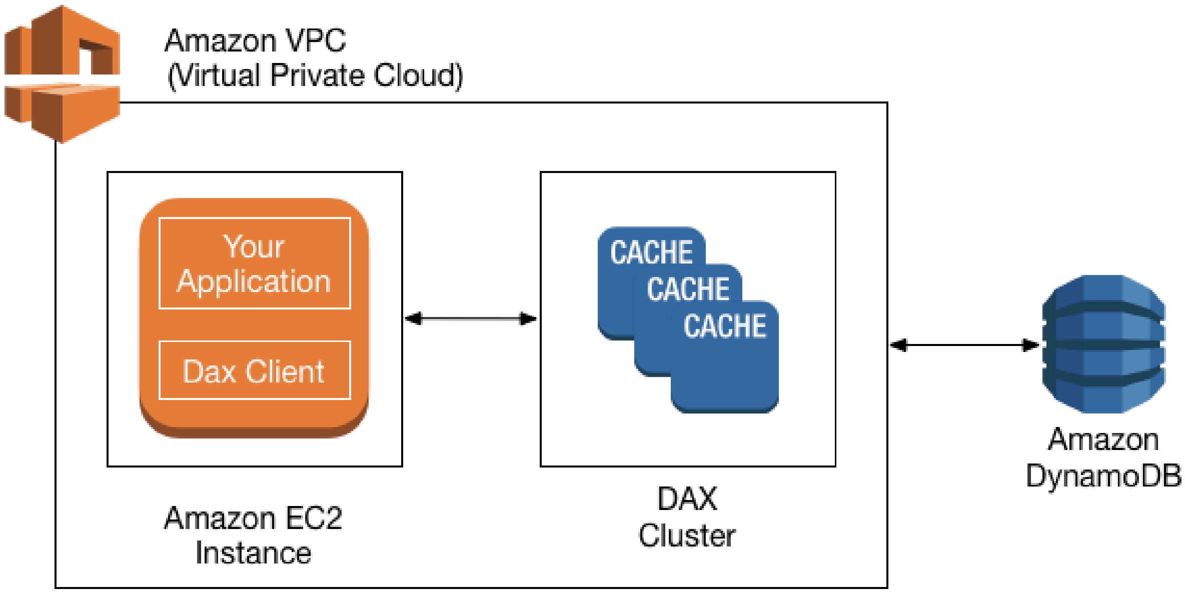
DAX: 仕組み

Amazon DynamoDB Accelerator (DAX) は、Amazon Virtual Private Cloud (Amazon VPC) 環境内で実行するように設計されています。Amazon VPC サービスは、従来のデータセンターに非常によく似た仮想ネットワークを定義します。VPC を使用すると、IP アドレス範囲、サブネット、ルーティングテーブル、ネットワークゲートウェイ、セキュリティ設定を適切に制御できます。仮想ネットワーク内で DAX クラスターを起動し、Amazon VPC のセキュリティグループを使用して、クラスターへのアクセスを制御できます。

ⓘ Note

2013 年 12 月 4 日以降に AWS アカウントを作成した場合は、各 AWS リージョンにデフォルトで VPC が用意されています。VPC はすぐに使用できます。追加のステップは必要ありません。
詳細については、「Amazon VPC ユーザーガイド」の「デフォルト VPC とデフォルトサブネット」を参照してください。

次の図は、DAX の高レベルの概要を示しています。



DAX クラスターを作成するには、AWS Management Console を使用します。特に指定しない限り、DAX クラスターはデフォルト VPC 内で実行されます。アプリケーションを実行するには、Amazon VPC 内に Amazon EC2 インスタンスを起動します。次に、アプリケーションを (DAX クライアントとともに) EC2 インスタンスにデプロイします。

実行時に、DAX クライアントはアプリケーションのすべての DynamoDB API リクエストを DAX クラスターに送信します。DAX がこれらの API リクエストのいずれかを直接処理できる場合は、処理します。それ以外の場合は、リクエストは DynamoDB に渡されます。

最終的に、DAX クラスターはアプリケーションに結果を返します。

トピック

- DAX でのリクエスト処理方法
- 項目キャッシュ
- クエリキャッシュ

DAX でのリクエスト処理方法

DAX クラスターは 1 つ以上のノードで構成されています。各ノードはそれぞれ、自身の DAX キャッシングソフトウエアのインスタンスを実行します。ノードのうち 1 つがクラスターのプライマリノードとして機能します。他のノード (存在する場合) はリードレプリカとして動作します。詳細については、「ノード」を参照してください。

アプリケーションは、DAX クラスターのエンドポイントを指定することで DAX にアクセスできます。DAX クライアントソフトウェアはクラスターエンドポイントと連携して、インテリジェントなロードバランシングとルーティングを実行します。

読み込み操作

DAX は次の API コールに対応できます。

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

リクエストが結果的に整合性のある読み込み (デフォルトの動作) を指定する場合は、その項目を DAX から読み込もうとします。

- DAX に項目がある場合 (キャッシュヒット)、DAX は DynamoDB にアクセスせずにアプリケーションに項目を返します。
- DAX に項目がない場合 (キャッシュミス)、DAX はリクエストを DynamoDB に渡します。DynamoDB からの応答を受信すると、DAX は結果をアプリケーションに返します。ただし、プライマリノードのキャッシュにも結果を書き込みます。

> **Note**
>
> クラスターにリードレプリカがある場合、DAX は自動的にレプリカをプライマリノードと同期した状態に保ちます。詳細については、「」を参照してくださいクラスター

リクエストが強力な整合性のある読み込みを指定する場合、DAX はリクエストを DynamoDB に渡します。DynamoDB からの結果は DAX にキャッシュされません。代わりに、それらは単にアプリケーションに返されます。

書き込み操作

次の DAX API オペレーションは「書き込みスルー」と見なされます。

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

これらのオペレーションでは、データはまず DynamoDB テーブルに書き込まれ、その後 DAX クラスターに書き込まれます。オペレーションは、データがテーブルと DAX の両方に正常に書き込まれた場合にのみ成功します。

その他のオペレーション

DAX では、テーブルを管理する DynamoDB のオペレーション (`CreateTable` や `UpdateTable` など) を認識しません。アプリケーションがこのようなオペレーションを行う必要がある場合は、DAX を使用するのではなく、直接 DynamoDB にアクセスする必要があります。

DAX および DynamoDB の整合性の詳細については、「DAX および DynamoDB の整合性モデル」を参照してください。

DAX でのトランザクションの動作の詳細については、「DynamoDB アクセラレーター (DAX) でのトランザクション API の使用」を参照してください。

リクエストレート制限

DAX に送信された要求の数がノードの容量を超える場合、DAX は ThrottlingException を返して追加の要求を受け入れるレートを制限します。DAX は CPU 使用率を継続的に評価し、正常なクラスター状態を維持しながら処理できるリクエスト量を判断します。

DAX が Amazon CloudWatch に公開する ThrottledRequestCount メトリクスをモニタリングできます。これらの例外が定期的に発生する場合は、クラスターをスケールアップすることを検討してください。

項目キャッシュ

DAX は項目キャッシュを維持して、`GetItem` および `BatchGetItem` オペレーションからの結果を保存します。キャッシュ内の項目は DynamoDB からの結果整合性データを表し、プライマリキー値別に保存されています。

アプリケーションが `GetItem` または `BatchGetItem` リクエストを送信すると、DAX は指定されたキー値を使用して項目キャッシュから直接項目の読み込みを試みます。項目が見つかった場合 (キャッシュヒット)、DAX はすぐにアプリケーションに項目を返します。項目が見つからない (キャッシュミス) 場合、DAX は DynamoDB にリクエストを送信します。DynamoDB は、結果整合性のある読み取りを使用してリクエストを処理し、項目を DAX に返します。DAX は、それらを項目キャッシュに格納し、アプリケーションに返します。

項目キャッシュには有効期限 (TTL) があります。デフォルトでは 5 分です。DAX は項目キャッシュに書き込むすべての項目にタイムスタンプを割り当てます。TTL 設定より長期間キャッシュに残り続けている項目は、期限切れになります。期限切れの項目について `GetItem` リクエストを発行すると、キャッシュミスとみなされ、DAX は `GetItem` リクエストを DynamoDB に送信します。

> **Note**
>
> 項目キャッシュの TTL 設定は、新しい DAX クラスターを作成する際に指定できます。詳細については、「」を参照してくださいDAX クラスターの管理

DAX は項目キャッシュの LRU (least recently used) リストも保持します。LRU リストは、項目が最初にキャッシュに書き込まれた時とその項目が最後にキャッシュから読み込まれた時を記録して保存します。項目キャッシュがいっぱいになると、DAX は古い項目を (期限切れでない場合であっても) 削除し、新しい項目のためのスペースを空けます。LRU アルゴリズムは項目キャッシュでは常に有効であり、ユーザーが設定することはできません。

項目キャッシュ TTL 設定としてゼロを指定した場合、項目キャッシュ内の項目は、LRU 削除または「書き込みスルー」オペレーションによってのみ更新されます。

DAX での項目キャッシュの整合性の詳細については、「DAX 項目キャッシュの動作」を参照してください。

クエリキャッシュ

DAX は、クエリキャッシュを維持して、`Query` および `Scan` オペレーションからの結果を保存します。このキャッシュの項目は、DynamoDB テーブルに対するクエリおよびスキャンの結果セットを表します。これらの結果セットはパラメータ値別に保存されています。

アプリケーションが `Query` または `Scan` リクエストを送信すると、DAX は指定されたパラメータ値を使用してクエリキャッシュから一致する結果セットの読み込みを試みます。結果セットが見つかった場合 (キャッシュヒット)、DAX はすぐにアプリケーションに結果セットを返します。結果セットが見つからない場合 (キャッシュミス)、DAX は DynamoDB にリクエストを送信します。DynamoDB は、結果整合性のある読み取りを使用してリクエストを処理し、結果セットを DAX に返します。DAX は結果セットをクエリキャッシュに保存し、アプリケーションに返します。

> Note
>
> クエリキャッシュの TTL 設定は、新しい DAX クラスターを作成する際に指定できます。詳細については、「」を参照してくださいDAX クラスターの管理

DAX は、クエリキャッシュの LRU リストも保持します。リストは、結果セットが最初にキャッシュに書き込まれた時とその結果が最後にキャッシュから読み込まれた時を記録して保存します。クエリキャッシュがいっぱいになると、DAX は古い結果セットを (期限切れでなくても) 削除して新しい結果セットのためのスペースを解放します。LRU アルゴリズムはクエリキャッシュでは常に有効であり、ユーザーが設定することはできません。

クエリキャッシュ TTL 設定としてゼロを指定した場合、クエリ応答はキャッシュされません。

DAX でのクエリキャッシュの整合性の詳細については、「DAX クエリキャッシュの動作」を参照してください。

DAX クラスターコンポーネント

Amazon DynamoDB Accelerator (DAX) クラスターは、AWS のインフラストラクチャのコンポーネントで構成されます。このセクションでは、これらのコンポーネントと、それらがどのように連携するかについて説明します。

トピック

- ノード

- クラスター
- リージョンとアベイラビリティーゾーン
- パラメータグループ
- セキュリティグループ
- クラスター ARN
- クラスターエンドポイント
- ノードエンドポイント
- [サブネットグループ]
- イベント
- メンテナンスウィンドウ

ノード

ノードとは、DAX クラスターにおける最小の構成要素です。各ノードは DAX ソフトウェアのインスタンス 1 つを実行し、キャッシュされたデータのレプリカを 1 つ保持します。

DAX クラスターは 2 通りの方法のいずれかでスケールできます。

- クラスターにさらにノードを追加する。これは、クラスター全体の読み込みスループットを増加します。
- より大きなノードタイプを使用する。ノードタイプが大きくなると、容量が大きくなり、スループットを増やすことができます。(新しいノードタイプで新しいクラスターを作成する必要があります。)

クラスター内の各ノードは同じノードタイプであり、同じ DAX キャッシュソフトウェアを実行します。使用可能なノードタイプのリストについては、「Amazon DynamoDB の料金表」を参照してください。

クラスター

クラスターは、DAX がユニットとして管理する 1 つまたは複数のノードの論理グループです。クラスター内のノードの 1 つがプライマリノードとして指定され、他のノード (ある場合) はリードレプリカです。

プライマリノードは以下を担当します。

- キャッシュ済みデータに対するアプリケーションリクエストの対応。
- DynamoDB への書き込みオペレーションの処理。
- クラスターの削除ポリシーに従った、キャッシュからのデータの削除。

プライマリノードにキャッシュされたデータが変更されると、DAX は、レプリケーションログを使用して、その変更をすべてのリードレプリカノードに伝播します。すべてのリードレプリカから確認を受け取ると、DynamoDB はプライマリノードからレプリケーションログを削除します。

リードレプリカは以下を担当します。

- キャッシュ済みデータに対するアプリケーションリクエストの対応。
- クラスターの削除ポリシーに従った、キャッシュからのデータの削除。

ただし、プライマリノードとは異なり、リードレプリカは DynamoDB には書き込みません。

リードレプリカにはさらに 2 つの目的があります。

- スケーラビリティ。DAX への同時アクセスを必要とするクライアントが多数ある場合は、レプリカを追加して読み込みをスケーリングできます。DAX は、クラスター内のすべてのノードに均等にロードを分散します。(スループットを増やすもう 1 つの方法は、より大規模なキャッシュノードタイプを使用することです。)
- 高可用性。プライマリノード障害時に、DAX は自動的にリードレプリカにフェイルオーバーし、新しいプライマリとして指定します。レプリカノードに障害が発生しても、障害が発生したノードが回復するまで、DAX クラスター内の他のノードがリクエストを処理することができます。耐障害性を最大にするため、リードレプリカを異なるアベイラビリティーゾーンにデプロイする必要があります。この設定により、アベイラビリティーゾーン全体が使用できない場合でも DAX クラスターが機能し続けることができます。

DAX クラスターは、クラスターごとに最大 11 個のノード (プライマリノードと最大 10 個のリードレプリカ) をサポートできます。

> ⚠ Important
>
> 本稼働環境での使用においては、少なくとも 3 つのノードをそれぞれ異なるアベイラビリティーゾーンに置いて DAX を使用することを強くお勧めします。DAX クラスターが耐障害性を持つためには 3 つのノードが必要です。

DAX クラスターは、開発またはテストワークロードでは 1 つまたは 2 つのノードでデプロイできます。1 つまたは 2 つのノードクラスターでは耐障害性がないため、本稼働での使用では 3 つ未満のノードはお勧めしません。1 つまたは 2 つのノードクラスターでソフトウェアまたはハードウェアの障害が発生した場合、クラスターが使用できなくなったり、キャッシュ済みデータが失われることがあります。

リージョンとアベイラビリティーゾーン

ある AWS リージョンにある DAX クラスターは、同じリージョンにある DynamoDB テーブルとのみやり取りできます。したがって、適切なリージョンで DAX クラスターを起動することを確認してください。他のリージョンに DynamoDB テーブルがある場合は、そのリージョンにも DAX クラスターを起動する必要があります。

各リージョンは、他のリージョンと完全に分離されるように設計されています。各リージョンには複数のアベイラビリティーゾーンがあります。別のアベイラビリティーゾーンでノードを起動して、最大限の耐障害性を実現できます。

⚠ Important

単一アベイラビリティーゾーンにクラスターのすべてのノードを置かないでください。この設定では、アベイラビリティーゾーンの障害時に DAX クラスターが利用できなくなります。
本稼働環境での使用においては、少なくとも 3 つのノードをそれぞれ異なるアベイラビリティーゾーンに置いて DAX を使用することを強くお勧めします。DAX クラスターが耐障害性を持つためには 3 つのノードが必要です。
DAX クラスターは、開発またはテストワークロードでは 1 つまたは 2 つのノードでデプロイできます。1 つまたは 2 つのノードクラスターでは耐障害性がないため、本稼働での使用では 3 つ未満のノードはお勧めしません。1 つまたは 2 つのノードクラスターでソフトウェアまたはハードウェアの障害が発生した場合、クラスターが使用できなくなったり、キャッシュ済みデータが失われることがあります。

パラメータグループ

パラメータグループは、DAX クラスターのランタイム設定を管理するために使用されます。DAX には、パフォーマンスを最適化するために使用できるいくつかのパラメータ (キャッシュ済みデータの TTL ポリシーを定義するなど) があります。パラメータグループはクラスターに適用可能なパラメー

タの名前付きセットです。これにより、そのクラスター内のすべてのノードがまったく同じ方法で設定されていることを確認できます。

セキュリティグループ

DAX クラスターは、Amazon Virtual Private Cloud (Amazon VPC) 環境で実行されます。この環境は、AWS アカウント専用の仮想ネットワークであり、他の VPC からは独立しています。セキュリティグループは、VPC の仮想ファイアウォールとして機能し、インバウンドトラフィックとアウトバウンドネットワークトラフィックをコントロールできます。

VPC でクラスターを起動する際に、着信ネットワークトラフィックを許可する進入ルールをセキュリティグループに追加します。進入ルールは、クラスターのプロトコル (TCP) とポート番号 (8111) を指定します。このルールをセキュリティグループに追加すると、VPC 内で実行されるアプリケーションが DAX クラスターにアクセスできます。

クラスター ARN

各 DAX クラスターには Amazon リソースネーム (ARN) が割り当てられます。ARN 形式は次のとおりです。

```
arn:aws:dax:region:accountID:cache/clusterName
```

IAM ポリシー内でクラスター ARN を使用して、DAX API オペレーションのアクセス許可を定義します。詳細については、「DAX のアクセスコントロール」を参照してください。

クラスターエンドポイント

すべての DAX クラスターは、アプリケーションで使用できるクラスターエンドポイントを提供します。エンドポイントを使用してクラスターにアクセスすることで、アプリケーションでクラスターの個別のノードのホスト名とポート番号を把握する必要がなくなります。アプリケーションは、リードレプリカを追加または削除した場合でも、自動的にクラスターのすべてのノードを「把握」します。

転送中に暗号化を使用するように設定されていない、us-east-1 リージョンのクラスターエンドポイントの例を以下に示します。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

転送中に暗号化を使用するように設定されている、同じリージョン内のクラスターエンドポイントの例を以下に示します。

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

ノードエンドポイント

DAX クラスターのノードそれぞれに独自のホスト名とポート番号があります。ノードエンドポイントの例を以下に示します。

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

アプリケーションは、エンドポイントを使用してノードに直接アクセスできます。しかし、DAX クラスターを単一のユニットとして扱い、代わりにクラスターのエンドポイントを使用してアクセスすることをお勧めします。クラスターエンドポイントを使用することで、アプリケーションでノードのリストを保持し、クラスターでノードが追加または削除されたときにリストを最新の状態に保つ必要がなくなります。

[サブネットグループ]

DAX クラスターノードへのアクセスは、Amazon VPC 環境内の Amazon EC2 インスタンスで実行されるアプリケーションのみに制限されます。サブネットグループを使用して、特定のサブネットで実行されている Amazon EC2 インスタンスからのクラスターアクセスを許可できます。サブネットグループは、Amazon VPC 環境で実行しているクラスターに対して指定できるサブネット (通常はプライベート) の集合です。

DAX クラスターを作成するときに、サブネットグループを指定する必要があります。DAX はそのキャッシュサブネットグループを使用して、そのサブネット内でノードに関連付けるサブネットおよび IP アドレスを選択します。

イベント

DAX は、ノードの追加の失敗、ノードの追加の成功、セキュリティグループの変更など、重要なイベントをクラスター内に記録します。主要イベントをモニタリングすることで、クラスターの現在の状態を知り、イベントに基づいて是正措置を取ることができます。AWS Management Console、または DAX 管理 API の `DescribeEvents` アクションを使用して、これらのイベントにアクセスできます。

特定の Amazon Simple Notification Service (Amazon SNS) トピックに通知を送信するようにリクエストすることもできます。そうすれば、DAX クラスターでイベントが発生したことがすぐにわかります。

メンテナンスウィンドウ

すべてのクラスターには、週ごとのメンテナンス時間があります。その時間内にシステムの変更が適用されます。キャッシュクラスターの作成または変更時に、必要なメンテナンス期間を指定しない場合、DAX では、ランダムに選択された曜日に対して 60 分のメンテナンス期間が割り当てられます。

60 分のメンテナンスウィンドウは、AWS リージョンごとに定められた 8 時間の時間ブロックからランダムに選択されます。次の表に、デフォルトでメンテナンス時間が割り当てられる各リージョンの時間ブロックを示します。

リージョンコード	リージョン名	メンテナンスウィンドウ
ap-northeast-1	Asia Pacific (Tokyo) Region	13:00～21:00 UTC
ap-southeast-1	Asia Pacific (Singapore) Region	14:00～22:00 UTC
ap-southeast-2	Asia Pacific (Sydney) Region	12:00～20:00 UTC
ap-south-1	Asia Pacific (Mumbai) Region	17:30～1:30 UTC
cn-northwest-1	中国 (寧夏) リージョン	23:00～07:00 UTC
cn-north-1	中国 (北京) リージョン	14:00～22:00 UTC
eu-central-1	Europe (Frankfurt) Region	23:00～07:00 UTC
eu-west-1	欧州 (アイルランド) リージョン	22:00～06:00 UTC
eu-west-2	Europe (London) Region	23:00～07:00 UTC
eu-west-3	欧州 (パリ) リージョン	23:00～07:00 UTC
sa-east-1	South America (São Paulo) Region	01:00～09:00 UTC
us-east-1	米国東部 (バージニア北部) リージョン	03:00～11:00 UTC

リージョンコード	リージョン名	メンテナンスウィンドウ
us-east-2	米国東部 (オハイオ) リージョン	23:00 ~ 07:00 UTC
us-west-1	US West (N. California) リージョン	06:00 ~ 14:00 UTC
us-west-2	米国西部 (オレゴン) リージョン	06:00 ~ 14:00 UTC

メンテナンスウィンドウは使用率の最も低い時間帯に設定する必要があります。このため、場合によっては変更が必要になります。お客様がリクエストしたメンテナンス作業が発生する時間範囲を最大 24 時間で指定できます。

DAX クラスターの作成

このセクションでは、デフォルトの Amazon Virtual Private Cloud (Amazon VPC) 環境で Amazon DynamoDB Accelerator (DAX) を最初にセットアップし使用する手順を順を追って説明します。最初の DAX クラスターは、AWS Command Line Interface (AWS CLI) または AWS Management Console のいずれかを使用して作成できます。

DAX クラスターを作成すると、同じ VPC で実行されている Amazon EC2 インスタンスからアクセスできるようになります。その後、アプリケーションプログラムで DAX クラスターを使用できるようになります。詳細については、「DynamoDB Accelerator (DAX) クライアントで開発する」を参照してください。

トピック

- DynamoDB にアクセスする DAX 用の IAM サービスロールを作成します
- AWS CLI を使用した DAX クラスターの作成
- AWS Management Console を使用した DAX クラスターの作成

DynamoDB にアクセスする DAX 用の IAM サービスロールを作成します

DAX クラスターがユーザーに代わって DynamoDB テーブルにアクセスできるようにサービスロールを作成する必要があります。サービスロールは、ユーザーに代わって AWS のサービスを承認す

る AWS Identity and Access Management (IAM) ロールです。サービスロールは、ユーザーがテーブルそのものにアクセスしているかのように、DAX に DynamoDB テーブルへのアクセスを許可します。DAX クラスターを作成する前にサービスロールを作成する必要があります。

コンソールを使用している場合は、クラスターを作成するワークフローで DAX サービスロールの有無が確認されます。何も見つからない場合は、コンソールは新しいサービスロールを作成します。詳細については、「」を参照してくださいthe section called “ステップ 2: DAX クラスターを作成”

AWS CLI を使用している場合は、すでに作成してある DAX サービスロールを指定するか、事前に新しいサービスロールを作成する必要があります。詳細については、「」を参照してくださいステップ 1: AWS CLI を使用して、DAX から DynamoDB にアクセスするための IAM サービスロールを作成する

サービスロールの作成に必要なアクセス許可

AWS マネージド `AdministratorAccess` ポリシーには、DAX クラスターおよびサービスロールの作成に必要なすべてのアクセス許可が含まれています。ユーザーに `AdministratorAccess` がアタッチされている場合、それ以上のアクションは不要です。

アタッチされていない場合は、IAM ポリシーに次のアクセス許可を追加して、ユーザーがサービスロールを作成できるようにする必要があります。

- `iam:CreateRole`
- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

アクションの実行を試みるユーザーにこれらのアクセス許可をアタッチする必要があります。

> **Note**
>
> `iam:CreateRole`、`iam:CreatePolicy`、`iam:AttachRolePolicy`、および `iam:PassRole` の各アクセス許可は、DynamoDB の AWS マネージドポリシーには含まれていません。これは意図的なものです。これらのアクセス許可が含まれていると、特権のエスカレーションが可能になり、ユーザーがこれらのアクセス権限を使用して新しい管理者ポリシーを作成し、それを既存のロールにアタッチできるようになるからです。そのため、DAX クラスターの管理者は、これらのアクセス権限を自分のポリシーに明示的に追加する必要があります。

トラブルシューティング

ユーザーポリシーに iam:CreateRole、iam:CreatePolicy、iam:AttachPolicy の各アクセス許可が含まれていないと、エラーメッセージが表示されます。次の表に、これらのメッセージを示し、問題を解決する方法を説明します。

次のエラーメッセージが表示された場合は ...。	次の作業を行います。
`User: arn:aws:iam:: *accountID* :user/*userName* is not authorized to perform: iam:CreateRole on resource: arn:aws:iam:: *accountID* :role/service-role/ *roleName*`	`iam:CreateRole` をユーザーポリシーに追加します。
`User: arn:aws:iam:: *accountID* :user/*userName* is not authorized to perform: iam:CreatePolicy on resource: policy *policyName*`	`iam:CreatePolicy` をユーザーポリシーに追加します。
`User: arn:aws:iam:: *accountID* :user/*userName* is not authorized to perform: iam:AttachRolePolicy on resource: role *daxServiceRole*`	`iam:AttachRolePolicy` をユーザーポリシーに追加します。

DAX クラスターの管理に必要な IAM ポリシーの詳細については、「DAX のアクセスコントロール」を参照してください。

AWS CLI を使用した DAX クラスターの作成

このセクションでは、AWS Command Line Interface (AWS CLI) を使用して Amazon DynamoDB Accelerator (DAX) クラスターを作成する方法を説明します。まだ AWS CLI をインストールして設定していない場合は、インストールして設定する必要があります。これを行うには、「AWS Command Line Interface ユーザーガイド」の手順を参照してください。

- AWS CLI のインストール
- AWS CLI の設定

⚠ Important

AWS CLI を使用して DAX クラスターを管理するには、バージョン 1.11.110 以上をインストールするか、アップグレードしてください。

AWS CLI のすべての例で、`us-west-2` リージョンと架空のアカウント ID を使用しています。

トピック

- ステップ 1: AWS CLI を使用して、DAX から DynamoDB にアクセスするための IAM サービスロールを作成する
- ステップ 2: サブネットグループの作成
- ステップ 3: AWS CLI を使用して DAX クラスターを作成
- ステップ 4: AWS CLI を使用してセキュリティグループのインバウンドルールを設定

ステップ 1: AWS CLI を使用して、DAX から DynamoDB にアクセスするための IAM サービスロールを作成する

Amazon DynamoDB Accelerator (DAX) クラスターを作成する前に、クラスター用のサービスロールを作成する必要があります。サービスロールは、ユーザーに代わって AWS のサービスを承認する AWS Identity and Access Management (IAM) ロールです。サービスロールは、ユーザーがテーブルそのものにアクセスしているかのように、DAX に DynamoDB テーブルへのアクセスを許可します。

このステップでは、IAM ポリシーを作成し、次にそのポリシーを IAM ロールにアタッチします。こうすることで、DAX クラスターにロールを割り当て、ユーザーの代わりに DynamoDB オペレーションを実行させることができます。

DAX の IAM サービスロールを作成するには

1. 次の内容で、`service-trust-relationship.json` というファイルを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. サービスロールを作成します。

```
aws iam create-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --assume-role-policy-document file://service-trust-relationship.json
```

3. 次の内容で、`service-role-policy.json` というファイルを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:accountID:*"
            ]
        }
    ]
}
```

accountID をご自分の AWS アカウント ID に置き換えます。AWS アカウント ID を見つけるには、コンソールの右上隅で、ログイン ID を選択します。ドロップダウンメニューに AWS アカウント ID が表示されます

この例の Amazon リソースネーム (ARN) では、*accountID* は 12 桁の数字である必要があります。ハイフンなどの区切り文字を使用しないでください。

4. サービスロールの IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

出力では、以下の例のように、作成したポリシーの ARN に注意してください。

`arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess`

5. ポリシーをサービスロールにアタッチします。以下のコードにある *arn* を、前のステップで書き留めた実際のロール ARN に置き換えます。

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

次に、デフォルト VPC のサブネットグループを指定します。サブネットグループは、VPC 内の 1 つ以上のサブネットのコレクションです。「」を参照してくださいステップ 2: サブネットグループの作成

ステップ 2: サブネットグループの作成

この手順に従って、AWS Command Line Interface (AWS CLI) を使用して Amazon DynamoDB Accelerator (DAX) クラスターのサブネットグループを作成します。

> **Note**
>
> デフォルト VPC のサブネットグループを作成済みである場合は、このステップを省略できます。

DAX は、Amazon Virtual Private Cloud (Amazon VPC) 環境内で実行するように設計されています。2013 年 12 月 4 日以降に AWS アカウントを作成した場合は、各 AWS リージョンにデフォルトで VPC が用意されています。詳細については、「Amazon VPC ユーザーガイド」の「デフォルト VPC とデフォルトサブネット」を参照してください。

サブネットグループを作成するには

1. デフォルト VPC の ID を確認するには、次のコマンドを入力します。

   ```
   aws ec2 describe-vpcs
   ```

   出力では、以下の例のように、デフォルト VPC の識別子に注意してください。

   vpc-12345678

2. デフォルト VPC に関連付けられているサブネットの ID を確認します。*vpcID* を実際の VPC ID に置き換えます (vpc-12345678 など)。

   ```
   aws ec2 describe-subnets \
       --filters "Name=vpc-id,Values=vpcID" \
       --query "Subnets[*].SubnetId"
   ```

   出力で、サブネット ID を記録します (subnet-11111111 など)。

3. サブネットグループを作成します。--subnet-ids パラメータに少なくとも 1 つのサブネット ID を指定してください。

   ```
   aws dax create-subnet-group \
       --subnet-group-name my-subnet-group \
       --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
   ```

クラスターを作成するには、「ステップ 3: AWS CLI を使用して DAX クラスターを作成」を参照してください。

ステップ 3: AWS CLI を使用して DAX クラスターを作成

次の手順に従って、AWS Command Line Interface (AWS CLI) を使用して、デフォルト Amazon VPC に Amazon DynamoDB Accelerator (DAX) クラスターを作成します。

DAX クラスターを作成するには

1. サービスロールの Amazon リソースネーム (ARN) を取得します。

   ```
   aws iam get-role \
       --role-name DAXServiceRoleForDynamoDBAccess \
   ```

```
    --query "Role.Arn" --output text
```

出力では、以下の例のように、サービスロール ARN に注意してください。

`arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess`

2. DAX クラスターを作成します。*roleARN* を、前のステップで書き留めた ARN に置き換えます。

```
aws dax create-cluster \
    --cluster-name mydaxcluster \
    --node-type dax.r4.large \
    --replication-factor 3 \
    --iam-role-arn roleARN \
    --subnet-group my-subnet-group \
    --sse-specification Enabled=true \
    --region us-west-2
```

クラスター内のすべてのノードは、タイプ `dax.r4.large` (`--node-type`) です。3つのノード (`--replication-factor`) があります (1 つのプライマリノードと 2 つのレプリカ)。

> Note
>
> sudo と grep が予約キーワードである場合、クラスター名にこれらの語句を含む DAX クラスターを作成することはできません。たとえば、sudo と sudocluster は無効なクラスター名です。

クラスターステータスを表示するには、次のコマンドを入力します。

```
aws dax describe-clusters
```

ステータスは出力に表示されます (`"Status": "creating"` など)。

> Note
>
> クラスターを作成するには数分かかります。クラスターが使用可能になると、ステータスは `available` に変わります。その間に、「ステップ 4: AWS CLI を使用してセキュリティグループのインバウンドルールを設定」に進んでその指示に従います。

ステップ 4: AWS CLI を使用してセキュリティグループのインバウンドルールを設定

Amazon DynamoDB Accelerator (DAX) クラスター内のノードは、Amazon VPC のデフォルトのセキュリティグループを使用します。デフォルトのセキュリティグループでは、暗号化されていないクラスターには TCP ポート 8111、暗号化されたクラスターには 9111 のインバウンドトラフィックを許可する必要があります。これで Amazon VPC の Amazon EC2 インスタンスは DAX クラスターにアクセスできます。

> **Note**
>
> 異なるセキュリティグループ (`default` 以外) で DAX クラスターを起動した場合、グループに対してこの手順を代わりに実行する必要があります。

セキュリティグループのインバウンドルールを設定するには

1. デフォルトセキュリティグループの ID を確認するには、次のコマンドを入力します。*`vpcID`* を実際の VPC ID (「ステップ 2: サブネットグループの作成」を参照) に置き換えます。

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   出力で、セキュリティグループ ID を記録します (`sg-01234567` など)。

2. 次のように入力します。*`sgID`* を実際のセキュリティグループ ID に置き換えます。暗号化されていないクラスターの場合はポート 8111 を、暗号化されたクラスターの場合は 9111 を使用します。

   ```
   aws ec2 authorize-security-group-ingress \
       --group-id sgID --protocol tcp --port 8111
   ```

AWS Management Console を使用した DAX クラスターの作成

このセクションでは、AWS Management Consoleを使用して Amazon DynamoDB Accelerator (DAX) クラスターを作成する方法を説明します。

トピック

- ステップ 2: AWS Management Console を使用して DAX クラスターを作成
- ステップ 3: AWS Management Console を使用してセキュリティグループのインバウンドルールを設定

ステップ 1: AWS Management Console を使用してサブネットグループを作成

この手順に従って、AWS Management Console を使用して Amazon DynamoDB Accelerator (DAX) クラスターのサブネットグループを作成します。

> ⓘ Note
>
> デフォルト VPC のサブネットグループを作成済みである場合は、このステップを省略できます。

DAX は、Amazon Virtual Private Cloud (Amazon VPC) 環境内で実行するように設計されています。2013 年 12 月 4 日以降に AWS アカウントを作成した場合は、各 AWS リージョンにデフォルトで VPC が用意されています。詳細については、「Amazon VPC ユーザーガイド」の「デフォルト VPC とデフォルトサブネット」を参照してください。

DAX クラスターの作成プロセスの一部として、サブネットグループを指定する必要があります。サブネットグループは、VPC 内の 1 つ以上のサブネットのコレクションです。DAX クラスターを作成する際、ノードがサブネットグループ内のサブネットにデプロイされます。

サブネットグループを作成するには

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ナビゲーションペインの、[DAX] で、[サブネットグループ]を選択します。
3. [サブネットグループの作成] を選択します。
4. [サブネットグループの作成] ウィンドウで以下を行います。

   a. [Name] (名前) - サブネットグループの短い名前を入力します。

   b. [Description] (説明) - サブネットグループの説明を入力します。

   c. VPC ID - Amazon の VPC 環境の ID を選択します。

   d. [Subnets] (サブネット) - リストから 1 つ以上のサブネットを選択します。

> Note
>
> サブネットは、複数のアベイラビリティーゾーンに分散されます。マルチノード DAX クラスター (プライマリノードと 1 つ以上のリードレプリカ) の作成を計画する場合は、複数のサブネット ID を選択することをお勧めします。その後、DAX はクラスターノードを複数のアベイラビリティーゾーンにデプロイできます。アベイラビリティーゾーンが使用できなくなると、DAX は自動的に残っているアベイラビリティーゾーンにフェイルオーバーします。DAX クラスターは中断することなく機能し続けます。

設定が正しいことを確認したら、[サブネットグループの作成] を選択します。

クラスターを作成するには、「ステップ 2: AWS Management Console を使用して DAX クラスターを作成」を参照してください。

ステップ 2: AWS Management Console を使用して DAX クラスターを作成

次の手順に従って、デフォルト Amazon VPC に Amazon DynamoDB Accelerator (DAX) クラスターを作成します。

DAX クラスターを作成するには

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ナビゲーションペインの、[DAX] で、[クラスター]を選択します。
3. [クラスターを作成] を選択します。
4. [クラスターの作成] ウィンドウで、以下の操作を行います。

    a. [クラスター名] - DAX クラスターの短縮名を入力します。

    > Note
    >
    > sudo と grep が予約キーワードである場合、クラスター名にこれらの語句を含む DAX クラスターを作成することはできません。たとえば、sudo と sudocluster は無効なクラスター名です。

    b. [Cluster description] (クラスターの説明) - クラスターの説明を入力します。

c. ノードタイプ - クラスター内のすべてのノードのノードタイプを選択します。

d. [Cluster size] (クラスターサイズ) - クラスター内のノードの数を選択します。クラスターは 1 つのプライマリノードと最大 9 つのリードレプリカで構成されます。

> Note
>
> シングルノードクラスターを作成する場合は、[1] を選択します。クラスターは 1 つのプライマリノードで構成されます。
> マルチノードクラスターを作成する場合は、[3] (プライマリノード 1 つとリードレプリカ 2 つ) ~ [10] (プライマリノード 1 つとリードレプリカ 9 つ) の数字を選択します。
>
> > Important
> >
> > 本稼働環境での使用においては、少なくとも 3 つのノードをそれぞれ異なるアベイラビリティーゾーンに置いて DAX を使用することを強くお勧めします。DAX クラスターが耐障害性を持つためには 3 つのノードが必要です。
> > DAX クラスターは、開発またはテストワークロードでは 1 つまたは 2 つのノードでデプロイできます。1 つまたは 2 つのノードクラスターでは耐障害性がないため、本稼働での使用では 3 つ未満のノードはお勧めしません。1 つまたは 2 つのノードクラスターでソフトウェアまたはハードウェアの障害が発生した場合、クラスターが使用できなくなったり、キャッシュ済みデータが失われることがあります。

e. [Next] を選択します。

f. サブネットグループ - [既存のものを選択] を選択し、ステップ 1: AWS Management Console を使用してサブネットグループを作成 で作成したサブネットグループを選択します。

g. アクセスコントロール - デフォルトのセキュリティグループを選択します。

h. アベイラビリティーゾーン (AZ) - [自動] を選択します。

i. [次へ] を選択します。

j. [IAM service role for DynamoDB access] (DynamoDB アクセス用の IAM サービスロール) - [Create new] (新規作成) を選択し、次の情報を入力します。

- IAM ロール名 - IAM ロールの名前 (`DAXServiceRole` など) を入力します。コンソールは新しい IAM ロールを作成し、DAX クラスターは実行時にこのロールを割り当てます。
- [ポリシーの作成] の横にあるボックスを選択します。
- [IAM role policy] (IAM ロールポリシー) - [Read/Write] (読み込み/書き込み) を選択します。これにより、DAX クラスターが DynamoDB で読み込み/書き込み操作を実行できるようになります。
- 新しい IAM ポリシー名 — IAM ロール名を入力すると、このフィールドが入力されます。IAM ポリシーの名前も取得できます (`DAXServicePolicy` など)。コンソールは新しい IAM ポリシーを作成し、ポリシーを IAM ロールにアタッチします。
- DynamoDB テーブルへのアクセス — [すべてのテーブル] を選択します。

k. 暗号化 - [保管時の暗号化を有効にする] および [転送時の暗号化を有効にする] を選択します。詳細については、保管時の DAX 暗号化および転送時の DAX 暗号化を参照してください。

DAX が Amazon EC2 にアクセスするための別のサービスロールも必要です。このサービスロールは DAX によって自動的に作成されます。詳細については、「DAX のサービスにリンクされたロールの使用」を参照してください。

5. すべての設定が正しいことを確認したら、[次へ] を選択します。
6. パラメータグループ — [既存のものを選択] を選択します。
7. メンテナンスウィンドウ — ソフトウエアのアップグレードが適用されるタイミングに指定がない場合は、[指定なし] を選択するか、[時間枠を指定] を選択し、[平日]、[時間 (UTC)]、[開始時間 (時間)] のオプションを指定して、メンテナンスウィンドウのスケジュールを設定します。
8. タグ: — [新しいタグを追加] を選択して、タグ付けするキーと値のペアを入力します。
9. [Next] を選択します。

[確認と作成] 画面では、すべての設定を確認できます。クラスターを作成する準備ができたら、[クラスターの作成] を選択します。

[クラスター] 画面で、DAX クラスターが [作成中] というステータスで一覧表示されます。

> Note
>
> クラスターを作成するには数分かかります。クラスターが使用可能になると、ステータスは [Available (使用可能)] に変わります。

その間に、「ステップ 3: AWS Management Console を使用してセキュリティグループのインバウンドルールを設定」に進んでその指示に従います。

ステップ 3: AWS Management Console を使用してセキュリティグループのインバウンドルールを設定

Amazon DynamoDB Accelerator (DAX) クラスターは通信に TCP ポート 8111 (暗号化されていないクラスター) または 9111 (暗号化されたクラスター) を使用するため、このポートのインバウンドトラフィックを認証する必要があります。これで Amazon VPC の Amazon EC2 インスタンスは DAX クラスターにアクセスできます。

> **Note**
>
> 異なるセキュリティグループ (`default` 以外) で DAX クラスターを起動した場合、グループに対してこの手順を代わりに実行する必要があります。

セキュリティグループのインバウンドルールを設定するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[Security Groups] を選択します。
3. [デフォルト] セキュリティグループを選択します。[アクション] メニューで、[インバウンドルールの編集] を選択します。
4. [ルールの追加] を選択し、次の情報を入力します。
   - [ポート範囲] - 8111 (クラスターが暗号化されていない場合) または 9111 (クラスターが暗号化されている場合)を入力します。
   - ソース — [カスタム] のままにして、右側の検索フィールドを選択します。ドロップダウンメニューが表示されます。デフォルトのセキュリティグループの識別子を選択します。
5. [Save rules] (ルールの保存) を選択して、変更を保存します。
6. コンソールの名前を更新するには、[名前] プロパティに移動して、表示される [編集] オプションを選択します。

DAX および DynamoDB の整合性モデル

Amazon DynamoDB Accelerator (DAX) は、DynamoDB テーブルにキャッシュを追加するプロセスを簡素化するために設計された書き込みスルーキャッシュサービスです。DAX は DynamoDB とは別に動作するため、アプリケーションが意図どおりに動作することを確認するには DAX と DynamoDB の両方の整合性モデルを理解することが重要です。

多くのユースケースでは、アプリケーションでの DAX の使用方法が、DAX クラスター内のデータの整合性、および DAX と DynamoDB 間のデータの整合性に影響します。

トピック

DAX クラスターノード間の整合性

アプリケーションの高可用性を実現するには、DAX クラスターのプロビジョニングには少なくとも 3 つのノードを使用してください。次にこれらのノードをリージョン内の複数のアベイラビリティーゾーンに配置します。

クラスターの実行中、データがクラスター内のすべてのノードにレプリケートされます (複数のノードをプロビジョニング済みの場合)。DAX を使用して `UpdateItem` を正常に実行するアプリケーションの場合を考えてみましょう。このアクションにより、プライマリノードの項目キャッシュが新しい値で変更されます。この値は、クラスター内の他のすべてのノードにレプリケートされます。このレプリケーションの結果、整合性が保たれます。完了までにかかる時間は通常は 1 秒未満です。

このシナリオでは、2 つのクライアントが同じ DAX クラスターから同じキーを読み取った時に、それぞれのクライアントがアクセスしたノードによって、異なる値を受け取る可能性があります。更新がクラスター内のすべてのノードで完全にレプリケートされると、ノードはすべて整合性がとれます。(この動作は、DynamoDB の結果整合性特性と似ています。)

DAX を使用するアプリケーションを構築する場合、そのアプリケーションは結果整合性データを許容するように設計されている必要があります。

DAX 項目キャッシュの動作

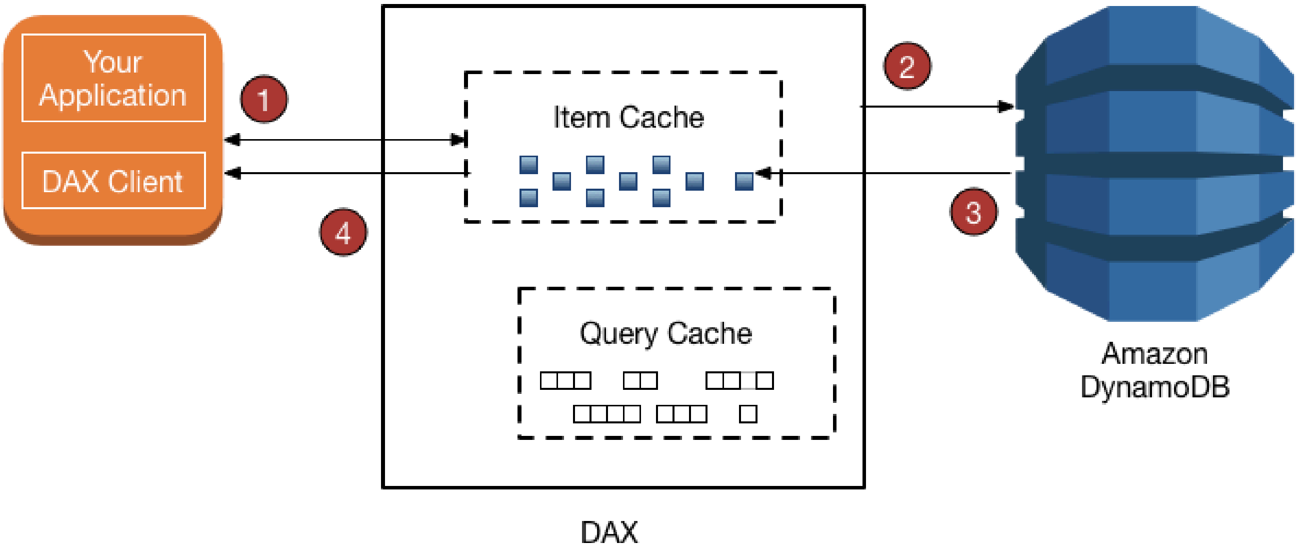
各 DAX クラスターには、2 つの異なるキャッシュ (項目キャッシュとクエリキャッシュ) があります。詳細については、「DAX: 仕組み」を参照してください。

このセクションでは、DAX 項目キャッシュに対する読み込みおよび書き込みの整合性の実装について取り扱います。

読み込みの整合性

DynamoDB では、`GetItem` オペレーションはデフォルトで結果整合性のある読み込みを行います。DynamoDB クライアントで `UpdateItem` を使用するとします。すぐ後に同じ項目を読み取ろうとすると、更新前のデータが表示される場合があります。これは、DynamoDB ストレージロケーション全体への伝播が遅れるためです。整合性は通常、数秒以内に達成します。そのため、読み込みを再試行すると、更新された項目が表示されます。

DAX クライアントで `GetItem` を使用する場合、オペレーション (この場合は結果整合性のある読み込み) は次のように進行します。



1. DAX クライアントが `GetItem` リクエストを発行します。DAX は、項目キャッシュからリクエストされた項目を読み取ろうとします。項目がキャッシュ内にある場合 (キャッシュヒット)、DAX によりアプリケーションに返されます。

2. 項目がない場合 (キャッシュミス)、DAX は DynamoDB に対して結果整合性 GetItem オペレーションを実行します。
3. DynamoDB からリクエストされた項目が返され、DAX で項目キャッシュに保存されます。
4. DAX はその項目をアプリケーションに返します。
5. (非表示) DAX クラスターに複数のノードがある場合、項目がクラスター内の他のすべてのノードにレプリケートされます。

項目は、キャッシュの有効期限 (TTL) 設定および最も長い間使用されていない (LRU) アルゴリズムによりますが、DAX 項目キャッシュに残ります。詳細については、「DAX: 仕組み」を参照してください。

ただし、この期間は、DAX は項目を DynamoDB から再ロードしません。他のユーザーが DynamoDB クライアントを使用して、DAX 全体をバイパスして項目を更新すると、DAX クライアントを使用した GetItem リクエストで、DynamoDB クライアントを使用した GetItem リクエストとは異なる結果が呼び出されます。このシナリオでは、DAX と DynamoDB は、DAX 項目の TTL が期限切れになるまで、同じキーに対して矛盾する値を保持します。

アプリケーションが をバイパスして基礎となる テーブルのデータを変更する場合は、そのアプリケーションはデータの不整合が発生する可能性を予期し許容する必要があります。

> Note
>
> GetItem に加えて、DAX クライアントは BatchGetItem リクエストもサポートします。BatchGetItem は基本的には 1 つ以上の GetItem リクエストをまとめるラッパーです。そのため、DAX ではそれぞれを個別の GetItem オペレーションとして扱います。

書き込みの整合性

DAX は書き込みスルーキャッシュであり、DAX の項目キャッシュが基礎となる DynamoDB テーブルに対して整合性を保つためのプロセスを簡易化します。

DAX クライアントは、DynamoDB と同じ書き込み API オペレーションをサポートします (PutItem、UpdateItem、DeleteItem、BatchWriteItem、および TransactWriteItems)。DAX クライアントでこれらのオペレーションを使用すると、項目は DAX と DynamoDB の両方で変更されます。DAX は、その TTL 値に関係なく項目キャッシュ内の項目を更新します。

たとえば、DAX クライアントから GetItem リクエストを発行して、ProductCatalog テーブルから項目を読み取るとします。(パーティションキーは Id で、ソートキーはありません。) Id が 101 である項目を取得します。その項目の QuantityOnHand 値は 42 です。DAX は特定の TTL で項目を項目キャッシュに保存します。この例では、TTL が 10 分とします。3 分後、別のアプリケーションが DAX クライアント使用して、QuantityOnHand 値が 41 になるように同じ項目を更新します。その後はその項目が更新されないとすると、その後 10 分間の同じ項目に対する後続の読み込みは、QuantityOnHand に対してキャッシュされた値を返します (41)。

DAX の書き込み処理方法

DAX は、高パフォーマンスの読み込みを必要とするアプリケーションを対象としています。書き込みスルーキャッシュとして、DAX は書き込みを DynamoDB に同期的に渡し、その結果生じた更新をクラスター内のすべてのノードで項目キャッシュに自動的に非同期にレプリケートします。キャッシュ無効化ロジックを管理する必要はありません。DAX がお客様の代わりにそれを行います。

DAX では、PutItem、UpdateItem、DeleteItem、BatchWriteItem、および TransactWriteItems の各書き込みオペレーションがサポートされます。

これらの PutItem、UpdateItem、DeleteItem、または BatchWriteItem リクエストを DAX に送信すると、次のことが発生します。

- DAX が DynamoDB にリクエストを送信します。
- DynamoDB が DAX に応答し、書き込みが成功したことが確認されます。
- DAX がキャッシュ項目に項目を書き込みます。
- DAX がリクエスタに成功を返します。

これらの TransactWriteItems リクエストを DAX に送信すると、次のことが発生します。

- DAX が DynamoDB にリクエストを送信します。
- DynamoDB が DAX に応答し、トランザクションが完了したことが確認されます。
- DAX がリクエスタに成功を返します。
- バックグラウンドで、DAX は項目キャッシュに項目を格納するために、TransactWriteItems リクエスト内の各項目に対して TransactGetItems リクエストを行います。TransactGetItems は、直列化可能分離を確実にするために使用されます。

スロットルなどの理由で DynamoDB への書き込みが失敗した場合、項目は DAX にキャッシュされません。失敗の例外がリクエスタに返されます。これにより、データが先に DynamoDB に正常に書き込まれない限り、DAX には書き込まれません。

> **Note**
>
> DAX へのすべての書き込みは、項目キャッシュの状態を変更します。ただし、項目キャッシュへの書き込みはクエリキャッシュに影響しません。(DAX の項目キャッシュとクエリキャッシュは異なる目的で使用されます。オペレーションは互いに独立しています。)

DAX クエリキャッシュの動作

DAX は、Query および Scan リクエストからの結果をクエリキャッシュにキャッシュします。ただし、これらの結果は項目キャッシュにまったく影響しません。アプリケーションが Query または Scan 要求を発行すると、結果セットは項目キャッシュではなく、クエリキャッシュに保存されます。Scan オペレーションを実行して項目キャッシュを「ウォームアップ」することはできません。項目キャッシュとクエリキャッシュは別々の存在です。

クエリ-更新-クエリの整合性

項目キャッシュ、または基礎となる DynamoDB テーブルに対する更新によって、クエリキャッシュに保存されている結果が無効化されたり変更されたりすることはありません。

説明のため、以下のシナリオを想定してください。アプリケーションは、パーティションキーとして DocId を使用し、ソートキーとして RevisionNumber を使用する DocumentRevisions テーブルを使用しています。

1. 5 より大きいか等しい Query を持つすべての項目に対して、クライアントが DocId 101 に RevisionNumber を発行します。結果を DAX クエリキャッシュに保存し、ユーザーに結果セットを返します。
2. クライアントは、RevisionNumber 値が 20 の DocId 101 に対して PutItem リクエストを発行します。
3. クライアントはステップ 1 の説明と同じ Query を発行します (DocId 101 および RevisionNumber >= 5)。

このシナリオでは、ステップ 3 で発行された Query のキャッシュ結果セットは、ステップ 1 でキャッシュされた結果セットと同じです。これは、DAX では Query または Scan の結果セットを個

別の項目に対する更新に基づいて無効化しないためです。ステップ 2 の PutItem オペレーションは、Query の TTL の有効期限が切れたときにのみ DAX クエリキャッシュに反映されます。

アプリケーションでは、クエリキャッシュの TTL 値、およびクエリキャッシュと項目キャッシュ間の結果の不一致をアプリケーションで許容できる期間を考慮に入れる必要があります。

強力な整合性のあるトランザクション読み込み

強い整合性のある GetItem、BatchGetItem、Query、または Scan リクエストを実行するには、ConsistentRead パラメータを true に設定します。DAX は強力な整合性のある読み込みリクエストを DynamoDB に渡します。DynamoDB から返答を受け取ると、DAX はクライアントに結果を返しますが、結果をキャッシュしません。DAX は DynamoDB と緊密には連携していないため、単独では強い整合性のある読み込みを提供できません。このため、DAX からの後続の読み込みは、結果整合性のある読み込みである必要があります。そして、後続の強い整合性のある読み込みはすべて DynamoDB に渡す必要があります。

DAX は、TransactGetItems リクエストを一貫性のある読み込みと同じ方法で処理します。DAX はすべての TransactGetItems リクエストを DynamoDB に渡します。DynamoDB から返答を受け取ると、DAX はクライアントに結果を返しますが、結果をキャッシュしません。

ネガティブキャッシング

DAX は、項目キャッシュとクエリキャッシュの両方でネガティブキャッシュエントリをサポートします。ネガティブキャッシュエントリは、DAX が基礎となる DynamoDB テーブルでリクエストされた項目を見つけられなかったときに発生します。DAX は、エラーを生成する代わりに空の結果をキャッシュし、その結果をユーザーに返します。

たとえば、アプリケーションが GetItem リクエストを DAX クラスターに送信し、DAX 項目キャッシュの中に一致する項目がなかったとします。この場合、DAX は基礎となる DynamoDB テーブルから対応する項目を読み込むことになります。項目が DynamoDB に存在しない場合、DAX は空の項目を項目キャッシュに保存し、その空の項目をアプリケーションに返します。次に、アプリケーションが別の GetItem リクエストを同じ項目に送信するとします。DAX は、項目キャッシュ内に空の項目を見つけ、それをすぐにアプリケーションに返します。DynamoDB は一切参照されません。

ネガティブキャッシュエントリは、項目 TTL の有効期限が切れるか、LRU が呼び出されるか、PutItem、UpdateItem、または DeleteItem を使用して項目が変更されるまで、DAX 項目キャッシュに残ります。

DAX クエリキャッシュも同様の方法でネガティブキャッシュ結果を処理します。アプリケーションが Query または Scan を実行し、DAX クエリキャッシュにキャッシュされた結果がない場合、DAX

は DynamoDB にリクエストを送信します。結果セットに一致する項目がない場合、DAX は空の項目セットをクエリキャッシュに保存し、その空の結果セットをアプリケーションに返します。それ以降の `Query` または `Scan` リクエストは、その結果セットの TTL の有効期限が切れるまで、同じ (空の) 結果セットを呼び出します。

書き込みの方法

DAX の書き込みスルー動作は、多くのアプリケーションパターンに適しています。ただし、書き込みスルーモデルが適切ではないアプリケーションパターンもいくつかあります。

レイテンシーの影響を受けやすいアプリケーションの場合、DAX を介した書き込みでは余分なネットワークホップが発生します。そのため、DAX への書き込みは、DynamoDB への直接の書き込みよりも少し遅くなります。書き込みレイテンシーが重要なアプリケーションの場合は、代わりに DynamoDB に直接書き込むことで、レイテンシーを軽減できます。詳細については、「書き込み迂回」を参照してください。

書き込み負荷の高いアプリケーション (バルクデータのロードを実行するものなど) では、すべてのデータを DAX 経由で書き込むのは望ましくない場合があります。アプリケーションで読み込むのはそのデータのほんのわずかな割合であるためです。大量のデータを DAX 経由で書き込む場合、読み取られる新しい項目用にキャッシュ内に空きを作るため、LRU アルゴリズムが実行される必要があります。そのため、読み込みキャッシュとしての DAX の効果が低下します。

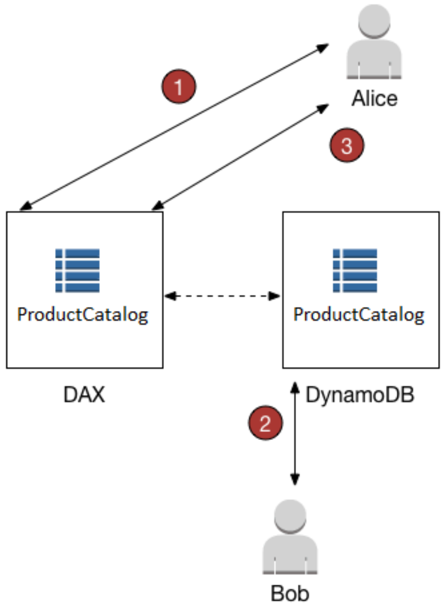
DAX に項目を書き込む場合、項目キャッシュの状態が新しい項目に対応するように変更されます。(たとえば、DAX は新しい項目用の空きを作るために、古いデータを項目キャッシュから削除する必要がある場合があります。) 新しい項目は、キャッシュの LRU アルゴリズムおよびキャッシュの TTL 設定によりますが、項目キャッシュに残ります。項目が項目キャッシュに保持される限り、DAX はその項目を DynamoDB から再ロードしません。

書き込みスルー

DAX 項目キャッシュには書き込みスルーポリシーが導入されています。詳細については、「DAX の書き込み処理方法」を参照してください。

項目を書き込むと、DAX はキャッシュされた項目が DynamoDB に存在する項目と同期されていることを確認します。これは、項目を書き込み直後に再ロードする必要があるアプリケーションにとって有用です。ただし、他のアプリケーションが DynamoDB テーブルに直接書き込んだ場合、DAX 項目キャッシュ内のその項目は、DynamoDB と同期しなくなります。

例として、`ProductCatalog` テーブルを使用している 2 人のユーザー (Alice と Bob) がいるとします。Alice は DAX を使用してテーブルにアクセスしますが、Bob は DAX をバイパスして、DynamoDB 内のテーブルに直接アクセスします。



1. Alice が `ProductCatalog` テーブル内の項目を更新します。DAX が DynamoDB にリクエストを転送し、更新が正常終了します。DAX は、項目を項目キャッシュに書き込み、Alice に成功応答を返します。この時点から、項目が最終的にキャッシュから削除されるまで、この項目を DAX から読み込んだユーザーには Alice が更新した項目が表示されます。

2. しばらくして、Bob は Alice が書き込んだのと同じ `ProductCatalog` 項目を更新します。ただし、ボブは項目を DynamoDB で直接更新します。DAX では、DynamoDB 経由の更新に応じて項目キャッシュが自動的に更新されません。そのため、DAX のユーザーからは、Bob の更新が見えません。

3. Alice が DAX からその項目を再度読み込みます。項目は項目キャッシュにあります。したがって、DAX は DynamoDB テーブルにアクセスせずに Alice に項目を返します。

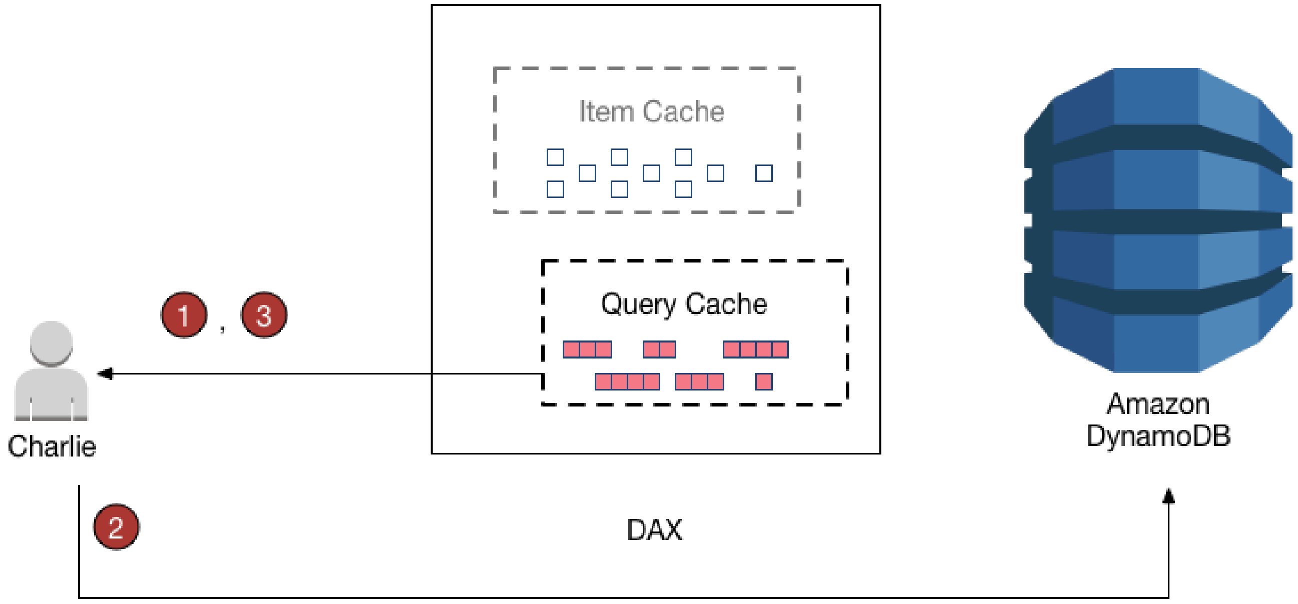
このシナリオでは、Alice と Bob では、同じ `ProductCatalog` 項目で表示が異なります。DAX がその項目を項目キャッシュから削除するか、別のユーザーが同じ項目を DAX で再度更新するまで、この状態になります。

書き込み迂回

アプリケーションで大量のデータ (バルクデータのロードなど) を書き込む必要がある場合、DAX をバイパスしてデータを直接 DynamoDB に書き込む方が妥当な場合があります。このような書き込み迂回の方法により、書き込みレイテンシーを軽減します。ただし、項目キャッシュは DynamoDB のデータと同期しません。

書き込み迂回戦略を使用する場合は、DAX で項目キャッシュに書き込まれるのは、アプリケーションが DAX クライアントを使用してデータを読み取るときであることに注意してください。これは一部のケースでは利点になります。最も頻繁に読み取られるデータのみが (最も頻繁に書き込まれるデータとは反対に) キャッシュされるためです。

たとえば、あるユーザー (Charlie) が、DAX を使用して別のテーブルである `GameScores` テーブルを使用するとしましょう。`GameScores` のパーティションキーは `UserId` であるため、Charlie のスコアは同じ `UserId` を持ちます。



1. Charlie は自分のスコアをすべて取得するため、`Query` を DAX に送信します。このクエリが以前に発行されていないとすると、DAX は処理のためにクエリを DynamoDB に転送します。結果を DAX クエリキャッシュに保存し、Charlie に結果セットを返します。この結果セットは、削除されるまでクエリキャッシュで使用できます。

2. ここで、Charlie が Meteor Blasters ゲームをプレイし、ハイスコアを達成したとします。Charlie は `UpdateItem` リクエストを DynamoDB に送信して、`GameScores` テーブルの項目を変更します。

3. 最後に、Charlie は先の `Query` を再実行して、自分のデータをすべて `GameScores` から取得することにします。この結果には、Meteor Blasters のハイスコアは表示されません。これは、クエリの結果が項目キャッシュではなくクエリキャッシュから取得されるためです。2 つのキャッシュは互いに独立しており、一方のキャッシュでの変更は他方のキャッシュに影響しません。

DAX では、クエリキャッシュの結果セットを DynamoDB の最新のデータに更新しません。クエリキャッシュ内の各結果セットは、`Query` または `Scan` オペレーションが実行された時点のものです。したがって、Charlie の `Query` の結果は、彼の `PutItem` オペレーションを反映していません。DAX が結果セットをクエリキャッシュから削除するまで、この状態が続きます。

DynamoDB Accelerator (DAX) クライアントで開発する

DAX をアプリケーションから使用するには、プログラミング言語に DAX クライアントを使用します。DAX クライアントは、既存の Amazon DynamoDB アプリケーションの中断を最小限に抑えるように設計されています。必要な作業は簡単なコード変更だけです。

> **Note**
>
> さまざまなプログラミング言語用の DAX クライアントは以下のサイトで入所できます。
>
> - http://dax-sdk.s3-website-us-west-2.amazonaws.com

このセクションでは、デフォルト Amazon VPC で Amazon EC2 インスタンスを起動し、インスタンスに接続してサンプルアプリケーションを実行する方法を示します。また、DAX クラスターを使用できるように既存のアプリケーションを変更する方法についての情報を提供します。

トピック

チュートリアル: DynamoDB Accelerator (DAX) を使用してサンプルアプリケーションを実行する

このチュートリアルでは、デフォルトの Virtual Private Cloud (VPC) で Amazon EC2 インスタンスを起動し、インスタンスに接続して、Amazon DynamoDB Accelerator (DAX) を使用するサンプルアプリケーションを実行する方法を示します。

> Note
>
> このチュートリアルを完了するには、デフォルト VPC で実行している DAX クラスターが必要です。まだ DAX クラスターを作成していない場合は、「DAX クラスターの作成」の手順を参照してください。

トピック

ステップ 1: Amazon EC2 インスタンスを起動する

Amazon DynamoDB Accelerator (DAX) クラスターが使用可能な場合、デフォルト Amazon VPC で Amazon EC2 インスタンスを起動できます。その後、そのインスタンスで DAX クライアントソフトウェアをインストールし、実行できます。

EC2 インスタンスを起動するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. [Launch Instance] (インスタンスを起動) を選択して、以下を実行します。

   ステップ 1: Amazon マシンイメージ (AMI) の選択

   1. AMI のリストで Amazon Linux AMI を見つけ、[Select] (選択) を選択します。

   ステップ 2: インスタンスタイプを選択する

1. インスタンスタイプのリストで、t2.micro を選択します。
2. [Next: Configure Instance Details] (次のステップ: インスタンスの詳細の設定) を選択します。

ステップ 3: インスタンスの詳細を設定する

1. [Network] (ネットワーク) で、デフォルトの VPC を選択します。
2. [Next: Add Storage] (次の手順: ストレージの追加) をクリックします。

ステップ 4: ストレージを追加する

1. [Next: Add Tags] (次の手順: タグの追加) を選択してこのステップをスキップします。

ステップ 5: タグの追加

1. [Next: Configure Security Group] (次のステップ: セキュリティグループの設定) を選択してこのステップをスキップします。

ステップ 6: セキュリティグループを設定する

1. [Select an existing security group] (既存のセキュリティグループの選択) を選択します。
2. セキュリティグループのリストで、[default (デフォルト)] を選択します。これは VPC のデフォルトのセキュリティグループです。
3. [Next: Review and Launch] (次のステップ: 確認と起動) を選択します。

ステップ 7: インスタンス起動の確認

1. [Launch] (起動する) を選択します。

3. [Select an existing key pair or create a new key pair] (既存のキーペアを選択するか、新しいキーペアを作成する) ウィンドウで、次のいずれかを実行します。

   - Amazon EC2 キーペアがない場合は、[Create a new key pair] (新しいキーペアの作成) を選択して画面に表示される指示に従います。プライベートキーファイル (`.pem`) をダウンロードするかどうか質問されます。このファイルは、後で Amazon EC2 インスタンスにログインするときに必要になります。

- 既存の Amazon EC2 キーペアがすでにある場合は、[Select a key pair] (キーペアの選択) を選択して、リストからキーペアを選択します。Amazon EC2 インスタンスにログインするには、すでにプライベートキーファイル (`.pem` ファイル) が利用可能になっている必要があります。

4. キーペアを設定したら、[Launch Instances] (インスタンスの起動) を選択します。
5. コンソールナビゲーションペインで [EC2 Dashboard] (EC2 ダッシュボード) を選択し、起動したインスタンスを選択します。下のペインの [Description] (説明) タブで、インスタンスの [Public DNS] (パブリック DNS) を見つけます。たとえば、`ec2-11-22-33-44.us-west-2.compute.amazonaws.com` です。このパブリック DNS 名をメモします。ステップ 3: Amazon EC2 インスタンスの設定 で必要になります。

> **Note**
>
> Amazon EC2 インスタンスが使用可能になるまでに数分かかります。その間に、「ステップ 2: ユーザーとポリシーを作成する」に進んでその指示に従います。

ステップ 2: ユーザーとポリシーを作成する

このステップでは、AWS Identity and Access Management を使用して Amazon DynamoDB Accelerator (DAX) クラスターおよび DynamoDB へのアクセス許可を付与するポリシーを持つユーザーを作成します。その後、DAX クラスターを使用するアプリケーションを実行できるようになります。

AWS アカウントへのサインアップ

AWS アカウントがない場合は、以下のステップを実行して作成します。

AWS アカウントにサインアップするには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   AWS アカウントにサインアップすると、AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべてのAWS のサービスとリソースへのアクセス権があり

ます。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

サインアップ処理が完了すると、AWS からユーザーに確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

AWS アカウント にサインアップしたら、AWS アカウントのルートユーザー をセキュリティで保護し、AWS IAM Identity Center を有効にして、管理ユーザーを作成します。これにより、日常的なタスクにルートユーザーを使用しないようにします。

AWS アカウントのルートユーザーをセキュリティで保護する

1. [ルートユーザー] を選択し、AWS アカウントのメールアドレスを入力して、アカウント所有者として AWS Management Console にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの「AWS アカウント のルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Center の有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   IAM アイデンティティセンターディレクトリ をアイデンティティソースとして使用するチュートリアルについては、「AWS IAM Identity Center ユーザーガイド」の「デフォルト IAM アイデンティティセンターディレクトリを使用したユーザーアクセスの設定」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「AWS アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

JSON ポリシーエディタでポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
5. JSON ポリシードキュメントを入力するか貼り付けます。IAM ポリシー言語の詳細については、「IAM JSON ポリシーリファレンス」を参照してください。
6. ポリシーの検証中に生成されたセキュリティ警告、エラー、または一般的な警告を解決してから、[Next] (次へ) を選択します。

   > Note
   >
   > いつでも [Visual] と [JSON] エディタオプションを切り替えることができます。ただし、[Visual] エディタで [次] に変更または選択した場合、IAM はポリシーを再構成して visual エディタに合わせて最適化することがあります。詳細については、『IAM ユーザーガイド』の「ポリシーの再構成」を参照してください。
7. （オプション）AWS Management Console でポリシーを作成または編集するときに、AWS CloudFormation テンプレートで使用できる JSON または YAML ポリシーテンプレートを生成できます。

   これを行うには、ポリシーエディタで [アクション] を選択し、次に [CloudFormation テンプレートを生成] を選択します。AWS CloudFormation の詳細については、『AWS CloudFormation ユーザーガイド』の「AWS Identity and Access Management リソースタイプリファレンス」を参照してください。
8. ポリシーにアクセス権限を追加し終えたら、[次へ] を選択します。
9. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。[このポリシーで定義されているアクセス許可] を確認して、ポリシーによって付与されたアクセス許可を確認します。

10. (オプション) タグをキー - 値のペアとしてアタッチして、メタデータをポリシーに追加します。IAM でのタグの使用に関する詳細については、『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。
11. [Create Policy (ポリシーを作成)] をクリックして、新しいポリシーを保存します。

[ポリシードキュメント] — 以下のドキュメントをコピーして貼り付け、JSON を作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

ステップ 3: Amazon EC2 インスタンスの設定

Amazon EC2 インスタンスが使用可能になったら、インスタンスにログインして使用準備を整えます。

> **Note**
>
> 次のステップは、Linux を実行するコンピュータから Amazon EC2 インスタンスに接続することを想定しています。その他の接続方法については、Amazon EC2 の Linux インスタンス用ユーザーガイドの「Linux インスタンスへの接続」を参照してください。

EC2 インスタンスを設定するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. 次の例のように、ssh コマンドを使用して Amazon EC2 インスタンスにログインします。

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   プライベートキーファイル (.pem ファイル) とインスタンスのパブリック DNS 名を指定する必要があります。(「ステップ 1: Amazon EC2 インスタンスを起動する」を参照してください。)

   ログイン ID は ec2-user です。パスワードは不要です。
3. EC2 インスタンスにログインしたら、次に示すように AWS 認証情報を設定します。AWS アクセスキー ID とシークレットキー (「ステップ 2: ユーザーとポリシーを作成する」で取得したもの) を入力し、デフォルトリージョン名に現在のリージョンを設定します。(次の例では、デフォルトリージョン名は us-west-2 です)。

   ```
   aws configure

   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]:
   ```

Amazon EC2 インスタンスを起動して設定した後、利用可能なサンプルアプリケーションの 1 つを使用して DAX の機能をテストできます。詳細については、「ステップ 4: サンプルアプリケーションの実行」を参照してください。

ステップ 4: サンプルアプリケーションの実行

Amazon DynamoDB Accelerator (DAX) の機能をテストしやすくするため、Amazon EC2 インスタンスで、使用可能なサンプルアプリケーションの 1 つを実行できます。

トピック

- DAX SDK for Go
- Java および DAX
- .NET および DAX
- Node.js および DAX
- Python および DAX

DAX SDK for Go

この手順に従って、Amazon EC2 インスタンスで Go 対応 Amazon DynamoDB Accelerator (DAX) SDK for Go サンプルアプリケーションを実行します。

DAX の SDK for Go サンプルを実行するには

1. Amazon EC2 インスタンスで SDK for Go をセットアップします。

   a. Go プログラミング言語をインストール (Golang) します。

   ```
   sudo yum install -y golang
   ```

   b. Golang が正しくインストールおよび実行されているかテストします。

   ```
   go version
   ```

   次のようなメッセージが表示されます。

   ```
   go version go1.15.5 linux/amd64
   ```

   残りの手順は、Go バージョン 1.13 のデフォルトとなったモジュールのサポートに基づいています。

2. サンプル Golang アプリケーションをインストールします。

   ```
   go get github.com/aws-samples/aws-dax-go-sample
   ```

3. 以下の Golang プログラムを実行します。最初のプログラムは、`TryDaxGoTable` という DynamoDB テーブルを作成します。2 番目のプログラムは、テーブルにデータを書き込みます。

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command put-item
```

4. 以下の Golang プログラムを実行します。

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command scan
```

タイミング情報を書き留めます。これは `GetItem`、`Query`、`Scan` テストに必要なミリ秒の数字です。

5. 前のステップで、DynamoDB エンドポイントに対してプログラムを実行しました。ここでプログラムを再度実行しますが、今度は `GetItem`、`Query`、`Scan` オペレーションが DAX クラスターによって処理されます。

DAX クラスターのエンドポイントを確認するには、次のいずれかを選択します。

- DynamoDB コンソールの使用 — DAX クラスターを選択します。次の例のように、クラスターエンドポイントがコンソールに表示されます。

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- AWS CLI の使用 — 次のコマンドを入力します。

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

次の例のように、クラスターエンドポイントが出力に表示されます。

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

ここでプログラムを再度実行しますが、今度はクラスターエンドポイントをコマンドラインパラメータとして指定します。

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

出力の残りの部分を見て、タイミング情報を書き留めます。`GetItem`、`Query` および `Scan` の経過時間は、DynamoDB を使用した場合より DAX を使用した方が大幅に低いはずです。

6. 次の Golang プログラムを実行して `TryDaxGoTable` を削除します。

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command delete-table
```

Java および DAX

DAX SDK for Java 2.x は AWS SDK for Java 2.x と互換性があります。Java 8 以降をベースに構築されており、非ブロック I/O がサポートされています。AWS SDK for Java 1.x での DAX の使用については、「DAX を AWS SDK for Java 1.x で使用する」を参照してください。

クライアントを Maven 依存関係として使用する

これらのステップに従って、アプリケーションで Java 用 DAX SDK のクライアントを依存関係として使用します。

1. Apache Maven をダウンロードし、インストールします。詳細については、「Apache Maven のダウンロード」および「Apache Maven のインストール」を参照してください。
2. Maven 依存関係クライアントをアプリケーションのプロジェクトオブジェクトモデル (POM) ファイルに追加します。この例では、*x.x.x* を クライアントの実際のバージョン番号に置き換えます。

   ```
   <!--Dependency:-->
   <dependencies>
       <dependency>
           <groupId>software.amazon.dax</groupId>
           <artifactId>amazon-dax-client</artifactId>
           <version>x.x.x</version>
       </dependency>
   </dependencies>
   ```

TryDax サンプルコード

ワークスペースを設定し、DAX SDK を依存関係として追加したら、TryDax.java をプロジェクトに追加します。

このコマンドを使用してコードを実行します。

```
java -cp classpath TryDax
```

次のような出力が表示されます。

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
...

Running GetItem and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1-100 and sort keys 1-10
```

```
  Total time: 4390.240 ms - Avg time: 4.390 ms
  Total time: 3097.089 ms - Avg time: 3.097 ms
  Total time: 3273.463 ms - Avg time: 3.273 ms
  Total time: 3353.739 ms - Avg time: 3.354 ms
  Total time: 3533.314 ms - Avg time: 3.533 ms
Query test - partition key 1-100 and sort keys between 2 and 9
  Total time: 475.868 ms - Avg time: 4.759 ms
  Total time: 423.333 ms - Avg time: 4.233 ms
  Total time: 460.271 ms - Avg time: 4.603 ms
  Total time: 397.859 ms - Avg time: 3.979 ms
  Total time: 466.644 ms - Avg time: 4.666 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

タイミング情報を書き留めます。これは、GetItem および Query テストに必要なミリ秒の数字です。このケースでは、DynamoDB エンドポイントに対してプログラムを実行しました。次に、プログラムを再度実行します。今回は DAX クラスターに対して実行します。

DAX クラスターのエンドポイントを確認するには、次のいずれかを選択します。

- DynamoDB コンソールで DAX クラスターを選択します。次の例のように、クラスターエンドポイントがコンソールに表示されます。

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- AWS CLI を使用して、次のコマンドを入力します。

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  次の例のように、クラスターエンドポイントのアドレス、ポート、URL が出力に表示されます。

  ```
  {
      "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
      "Port": 8111,
      "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
  }
  ```

ここでプログラムを再度実行しますが、今度はクラスターエンドポイント URL をコマンドラインパラメータとして指定します。

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

出力を見て、タイミング情報を書き留めます。GetItem および Query の経過時間は、DynamoDB を使用した場合より DAX を使用した方が大幅に低いはずです。

SDK のメトリクス

DAX SDK for Java 2.x を使用すると、アプリケーションのサービスクライアントに関するメトリクスを収集し、Amazon CloudWatch で出力を分析できます。詳細については、「メトリクスを有効にする」を参照してください。

> **Note**
>
> DAX SDK for Java は ApiCallSuccessful および ApiCallDuration メトリクスのみを収集します。

TryDax.java

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
    public static void main(String[] args) throws Exception {
        DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
                .build();
        DynamoDbAsyncClient daxClient = null;
        if (args.length >= 1) {
            daxClient = ClusterDaxAsyncClient.builder()
                    .overrideConfiguration(Configuration.builder()
                            .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
                            .build())
                    .build();
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        createTable(tableName, ddbClient);

        System.out.println("Populating table...");
        writeData(tableName, ddbClient, 100, 10);

        DynamoDbAsyncClient testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        getItemTest(tableName, testClient, 100, 10, 5);

        // Query
        queryTest(tableName, testClient, 100, 2, 9, 5);

        System.out.println("Deleting table...");
        deleteTable(tableName, ddbClient);
    }

    private static void createTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("Attempting to create table; please wait...");

            client.createTable(CreateTableRequest.builder()
                    .tableName(tableName)
                    .keySchema(KeySchemaElement.builder()
                            .keyType(KeyType.HASH)
                            .attributeName("pk")
                            .build(), KeySchemaElement.builder()
                            .keyType(KeyType.RANGE)
                            .attributeName("sk")
                            .build())
                    .attributeDefinitions(AttributeDefinition.builder()
                            .attributeName("pk")
                            .attributeType(ScalarAttributeType.N)
                            .build(), AttributeDefinition.builder()
                            .attributeName("sk")
                            .attributeType(ScalarAttributeType.N)
                            .build())
                    .billingMode(BillingMode.PAY_PER_REQUEST)
                    .build()).get();
            client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully created table.");

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            client.deleteTable(DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }
```

```
   private static void writeData(String tableName, DynamoDbAsyncClient client, int 
pkmax, int skmax) {
       System.out.println("Writing data to the table...");

       int stringSize = 1000;
       StringBuilder sb = new StringBuilder(stringSize);
       for (int i = 0; i < stringSize; i++) {
           sb.append('X');
       }
       String someData = sb.toString();

       try {
           for (int ipk = 1; ipk <= pkmax; ipk++) {
               System.out.println(("Writing " + skmax + " items for partition key: " + 
ipk));
               for (int isk = 1; isk <= skmax; isk++) {
                   client.putItem(PutItemRequest.builder()
                           .tableName(tableName)
                           .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData", 
attr(someData)))
                           .build()).get();
               }
           }
       } catch (Exception e) {
           System.err.println("Unable to write item:");
           e.printStackTrace();
       }
   }

   private static AttributeValue attr(int n) {
       return AttributeValue.builder().n(String.valueOf(n)).build();
   }

   private static AttributeValue attr(String s) {
       return AttributeValue.builder().s(s).build();
   }

   private static void getItemTest(String tableName, DynamoDbAsyncClient client, int 
pk, int sk, int iterations) {
       long startTime, endTime;
       System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-" 
+ sk);
```

```
       for (int i = 0; i < iterations; i++) {
           startTime = System.nanoTime();
           try {
               for (int ipk = 1; ipk <= pk; ipk++) {
                   for (int isk = 1; isk <= sk; isk++) {
                       client.getItem(GetItemRequest.builder()
                               .tableName(tableName)
                               .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
                               .build()).get();
                   }
               }
           } catch (Exception e) {
               System.err.println("Unable to get item:");
               e.printStackTrace();
           }
           endTime = System.nanoTime();
           printTime(startTime, endTime, pk * sk);
       }
   }

   private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
       long startTime, endTime;
       System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

       for (int i = 0; i < iterations; i++) {
           startTime = System.nanoTime();
           for (int ipk = 1; ipk <= pk; ipk++) {
               try {
                   // Pagination API for Query.
                   client.queryPaginator(QueryRequest.builder()
                           .tableName(tableName)
                           .keyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                           .expressionAttributeValues(Map.of(":pkval", attr(ipk),
":skval1", attr(sk1), ":skval2", attr(sk2)))
                           .build()).items().subscribe((item) -> {
                   }).get();
               } catch (Exception e) {
                   System.err.println("Unable to query table:");
                   e.printStackTrace();
               }
           }
```

```
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk);
        }
    }

    private static void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

.NET および DAX

次のステップに従って、Amazon EC2 インスタンスで .NET サンプルを実行します。

> Note
>
> このチュートリアルでは .NET 6 SDK を使用していますが、.NET Core SDK でも動作します。ここでは、デフォルト Amazon VPC のプログラムを実行して、Amazon DynamoDB Accelerator (DAX) クラスターにアクセスする方法を示しています。必要に応じて、AWS Toolkit for Visual Studio を使用して .NET アプリケーションを記述し、VPC 内にデプロイできます。
> 詳細については、「AWS Elastic Beanstalk デベロッパーガイド」の「AWS Toolkit for Visual Studio を使用した .NET での Elastic Beanstalk アプリケーションの作成とデプロイ」を参照してください。

DAX の .NET サンプルを実行するには

1. Microsoft ダウンロードページにアクセスし、最新の .NET 6 (または .NET Core) SDK for Linux をダウンロードします。ダウンロードされたファイルは、`dotnet-sdk-N.N.N-linux-x64.tar.gz` です。
2. SDK ファイルを展開します。

   ```
   mkdir dotnet
   tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
   ```

   `N.N.N` を .NET SDK の実際のバージョン番号 (例: `6.0.100`) で置き換えます。

3. インストールを確認します。

```
alias dotnet=$HOME/dotnet/dotnet
dotnet --version
```

これにより、.NET SDK のバージョン番号が表示されます。

> **Note**
>
> バージョン番号ではなく、以下のエラーが表示される場合があります。
>
> error: libunwind.so.8: 共有オブジェクトファイルを開けません: そのようなファイルまたはディレクトリは存在しません
>
> このエラーを解決するには、`libunwind` パッケージをインストールします。
>
> ```
> sudo yum install -y libunwind
> ```
>
> この操作を行うと、`dotnet --version` コマンドがエラーなしで実行されます。

4. 新しい .NET プロジェクトを作成します。

```
dotnet new console -o myApp
```

この 1 回限りの設定には数分かかります。設定が完了したら、サンプルプロジェクトを実行します。

```
dotnet run --project myApp
```

`Hello World!` というメッセージが表示されます。

5. `myApp/myApp.csproj` ファイルにはプロジェクトに関するメタデータが含まれています。アプリケーションで DAX クライアントを使用するには、このファイルを次のように変更します。

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net6.0</TargetFramework>
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
```

```
    </ItemGroup>
</Project>
```

6. サンプルプログラムソースコード (`.zip` ファイル) をダウンロードします。

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   ダウンロードが完了したら、ソースファイルを解凍します。

   ```
   unzip TryDax.zip
   ```

7. 次に、サンプルプログラムを一度に 1 つずつ実行します。プログラムごとにそのコンテンツを `myApp/Program.cs` 内にコピーし、`MyApp` プロジェクトを実行します。

   以下の .NET プログラムを実行します。最初のプログラムは、`TryDaxTable` という DynamoDB テーブルを作成します。2 番目のプログラムは、テーブルにデータを書き込みます。

   ```
   cp TryDax/dotNet/01-CreateTable.cs myApp/Program.cs
   dotnet run --project myApp

   cp TryDax/dotNet/02-Write-Data.cs myApp/Program.cs
   dotnet run --project myApp
   ```

8. 次に、いくつかのプログラムを実行し、`GetItem`、`Query`、および `Scan` の各オペレーションを DAX クラスターに対して実行します。DAX クラスターのエンドポイントを確認するには、次のいずれかを選択します。

   - DynamoDB コンソールの使用 — DAX クラスターを選択します。次の例のように、クラスターエンドポイントがコンソールに表示されます。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI の使用 — 次のコマンドを入力します。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     次の例のように、クラスターエンドポイントが出力に表示されます。

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

次に、以下のプログラムを実行し、クラスターエンドポイントをコマンドラインパラメータとして指定します (サンプルのエンドポイントを実際の DAX クラスターエンドポイントに置き換えてください)。

```
cp TryDax/dotNet/03-GetItem-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/04-Query-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/05-Scan-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
```

タイミング情報を書き留めます。これは GetItem、Query、Scan テストに必要なミリ秒の数字です。

9. 次の .NET プログラムを実行し、TryDaxTable を削除します。

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

これらのプログラムの詳細については、以下のセクションを参照してください。

- 01-CreateTable.cs
- 02-Write-Data.cs
- 03-GetItem-Test.cs
- 04-Query-Test.cs
- 05-Scan-Test.cs

- 06-DeleteTable.cs

01-CreateTable.cs

01-CreateTable.cs プログラムはテーブル (TryDaxTable) を作成します。このセクションの残りの .NET プログラムは、このテーブルに依存します。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new CreateTableRequest()
            {
                TableName = tableName,
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement{ AttributeName = "pk",KeyType = "HASH"},
                    new KeySchemaElement{ AttributeName = "sk",KeyType = "RANGE"}
                },
                AttributeDefinitions = new List<AttributeDefinition>() {
                    new AttributeDefinition{ AttributeName = "pk",AttributeType = "N"},
                    new AttributeDefinition{ AttributeName = "sk",AttributeType  = "N"}
                },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 10
                }
            };
            var response = await client.CreateTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

02-Write-Data.cs

`02-Write-Data.cs` プログラムはテストデータを `TryDaxTable` に書き込みます。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            string someData = new string('X', 1000);
            var pkmax = 10;
            var skmax = 10;

            for (var ipk = 1; ipk <= pkmax; ipk++)
            {
                Console.WriteLine($"Writing {skmax} items for partition key: {ipk}");
                for (var isk = 1; isk <= skmax; isk++)
                {
                    var request = new PutItemRequest()
                    {
                        TableName = tableName,
                        Item = new Dictionary<string, AttributeValue>()
                       {
                            { "pk", new AttributeValue{N = ipk.ToString() } },
                            { "sk", new AttributeValue{N = isk.ToString() } },
                            { "someData", new AttributeValue{S = someData } }
                        }
                    };

                    var response = await client.PutItemAsync(request);
                }
            }

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

03-GetItem-Test.cs

`03-GetItem-Test.cs` プログラムは、`GetItem` で `TryDaxTable` オペレーションを実行します。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);
            var tableName = "TryDaxTable";

            var pk = 1;
            var sk = 10;
            var iterations = 5;

            var startTime = System.DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                for (var ipk = 1; ipk <= pk; ipk++)
                {
                    for (var isk = 1; isk <= sk; isk++)
                    {
                        var request = new GetItemRequest()
                        {
                            TableName = tableName,
                            Key = new Dictionary<string, AttributeValue>() {
                            {"pk", new AttributeValue {N = ipk.ToString()} },
                            {"sk", new AttributeValue {N = isk.ToString() } }
                        }
                        };
                        var response = await client.GetItemAsync(request);
                        Console.WriteLine($"GetItem succeeded for pk: {ipk},sk:
 {isk}");
                    }
                }
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

04-Query-Test.cs

`04-Query-Test.cs` プログラムは、`Query` で `TryDaxTable` オペレーションを実行します。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");


            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 5;
            var sk1 = 2;
            var sk2 = 9;
            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new QueryRequest()
                {
                    TableName = tableName,
                    KeyConditionExpression = "pk = :pkval and sk between :skval1
 and :skval2",
                    ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>() {
                            {":pkval", new AttributeValue {N = pk.ToString()} },
                            {":skval1", new AttributeValue {N = sk1.ToString()} },
                            {":skval2", new AttributeValue {N = sk2.ToString()} }
                    }
                };
                var response = await client.QueryAsync(request);
                Console.WriteLine($"{i}: Query succeeded");

            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

05-Scan-Test.cs

`05-Scan-Test.cs` プログラムは、`Scan` で `TryDaxTable` オペレーションを実行します。

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new ScanRequest()
                {
                    TableName = tableName
                };
                var response = await client.ScanAsync(request);
                Console.WriteLine($"{i}: Scan succeeded");
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

06-DeleteTable.cs

`06-DeleteTable.cs` プログラムは `TryDaxTable` を削除します。テストが完了してから、このプログラムを実行します。

```
using System;
using System.Threading.Tasks;
using Amazon.DynamoDBv2.Model;
using Amazon.DynamoDBv2;

namespace ClientTest
```

```
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new DeleteTableRequest()
            {
                TableName = tableName
            };

            var response = await client.DeleteTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

Node.js および DAX

このステップに従って、Amazon EC2 インスタンスで Node.js サンプルアプリケーションを実行します。

DAX の Node.js サンプルを実行するには

1. 以下のように、Amazon EC2 インスタンスで Node.js を設定します。

    a. ノードバージョンマネージャー (nvm) をインストールしてください。

    ```
    curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
    ```

    b. nvm を使用して Node.js をインストールします。

    ```
    nvm install 12.16.3
    ```

    c. Node.js が正しくインストールおよび実行されているかテストします。

```
node -e "console.log('Running Node.js ' + process.version)"
```

これにより次のメッセージが表示されるはずです。

```
Running Node.js v12.16.3
```

2. ノードパッケージマネージャー (npm) を使用して、DAX Node.js クライアントをインストールします。

   ```
   npm install amazon-dax-client
   ```

3. サンプルプログラムソースコード (`.zip` ファイル) をダウンロードします。

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   ダウンロードが完了したら、ソースファイルを解凍します。

   ```
   unzip TryDax.zip
   ```

4. 次の Node.js プログラムを実行します。最初のプログラムは、`TryDaxTable` という Amazon DynamoDB テーブルを作成します。2 番目のプログラムは、テーブルにデータを書き込みます。

   ```
   node 01-create-table.js
   node 02-write-data.js
   ```

5. 次の Node.js プログラムを実行します。

   ```
   node 03-getitem-test.js
   node 04-query-test.js
   node 05-scan-test.js
   ```

   タイミング情報を書き留めます。これは `GetItem`、`Query`、`Scan` テストに必要なミリ秒の数字です。

6. 前のステップで、DynamoDB エンドポイントに対してプログラムを実行しました。プログラムを再度実行しますが、今度は `GetItem`、`Query`、`Scan` オペレーションが DAX クラスターによって処理されます。

DAX クラスターのエンドポイントを確認するには、次のいずれかを選択します。

- [Using the DynamoDB console] (DynamoDB コンソールの使用) — DAX クラスターを選択します。次の例のように、クラスターエンドポイントがコンソールに表示されます。

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- AWS CLI の使用 - 次のコマンドを入力します。

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  次の例のように、クラスターエンドポイントが出力に表示されます。

  ```
  {
      "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
      "Port": 8111,
      "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
  }
  ```

ここでプログラムを再度実行しますが、今度はクラスターエンドポイントをコマンドラインパラメータとして指定します。

```
node 03-getitem-test.js dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
node 04-query-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
node 05-scan-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

出力の残りの部分を見て、タイミング情報を書き留めます。`GetItem`、`Query` および `Scan` の経過時間は、DynamoDB を使用した場合より DAX を使用した方が大幅に低いはずです。

7. 次の Node.js プログラムを実行して `TryDaxTable` を削除します。

```
node 06-delete-table
```

これらのプログラムの詳細については、以下のセクションを参照してください。

- 01-create-table.js

- 02-write-data.js
- 03-getitem-test.js
- 04-query-test.js
- 05-scan-test.js
- 06-delete-table.js

01-create-table.js

01-create-table.js プログラムはテーブル (TryDaxTable) を作成します。このセクションの残りの Node.js プログラムは、このテーブルによって異なります。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
  KeySchema: [
    { AttributeName: "pk", KeyType: "HASH" }, //Partition key
    { AttributeName: "sk", KeyType: "RANGE" }, //Sort key
  ],
  AttributeDefinitions: [
    { AttributeName: "pk", AttributeType: "N" },
    { AttributeName: "sk", AttributeType: "N" },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 10,
    WriteCapacityUnits: 10,
  },
};

dynamodb.createTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to create table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Created table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

02-write-data.js

`02-write-data.js` プログラムはテストデータを `TryDaxTable` に書き込みます。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();

var tableName = "TryDaxTable";

var someData = "X".repeat(1000);
var pkmax = 10;
var skmax = 10;

for (var ipk = 1; ipk <= pkmax; ipk++) {
  for (var isk = 1; isk <= skmax; isk++) {
    var params = {
      TableName: tableName,
      Item: {
        pk: ipk,
        sk: isk,
        someData: someData,
      },
    };

    //
    //put item

    ddbClient.put(params, function (err, data) {
      if (err) {
        console.error("Unable to write data: ", JSON.stringify(err, null, 2));
      } else {
        console.log("PutItem succeeded");
      }
    });
  }
}
```

03-getitem-test.js

`03-getitem-test.js` プログラムは、`GetItem` で `TryDaxTable` オペレーションを実行します。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 1;
var sk = 10;
var iterations = 5;

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  for (var ipk = 1; ipk <= pk; ipk++) {
    for (var isk = 1; isk <= sk; isk++) {
      var params = {
        TableName: tableName,
        Key: {
          pk: ipk,
          sk: isk,
        },
      };

      client.get(params, function (err, data) {
        if (err) {
          console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
        } else {
          // GetItem succeeded
        }
      });
    }
  }

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

04-query-test.js

`04-query-test.js` プログラムは、`Query` で `TryDaxTable` オペレーションを実行します。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;

var params = {
  TableName: tableName,
  KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
  ExpressionAttributeValues: {
    ":pkval": pk,
    ":skval1": sk1,
    ":skval2": sk2,
  },
};

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  client.query(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Query succeeded
    }
  });

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

05-scan-test.js

`05-scan-test.js` プログラムは、Scan で `TryDaxTable` オペレーションを実行します。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}
```

```
var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var iterations = 5;

var params = {
  TableName: tableName,
};
var startTime = new Date().getTime();
for (var i = 0; i < iterations; i++) {
  client.scan(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Scan succeeded
    }
  });
}

var endTime = new Date().getTime();
console.log(
  "\tTotal time: ",
  endTime - startTime,
  "ms - Avg time: ",
  (endTime - startTime) / iterations,
  "ms"
);
```

06-delete-table.js

`06-delete-table.js` プログラムは `TryDaxTable` を削除します。テストが完了してから、このプログラムを実行します。

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
};

dynamodb.deleteTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to delete table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Deleted table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

Python および DAX

この手順に従って、Amazon EC2 インスタンスで Python サンプルアプリケーションを実行します。

DAX の Python サンプルを実行するには

1. `pip` ユーティリティを使用して DAX Python クライアントをインストールします。

   ```
   pip install amazon-dax-client
   ```

2. サンプルプログラムソースコード (`.zip` ファイル) をダウンロードします。

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   ダウンロードが完了したら、ソースファイルを解凍します。

```
unzip TryDax.zip
```

3. 以下の Python プログラムを実行します。最初のプログラムは、`TryDaxTable` という Amazon DynamoDB テーブルを作成します。2 番目のプログラムは、テーブルにデータを書き込みます。

   ```
   python 01-create-table.py
   python 02-write-data.py
   ```

4. 以下の Python プログラムを実行します。

   ```
   python 03-getitem-test.py
   python 04-query-test.py
   python 05-scan-test.py
   ```

   タイミング情報を書き留めます。これは `GetItem`、`Query`、`Scan` テストに必要なミリ秒の数字です。

5. 前のステップで、DynamoDB エンドポイントに対してプログラムを実行しました。ここでプログラムを再度実行しますが、今度は `GetItem`、`Query`、`Scan` オペレーションが DAX クラスターによって処理されます。

   DAX クラスターのエンドポイントを確認するには、次のいずれかを選択します。

   - DynamoDB コンソールの使用 — DAX クラスターを選択します。次の例のように、クラスターエンドポイントがコンソールに表示されます。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI の使用 — 次のコマンドを入力します。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     この例のように、クラスターエンドポイントが出力に表示されます。

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     ```

```
}
```

プログラムを再度実行しますが、今度はクラスターエンドポイントをコマンドラインパラメータとして指定します。

```
python 03-getitem-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 04-query-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 05-scan-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

出力の残りの部分を見て、タイミング情報を書き留めます。GetItem、Query および Scan の経過時間は、DynamoDB を使用した場合より DAX を使用した方が大幅に低いはずです。

6. 次の Python プログラムを実行して TryDaxTable を削除します。

```
python 06-delete-table.py
```

これらのプログラムの詳細については、以下のセクションを参照してください。

- 01-create-table.py
- 02-write-data.py
- 03-getitem-test.py
- 04-query-test.py
- 05-scan-test.py
- 06-delete-table.py

01-create-table.py

01-create-table.py プログラムはテーブル (TryDaxTable) を作成します。このセクションの残りの Python プログラムは、このテーブルに依存します。

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "ProvisionedThroughput": {"ReadCapacityUnits": 10, "WriteCapacityUnits": 10},
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

02-write-data.py

`02-write-data.py` プログラムはテストデータを `TryDaxTable` に書き込みます。

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

03-getitem-test.py

03-getitem-test.py プログラムは、GetItem で TryDaxTable オペレーションを実行します。この例は、リージョン eu-west-1 の場合に使用できます。

```
import argparse
import sys
import time
import amazondax
import boto3
```

```
def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource('dynamodb')

    table = dyn_resource.Table('TryDaxTable')
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(Key={
                    'partition_key': partition_key,
                    'sort_key': sort_key
                })
                print('.', end='')
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        'endpoint_url', nargs='?',
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.")
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url,
 region_name='eu-west-1') as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax)
    else:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the Boto3 client.")
        test_start, test_end = get_item_test(
            test_key_count, test_iterations)
    print(f"Total time: {test_end - test_start:.4f} sec. Average time: "
          f"{(test_end - test_start)/ test_iterations}.")
```

04-query-test.py

04-query-test.py プログラムは、Query で TryDaxTable オペレーションを実行します。

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this value.
    :param sort_keys: The range of sort key values for the query. The query returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations, dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

05-scan-test.py

`05-scan-test.py` プログラムは、`Scan` で `TryDaxTable` オペレーションを実行します。

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()
    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

06-delete-table.py

`06-delete-table.py` プログラムは `TryDaxTable` を削除します。Amazon DynamoDB Accelerator (DAX) 機能のテストが完了してから、このプログラムを実行します。

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

既存のアプリケーションを DAX を使用するように変更する

Amazon DynamoDB を使用している Java アプリケーションが既にある場合、DynamoDB Accelerator (DAX) クラスターにアクセスできるように変更できます。DAX Java クライアントは、AWS SDK for Java 2.x に含まれる DynamoDB 低レベルクライアントとよく似ているため、アプリケーション全体を書き換える必要はありません。詳細については、「DynamoDB での項目の操作」を参照してください。

> Note
>
> この例では AWS SDK for Java 2.x を使用します。レガシー SDK for Java 1.x バージョンの場合は、「既存の SDK for Java 1.x アプリケーションを DAX を使用するように変更する」を参照してください。

プログラムを変更するには、DynamoDB クライアントを DAX クライアントに置き換えます。

```
Region region = Region.US_EAST_1;

// Create an asynchronous DynamoDB client
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
                .region(region)
                .build();

// Create an asynchronous DAX client
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
                .overrideConfiguration(Configuration.builder()
                    .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
                    .region(region)
                    .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
 metric collection
                    .build())
                .build();
```

また、AWS SDK for Java 2.x の一部である高レベルのライブラリも使用できます。DynamoDB クライアントを DAX クライアントに置き換えてください。

```
Region region = Region.US_EAST_1;
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
```

```
        .overrideConfiguration(Configuration.builder()
            .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
            .region(region)
            .build())
        .build();

DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
        .dynamoDbClient(dax)
        .build();
```

詳細については、「DynamoDB テーブル内の項目のマッピング」を参照してください。

DAX クラスターの管理

このセクションでは、Amazon DynamoDB Accelerator (DAX) クラスターの一般的な管理タスクの一部を紹介します。

トピック

DAX クラスターを管理するための IAM アクセス許可

AWS Management Console または AWS Command Line Interface (AWS CLI)を使用して DAX クラスターを管理する場合は、ユーザーが実行できるアクションの範囲を絞り込むことを強くお勧めします。そうすることで、最小権限の原則に従って、リスクを緩和できます。

次の説明は、DAX 管理 API のアクセスコントロールに焦点を当てます。詳細については、「Amazon DynamoDB API リファレンス」の「Amazon DynamoDB Accelerator」を参照してください。

Note

AWS Identity and Access Management (IAM) アクセス許可を管理する方法の詳細については、以下を参照してください。

- IAM と DAX クラスターの作成: DAX クラスターの作成。
- IAM と DAX データプレーンオペレーション: DAX のアクセスコントロール。

DAX 管理 API では、特定のリソースへの API アクションを範囲に含めることはできません。Resource 要素は "*" に設定する必要があります。これは GetItem、Query、Scan などの DAX データプレーン API オペレーションとは異なります。データプレーンオペレーションは DAX のクライアントを通じて表示され、これらのオペレーションは特定のリソースを範囲にできます。

例として、次の IAM ポリシードキュメントについて考えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

このポリシーの目的は、特定のクラスター DAXCluster01 のみの DAX 管理 API コールを許可することであるとします。

ユーザーが次の AWS CLI コマンドを発行するとします。

```
aws dax describe-clusters
```

基盤となる DescribeClusters API コールが特定のクラスターを範囲にできないため、このコマンドは Not Authorized (権限がありません) という例外で失敗します。ポリシーが構文としては有効

であっても、このコマンドは Resource 要素が "*" に設定される必要があるため、失敗します。ただし、ユーザーが DAX データプレーンコール (GetItem や Query など) を DAXCluster01 に送信するプログラムを実行する場合は、それらのコールは成功します。これは、DAX データプレーン API が特定のリソース (この場合は DAXCluster01) を範囲にできるためです。

DAX 管理 API と DAX データプレーン API の両方を包括する単一の包括 IAM ポリシーを作成する場合は、ポリシードキュメントに 2 つの異なるステートメントを含めることをお勧めします。これらのステートメントの 1 つが DAX データプレーン API に対応し、もう 1 つのステートメントが DAX 管理 API に対応するようにします。

次のポリシー例で、この方法を示します。DAXDataAPIs ステートメントは DAXCluster01 リソースを範囲とするが、DAXManagementAPIs のリソースは "*" にする必要がある点に注意してください。各ステートメントに示されているアクションは、説明のみを目的としています。アプリの必要に応じて、これらをカスタマイズできます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXDataAPIs",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]},
            {
            "Sid": "DAXManagementAPIs",
            "Action": [
                "dax:CreateParameterGroup",
                "dax:CreateSubnetGroup",
                "dax:DecreaseReplicationFactor",
                "dax:DeleteCluster",
                "dax:DeleteParameterGroup",
                "dax:DeleteSubnetGroup",
                "dax:DescribeClusters",
                "dax:DescribeDefaultParameters",
                "dax:DescribeEvents",
                "dax:DescribeParameterGroups",
                "dax:DescribeParameters",
                "dax:DescribeSubnetGroups",
                "dax:IncreaseReplicationFactor",
                "dax:ListTags",
                "dax:RebootNode",
                "dax:TagResource",
                "dax:UntagResource",
                "dax:UpdateCluster",
                "dax:UpdateParameterGroup",
                "dax:UpdateSubnetGroup"
                ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

DAX クラスターのスケーリング

DAX クラスターのスケーリングには 2 つのオプションを使用できます。第 1 のオプションは水平スケーリングです。クラスターにリードレプリカを追加します。2 番目のオプションは、垂直スケーリング で、さまざまなノードタイプを選択します。アプリケーションに適切なクラスターサイズとノードタイプの選択方法の詳細については、「DAX クラスターサイジングガイド」を参照してください。

水平スケーリング

水平スケーリングでは、クラスターにリードレプリカを追加することで、読み込みオペレーションのスループットを向上させることができます。1 つの DAX クラスターで最大 10 個のリードレプリカをサポートし、レプリカはクラスターの実行中に追加または削除できます。

次の AWS CLI の例で、ノードの数を増減する方法を説明します。`--new-replication-factor` 引数は、クラスター内のノードの合計数を指定します。ノードの 1 つはプライマリノードであり、他のノードはリードレプリカです。

```
aws dax increase-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 5
```

```
aws dax decrease-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 3
```

> ⓘ Note
>
> レプリケーション係数を変更すると、クラスターの状態が modifying に変わります。変更が完了すると、ステータスが available に変わります。

垂直スケーリング

大規模なデータセットがある場合、より大きいノードタイプを使用することでアプリケーションに有益な場合があります。ノードが大きいほど、クラスターがより多くのデータをメモリに保存できるため、キャッシュミスが軽減され、アプリケーション全体のパフォーマンスが向上します。(DAX クラスターのすべてのノードは同じタイプでなければならないことに注意してください。)

DAX クラスターの書き込みオペレーションやキャッシュミスの割合が高い場合は、より大きなノードタイプを使用するとアプリケーションにも利点があります。書き込みオペレーションとキャッシュミスは、クラスターのプライマリノード上のリソースを消費します。したがって、より大きなノードタイプを使用すると、プライマリノードのパフォーマンスが向上し、これらのタイプのオペレーションのスループットが向上します。

実行中の DAX クラスターのノードタイプを変更することはできません。代わりに、目的のノードタイプの新しいクラスターを作成する必要があります。サポートされているノードタイプについては、「ノード」を参照してください。

新しい DAX クラスターを作成するには、AWS Management Console、AWS CloudFormation、AWS CLI、または AWS SDK を使用します。(AWS CLI の場合、--node-type パラメータを使用してノードタイプを指定します。)

DAX クラスターの設定のカスタマイズ

DAX クラスターを作成する場合、次のデフォルト設定が使用されます。

- 5 分の有効期限 (TTL) で有効化されたキャッシュの自動削除
- アベイラビリティーゾーンの指定なし
- メンテナンスウィンドウの指定なし
- 通知の無効化

新しいクラスターの場合は、作成時に設定をカスタマイズできます。AWS Management Consoleでこれを行うには、[デフォルト設定の使用] をオフにして以下の設定を変更します。

- [Network and Security] (ネットワークとセキュリティ) - 現在の AWS リージョン内の異なるアベイラビリティーゾーンで個別の DAX クラスターノードを実行できます。[No Preference] (指定なし) を選択すると、ノードが自動的にアベイラビリティーゾーン内に分散されます。
- [Parameter Group] (パラメータグループ) - クラスター内の各ノードに適用される指定のパラメータのセット。パラメータグループを使用してキャッシュ TTL の動作を指定できます。パラメータグループ (デフォルトのパラメータグループ `default.dax.1.0` を除く) 内の任意のパラメータの値をいつでも変更できます。
- [Maintenance Window] (メンテナンスウィンドウ) - ソフトウェアのアップグレードやパッチをクラスター内のノードに適用する間隔 (週)。メンテナンスウィンドウの開始日、開始時刻、および所要時間を選択できます。[No Preference] (指定なし) を選択すると、メンテナンスウィンドウはリージョンごとに 8 時間の時間ブロックからランダムに選択されます。詳細については、「メンテナンスウィンドウ」を参照してください。

> Note
>
> [Parameter Group] (パラメータグループ) と [Maintenance Window] (メンテナンスウィンドウ) も、実行中のクラスター上でいつでも変更できます。

メンテナンスイベントが発生すると、DAX から Amazon Simple Notification Service (Amazon SNS) を使用して通知が送信されます。通知を設定するには、[SNS 通知のトピック] セレクタからオプションを選択します。新しい Amazon SNS トピックを作成することも、既存のトピックを使用することもできます。

Amazon SNS トピックの設定とサブスクリプションの詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS の開始方法」を参照してください。

TTL 設定の構成

DAX は、DynamoDB から読み込んだデータを 2 つのキャッシュに保持します。

- 項目キャッシュ - `GetItem` または `BatchGetItem` を使用して取得した項目用。
- クエリキャッシュ - `Query` または `Scan` を使用して取得した結果セット用。

詳細については、「項目キャッシュ」および「クエリキャッシュ」を参照してください。

これらのキャッシュそれぞれのデフォルトの TTL は 5 分です。異なる TTL 設定を使用する場合は、カスタムパラメータグループを使用した DAX クラスターを起動できます。コンソールでこれを行うには、ナビゲーションペインの [DAX | Parameter groups (DAX | パラメータグループ)] を選択します。

上記のタスクを AWS CLI を使用して実行することもできます。次の例では、カスタムパラメータグループを使用して新しい DAX クラスターを起動する方法を示しています。この例では、項目キャッシュ TTL は 10 分に設定され、クエリキャッシュ TTL は 3 分に設定されます。

1. 新しいパラメータグループを作成します。

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. 項目キャッシュ TTL を 10 分 (600000 ミリ秒) に設定します。

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. クエリキャッシュ TTL を 3 分 (180000 ミリ秒) に設定します。

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. パラメータが正しく設定されたことを確認します。

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

このパラメータグループを使用して、新しい DAX クラスターを起動できます。

```
aws dax create-cluster \
    --cluster-name MyNewCluster \
    --node-type dax.r3.large \
    --replication-factor 3 \
    --iam-role-arn arn:aws:iam::123456789012:role/DAXServiceRole \
    --parameter-group custom-ttl
```

> **Note**
>
> 実行中の DAX インスタンスで使用されているパラメータグループを変更することはできません。

DAX のタグ付けサポート

DynamoDB を始めとする多くの AWS サービスは、タグ付け (ユーザー定義の名前でリソースにラベル付けする機能) をサポートします。DAX クラスターにタグ付けを割り当てて、同じタグ付けを持つすべての AWS リソースをすばやく識別したり、タグ付け別に AWS の請求書を分類したりできます。

詳細については、「」を参照してくださいリソースへのタグとラベルの追加

AWS Management Console の使用

DAX クラスタータグを管理するには

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ナビゲーションペインの、[DAX] で、[クラスター]を選択します。
3. 作業するクラスターを選択します。
4. [タグ] タブを選択します。ここでタグを追加、リスト、編集、または削除できます。

   設定が希望どおりになったら、[変更の適用] を選択します。

AWS CLI の使用

AWS CLI を使用して DAX クラスタータグを管理している場合は、最初にクラスターの Amazon リソースネーム (ARN) を確認する必要があります。次の例は、MyDAXCluster という名前のクラスターの ARN を確認する方法を示しています。

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

出力で、ARN は次の例のようになります。arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster

次の例は、クラスターにタグ付けする方法を示しています。

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

クラスターのすべてのタグを一覧表示します。

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

タグを削除するには、キーを指定します。

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

AWS CloudTrail の統合

DAX は AWS CloudTrail と統合されているため、DAX クラスターのアクティビティを監査できます。CloudTrail ログを使用して、クラスターレベルで行われたすべての変更を表示できます。ノード、サブネットグループ、パラメータグループなどのクラスターコンポーネントへの変更も確認できます。詳細については、「」を参照してくださいAWS CloudTrail を使用して DynamoDB オペレーションをログに記録する

クラスターを削除する

使用していないリソースに課金されないように、不要になった DAX クラスターを削除する必要があります。

コンソールまたは AWS CLI を使用して、DAX クラスターを削除できます。以下はその例です。

```
aws dax delete-cluster --cluster-name mydaxcluster
```

DAX をモニタリングする

モニタリングは、Amazon DynamoDB Accelerator (DAX) および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。

ただし、DAX のモニタリングを開始する前に、以下の質問に対する回答を反映したモニタリング計画を作成する必要があります。

- どのような目的でモニタリングしますか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

トピック

モニタリングツール

AWS では、Amazon DynamoDB Accelerator (DAX) のモニタリングに使用できるツールを提供しています。これらの中には、自動モニタリングを設定できるものもあれば、手動操作を必要とするものもあります。モニタリングタスクをできるだけ自動化することをお勧めします。

トピック

- 自動モニタリングツール
- 手動モニタリングツール

自動モニタリングツール

以下の自動化されたモニタリングツールを使用して DAX を監視し、問題が発生したときにレポートできます。

- Amazon CloudWatch アラーム - 指定した期間にわたって単一のメトリクスをモニタリングし、複数の期間にわたる特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すことはありません。状態が変更され、指定された期間維持されている必要があります。詳細については、「Amazon CloudWatch によるメトリクスのモニタリング」を参照してください。
- Amazon CloudWatch Logs – AWS CloudTrail またはその他のソースのログファイルのモニタリング、保存、アクセスを行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「ログファイルのモニタリング」を参照してください。
- Amazon CloudWatch Events - イベントに一致したものを 1 つ以上のターゲットの関数またはストリームに渡して、変更、状態の情報の収集、是正措置を行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Events とは」を参照してください。
- AWS CloudTrail のログのモニタリング – アカウント間でログファイルを共有し、CloudTrail のログファイルを CloudWatch Logs に送信してリアルタイムでモニタリングします。また、ログを処理するアプリケーションを Java で作成し、CloudTrail からの提供後にログファイルが変更されていないことを確認します。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail ログファイルの使用」を参照してください。

手動モニタリングツール

DAX のモニタリングでもう 1 つ重要な点は、CloudWatch のアラームの対象外の項目を手動でモニタリングすることです。DAX、CloudWatch、Trusted Advisor などの AWS Management Console ダッシュボードには、AWS 環境の状態が一目でわかるビューが表示されます。DAX のログファイルを確認することもお勧めします。

- DAX ダッシュボードでは以下について確認できます。

  - サービス状態
- CloudWatch のホームページでは以下について確認できます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して、次のことが可能です。

  - 重視するサービスをモニタリングするための[カスタマイズしたダッシュボード](#)を作成する。
  - メトリクスデータをグラフ化して、問題をトラブルシューティングして、傾向を確認する。
  - AWS リソースのすべてのメトリクスを検索して参照する。
  - 問題があることを通知するアラームを作成および編集する。

Amazon CloudWatch によるモニタリング

Amazon CloudWatch を使用して DynamoDB Accelerator (DAX) をモニタリングできます。CloudWatch は、DAX から raw データを収集し、読み込み可能なほぼリアルタイムのメトリクスに加工します。これらの統計は 2 週間記録されます。その後は履歴情報にアクセスして、ウェブアプリケーションやサービスの動作をより的確に把握できます。デフォルトでは、DAX メトリクスデータは CloudWatch に自動的に送信されます。詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」を参照してください。

トピック

DAX メトリクスの使用方法は?

DAX によってレポートされるメトリクスが提供する情報は、さまざまな方法で分析できます。以下のリストは、メトリクスの一般的な利用方法をいくつか示しています。ここで紹介するのは開始するための提案事項です。すべてを網羅しているわけではありません。

目的	関連するメトリクス
システムエラーが発生したかどうかを判断する	FaultRequestCount をモニタリングして HTTP 500 (サーバーエラー) コードを発生させたリクエストがあるかどうかを判断します。これは DAX の内部サービスエラー、または基盤となるテーブルの SystemErrors メトリクス の HTTP 500 の可能性を示します。
ユーザーエラーが発生したかどうかを判断する	ErrorRequestCount をモニタリングして HTTP 400 (クライアントエラー) コードを発生させたリクエストがあるかどうかを判断します。エラーカウントが増えている場合、よく調べて正しいクライアントリクエストを送信していることを確認してください。
キャッシュミスが発生したかどうかを判断する	ItemCacheMisses をモニタリングして、項目がキャッシュに見つからなかった回数を判別します。また、QueryCacheMisses と ScanCacheMisses をモニタリングして、クエリやスキャンの結果がキャッシュで見つからなかった回数を判別します。
キャッシュヒット率をモニタリングする	CloudWatch Metric Math を使用し、キャッシュヒット率メトリクスを数式を使用して定義します。 たとえば、項目キャッシュの場合、m1/SUM([m1, m2])*100 という式を使用できます。m1 はクラスターの ItemCacheHits メトリクス、m2 は ItemCacheMisses メトリクスを意味します。クエリキャッシュやスキャンキャッシュでは、対応するクエリやスキャンのキャッシュメトリクスを使用して、次のサンプルパターンに従います。

DAX メトリクスとディメンションの表示

Amazon DynamoDB を操作するとき、メトリクスとディメンションが Amazon CloudWatch に送信されます。以下の手順を使用して、DynamoDB Accelerator (DAX) のメトリクスを表示できます。

メトリクスを表示するには (コンソール)

メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [DAX] 名前空間を選択します。

メトリクス () を表示するにはAWS CLI

- コマンドプロンプトで、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

DAX メトリクスとディメンション

以下のセクションでは、DAX が CloudWatch に送信するメトリクスとディメンションについて説明します。

DAX メトリクス

次のメトリクスは DAX から入手できます。DAX は、値がゼロ以外のメトリクスのみを CloudWatch に送信します。

> Note
>
> CloudWatch は 1 分間隔で以下の DAX メトリクスを集計します。
>
> - `CPUUtilization`
> - `CacheMemoryUtilization`
> - `NetworkBytesIn`
> - `NetworkBytesOut`
> - `NetworkPacketsIn`
> - `NetworkPacketsOut`
> - `GetItemRequestCount`
> - `BatchGetItemRequestCount`

- `BatchWriteItemRequestCount`
- `DeleteItemRequestCount`
- `PutItemRequestCount`
- `UpdateItemRequestCount`
- `TransactWriteItemsCount`
- `TransactGetItemsCount`
- `ItemCacheHits`
- `ItemCacheMisses`
- `QueryCacheHits`
- `QueryCacheMisses`
- `ScanCacheHits`
- `ScanCacheMisses`
- `TotalRequestCount`
- `ErrorRequestCount`
- `FaultRequestCount`
- `FailedRequestCount`
- `QueryRequestCount`
- `ScanRequestCount`
- `ClientConnections`
- `EstimatedDbSize`
- `EvictedSize`
- `CPUCreditUsage`
- `CPUCreditBalance`
- `CPUSurplusCreditBalance`
- `CPUSurplusCreditsCharged`

`Average` や `Sum` など、すべての統計が必ずしも常にすべてのメトリクスに適用可能であるとは限りません。ただし、これらの値はすべて DAX コンソール経由で利用できます。または CloudWatch コンソール、AWS CLI、AWS SDK を使用してすべてのメトリクスを利用できます。次の表は、各メトリクスに適用可能な有効な統計のリストを示します。

メトリクス	説明
CPUUtilization	ノードまたはクラスタの CPU 使用率の割合。 単位: Percent 有効な統計: • Minimum • Maximum • Average
CacheMemoryUtilization	ノードまたはクラスター上のアイテムキャッシュおよびクエリキャッシュによって使用されている使用可能なキャッシュメモリの割合。メモリ使用率が 100% に達する前に、キャッシュされたデータが削除され始めます (EvictedSize メトリクスを参照)。いずれかのノードで CacheMemoryUtilization が 100% に達すると書き込み要求が抑制されるので、より大きなノードタイプのクラスターへの切り替えを検討する必要があります。 単位: Percent 有効な統計: • Minimum • Maximum • Average
NetworkBytesIn	ノードまたはクラスターによってすべてのネットワークインターフェイスで受信されたバイトの数。 単位: Bytes 有効な統計: • Minimum • Maximum

メトリクス	説明
	• Average
`NetworkBytesOut`	ノートまたはクラスターによってすべてのネットワークインターフェイスで送信されたバイトの数。このメトリクスは、送信トラフィックのボリュームを単一のノードまたはクラスターでのバイト数として識別します。 単位: `Bytes` 有効な統計: • `Minimum` • `Maximum` • `Average`
`NetworkPacketsIn`	ノードまたはクラスターによってすべてのネットワークインターフェイスで受信されたパケットの数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average`

メトリクス	説明
NetworkPacketsOut	ノートまたはクラスターによってすべてのネットワークインターフェイスで送信されたパケットの数。このメトリクスは、送信トラフィックのボリュームを単一のノードまたはクラスターでのパケット数として識別します。 単位: Count 有効な統計: • Minimum • Maximum • Average
GetItemRequestCount	ノードまたはクラスターで処理された GetItem リクエストの数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
BatchGetItemRequestCount	ノードまたはクラスターで処理された BatchGetItem リクエストの数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum
BatchWriteItemRequestCount	ノードまたはクラスターで処理された BatchWriteItem リクエストの数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
DeleteItemRequestCount	ノードまたはクラスターで処理された DeleteItem リクエストの数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum
PutItemRequestCount	ノードまたはクラスターで処理された PutItem リクエストの数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
UpdateItemRequestCount	ノードまたはクラスターで処理された UpdateItem リクエストの数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum
TransactWriteItemsCount	ノードまたはクラスターで処理された TransactWriteItems リクエストの数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
`TransactGetItemsCount`	ノードまたはクラスターで処理された `TransactGetItems` リクエストの数。 単位: `Count` 有効な統計: - `Minimum` - `Maximum` - `Average` - `SampleCount` - `Sum`
`ItemCacheHits`	ノードまたはクラスターによってキャッシュから項目が返された回数。 単位: `Count` 有効な統計: - `Minimum` - `Maximum` - `Average` - `SampleCount` - `Sum`

メトリクス	説明
`ItemCacheMisses`	項目がノードキャッシュまたはクラスターキャッシュに存在せず、DynamoDB から取得する必要があった回数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`QueryCacheHits`	ノードキャッシュまたはクラスターキャッシュからクエリ結果が返された回数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

メトリクス	説明
`QueryCacheMisses`	クエリ結果がノードキャッシュまたはクラスターキャッシュに存在せず、DynamoDB から取得する必要があった回数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanCacheHits`	ノードキャッシュまたはクラスターキャッシュからスキャン結果が返された回数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

メトリクス	説明
`ScanCacheMisses`	スキャン結果がノードキャッシュまたはクラスターキャッシュに存在せず、DynamoDB から取得する必要があった回数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TotalRequestCount`	ノードまたはクラスターによって処理されたリクエストの合計数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

メトリクス	説明
`ErrorRequestCount`	ノードまたはクラスターによって報告されたユーザーエラーの原因となったリクエストの合計数。ノードまたはクラスターによってスロットリングされたリクエストが含まれます。 単位: `Count` 有効な統計: - `Minimum` - `Maximum` - `Average` - `SampleCount` - `Sum`
`ThrottledRequestCount`	ノードまたはクラスターによってスロットリングされたリクエストの合計数。DynamoDB によってスロットリングされたリクエストは含まれません。DynamoDB メトリクス を使用してモニタリングできます。 単位: `Count` 有効な統計: - `Minimum` - `Maximum` - `Average` - `SampleCount` - `Sum`

メトリクス	説明
FaultRequestCount	ノードまたはクラスターによって報告された内部エラーの原因となったリクエストの総数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum
FailedRequestCount	ノードまたはクラスターによって報告されたエラーの原因となったリクエストの合計数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
`QueryRequestCount`	ノードまたはクラスターによって処理されたクエリリクエストの数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ScanRequestCount`	ノードまたはクラスターによって処理されたスキャンリクエストの数。 単位: `Count` 有効な統計: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

メトリクス	説明
ClientConnections	クライアントから行われたノードまたはクラスターへの同時接続の数。 単位: Count 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum
EstimatedDbSize	ノードまたはクラスターによって項目キャッシュおよびクエリキャッシュにキャッシュされるデータ量の近似。 単位: Bytes 有効な統計: • Minimum • Maximum • Average

メトリクス	説明
EvictedSize	新しくリクエストされたデータ用のスペースを作るために、ノードまたはクラスターによってエビクトされたデータの量。ミス率が上昇し、この指標も増加している場合は、ワーキングセットが増加したと考えられます。より大きなノードタイプのクラスターへの切り替えを検討する必要があります。 単位: Bytes 有効な統計: • Minimum • Maximum • Average • Sum
CPUCreditUsage	CPU 使用率に関してノードで消費される CPU クレジットの数。1 つの CPU クレジットは、1 個の vCPU が 100% の使用率で 1 分間実行されること、または、vCPU、使用率、時間の同等の組み合わせ (たとえば、1 個の vCPU が 50% の使用率で 2 分間実行されるか、2 個の vCPU が 25% の使用率で 2 分間実行される) に相当します。 CPU クレジットメトリクスは、5 分間隔でのみ利用可能です。5 分を超える期間を指定する場合は、Average の代わりに Sum 統計を使用します。 単位: Credits (vCPU-minutes) 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
CPUCreditBalance	ノードが起動または開始後に蓄積した獲得 CPU クレジットの数。 クレジットは、獲得後にクレジット残高に蓄積され、消費されるとクレジット残高から削除されます。クレジット残高には、DAX ノードサイズによって決まる上限があります。制限に到達すると、獲得された新しいクレジットはすべて破棄されます。 CPUCreditBalance のクレジットは、ノードがそのベースライン CPU 使用率を超えてバーストするために消費できます。 単位: Credits (vCPU-minutes) 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
CPUSurplusCreditBalance	CPUCreditBalance 値がゼロの場合に DAX ノードによって消費された余剰クレジットの数。 CPUSurplusCreditBalance 値は獲得した CPU クレジットによって支払われます。余剰クレジットの数が、24 時間にノードが獲得できるクレジットの最大数を超えている場合、最大数を超えて消費された余剰クレジットに対しては料金が発生します。 単位: Credits (vCPU-minutes) 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

メトリクス	説明
CPUSurplusCreditsCharged	獲得 CPU クレジットにより支払われないために追加料金が発生した消費済み余剰クレジットの数。 消費済み余剰クレジットは、消費された余剰クレジットがノードで 24 時間に獲得できる最大クレジット数を超えている場合に課金されます。最大数を越えて消費された余剰クレジットは、時間の最後またはノードが終了したときに課金されます。 単位: Credits (vCPU-minutes) 有効な統計: • Minimum • Maximum • Average • SampleCount • Sum

> Note
>
> CPUCreditUsage、CPUCreditBalance、CPUSurplusCreditBalance、および CPUSurplusCreditsCharged メトリクスは T3 ノードでのみ使用できます。

DAX メトリクスのディメンション

DAX のメトリクスは、アカウント、クラスター ID、またはクラスター ID とノード ID の組み合わせの値によって修飾されます。CloudWatch コンソールを使用して、以下の表に示すいずれかのディメンションに従って DAX データを取得できます。

ディメンション	説明
`Account`	アカウント内のすべてのノードで集計された統計情報を提供します。
`ClusterId`	データをクラスターに制限します。
`ClusterId, NodeId`	データをクラスター内のノードに制限します。

DAX をモニタリングする CloudWatch アラームを作成します

アラームの状態が変わったときに Amazon Simple Notification Service (Amazon SNS) メッセージを送信する Amazon CloudWatch のアラームを作成することができます。指定した期間中、1 つのアラームが 1 つのメトリクスを監視します。このアラームは、複数の期間にわたる一定のしきい値とメトリクスの値の関係性に基づき、1 つ以上のアクションを実行します。アクションは、Amazon SNS のトピックまたは Auto Scaling のポリシーに送信される通知です。アラームは、持続している状態変化に対してのみアクションを呼び出します。CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。

クエリキャッシュミスに関する通知を受け取る方法は?

1. Amazon SNS トピック (`arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`) を作成します。

詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon Simple Notification Service の設定」を参照してください。

2. アラームを作成します。

```
aws cloudwatch put-metric-alarm \
    --alarm-name QueryCacheMissesAlarm  \
    --alarm-description "Alarm over query cache misses" \
    --namespace AWS/DAX \
    --metric-name QueryCacheMisses  \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 8 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --period 60 \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-west-2:522194210714:QueryMissAlarm
```

3. アラームのテストを行います。

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value ALARM
```

> **Note**
>
> しきい値をアプリケーションに合わせた値に増減できます。また、CloudWatch Metric Math を使用して、キャッシュミス率メトリクスを定義し、そのメトリクスに対してアラームを設定できます。

リクエストがクラスターで内部エラーを起こした場合は、どのようにして通知されますか?

1. Amazon SNS トピック (`arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`) を作成します。

詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon Simple Notification Service の設定」を参照してください。

2. アラームを作成します。

```
aws cloudwatch put-metric-alarm \
    --alarm-name FaultRequestCountAlarm \
    --alarm-description "Alarm when a request causes an internal error" \
    --namespace AWS/DAX \
    --metric-name FaultRequestCount \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. アラームのテストを行います。

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value ALARM
```

本番モニタリング

さまざまなタイミングとロード条件でパフォーマンスを測定することにより、お客様の環境で通常の DAX のパフォーマンスのベースラインを確定する必要があります。DAX をモニタリングするには、モニタリングの履歴データを保存することを検討します。保存データを、最新のパフォーマンスデータと比較するベースラインとして使用し、通常のパフォーマンスのパターンやパフォーマンスの異常を検出して、問題への対応を検討することができます。

ベースラインを確立するには、最低でも、ロードテスト中と本番環境の両方で次の項目をモニタリングする必要があります。

- CPU 使用率とスロットル済みのリクエスト。これにより、クラスター内でより大きなノードタイプを使用する必要があるかどうかを判断できます。クラスターの CPU 使用率は、`CPUUtilization` CloudWatch メトリクスで利用できます。
- オペレーションのレイテンシー (クライアント側で測定) は、アプリケーションのレイテンシー要件内で一貫して維持する必要があります。
- `ErrorRequestCount`、`FaultRequestCount`、および `FailedRequestCount` CloudWatch メトリクスからわかるように、エラー率は低いままにしておきます。
- ネットワークバイト使用量。これにより、クラスター内でより大きなノードを使用する必要があるかどうかを判断できます。`NetworkBytesIn` と `NetworkBytesOut` メトリクスは CloudWatch で利用できるため、ここに記載されているインスタンスで使用可能なベースライン帯域幅と比較する必要があります。

> Note
>
> Amazon EC2 に記載されている利用可能なベースライン帯域幅はギガビット/秒 (Gbps) 単位ですが、`NetworkBytesIn` と `NetworkBytesOut` メトリクスはギガバイト/分 (GbPm) 単位です。Gbps を GbPm に換算して使用率を測定するには、ベースライン帯域幅に 7.5 を乗算して算出してください。

- キャッシュメモリの使用率と削除されたサイズ。これにより、クラスタのノードタイプに作業セットを保持するのに十分なメモリがあるかどうかを判断し、ない場合はより大きなノードタイプに切り替えることができます。

> Note
>
> キャッシュミスや書き込みが多数発生すると、キャッシュメモリの使用率が 100% まで増加し、可用性のダウンタイムが発生する可能性があります。

- クライアント接続。これを使用して、クラスターへの接続に原因がわからないスパイクがあるかどうかをモニターできます。

AWS CloudTrail を使用した DAX オペレーションのログ記録

Amazon DynamoDB Accelerator (DAX) は、DAX のユーザー、ロール、または AWS のサービスによって実行されたアクションをレコードするサービスである AWS CloudTrail と統合されています。

DAX と CloudTrail の詳細については、AWS CloudTrail を使用して DynamoDB オペレーションをログに記録する で DynamoDB Accelerator (DAX) のセクションを参照してください。

DAX T3/T2 バーストインスタンス

DAX では、固定パフォーマンスインスタンス (R4 や R5 など) とバーストパフォーマンスインスタンス (T2 や T3 など) を選択できます。バーストパフォーマンスインスタンスは、ベースラインレベルの CPU パフォーマンスを提供しながら、必要に応じてバーストする機能を備えています。

ベースラインパフォーマンスとバースト機能は、CPU クレジットにより管理されます。バーストパフォーマンスインスタンスは、ワークロードがベースラインしきい値を下回ったときに、インスタンスサイズによって決定されるレートで CPU クレジットを継続的に蓄積します。これらのクレジットは、ワークロードが増加すると消費される可能性があります。CPU クレジットは、フル CPU パフォーマンスを 1 分間実現します。

多くのワークロードは、一貫して高レベルの CPU が必要になることはありませんが、必要なときに非常に高速な CPU にフルアクセスできるため、大きなメリットが得られます。バーストパフォーマンスインスタンスは、これらのユースケースに特化して設計されています。データベースで一貫して高い CPU パフォーマンスが必要な場合は、固定パフォーマンスのインスタンスを使用することをお勧めします。

DAX T2 インスタンスファミリー

DAX T2 インスタンスは、バースト可能な汎用パフォーマンスインスタンスを使用し、ベースラインレベルの CPU パフォーマンスを提供しながら、そのベースラインを超えるとバーストする機能を備えています。T2 インスタンスは価格予測可能性を必要とするテストおよび開発ワークロードに適しています。DAX T2 インスタンスはスタンダードモードに設定されます。したがって、インスタンスの蓄積されたクレジットが少なくなると、CPU 使用率は徐々にベースラインレベルまで下がります。スタンダードモードの詳細については、Amazon EC2 ユーザーガイドの「バーストパフォーマンスインスタンスのスタンダードモード」を参照してください。

DAX T3 インスタンスファミリー

DAX T3 インスタンスは、次世代のバースタブルな汎用インスタンスタイプで、ベースラインレベルの CPU パフォーマンスを提供しながら、いつでも必要なだけ CPU 使用をバーストできる機能を備えています。T3 インスタンスはコンピューティング、メモリ、ネットワークリソースを提供するので、一時的な使用スパイクが発生する CPU 使用率の中程度のワークロードに最適です。DAX T3 インスタンスは、無制限モードに設定されています。したがって、24 時間にわたってベースラインを

超えてバーストすることがあり、その場合は追加料金が発生します。無制限モードの詳細については、Amazon EC2 Linux インスタンス用ユーザーガイドの「バーストパフォーマンスインスタンスの Unlimited モード」を参照してください。

DAX T3 インスタンスは、ワークロードが必要とする限り、高い CPU パフォーマンスを維持できます。ほとんどの汎用ワークロードでは、T3 インスタンスは追加料金なしで十分なパフォーマンスを提供します。24 時間のT3 インスタンスの平均 CPU 使用率がベースライン以下になった場合、1 時間ごとのインスタンス料金は自動的にすべての暫定スパイクをカバーします。

たとえば、dax.t3.small インスタンスは、1 時間あたり 24 CPU クレジットのレートで継続的にクレジットを受け取ります。この機能は、CPU コアの 20% に相当するベースラインパフォーマンスを提供します (20% x 60 分 = 12 分)。インスタンスが受け取ったクレジットを使用しない場合、それらは CPU クレジットバランスに最大 576 CPU クレジットまで格納されます。t3.small インスタンスはコアの 20% 以上にバーストする必要がある場合、このサージは CPU クレジットバランスから自動的に処理されます。

CPU クレジットバランスがゼロになると、DAX T2 インスタンスはベースラインパフォーマンスに制限されますが、DAX T3 インスタンスは CPU クレジットバランスがゼロの場合でも、ベースラインを超えてバーストする可能性があります。CPU 使用率がベースラインパフォーマンス以下である大部分のワークロードでは、t3.small の基本時間料金は、すべての CPU バーストをカバーします。インスタンスが CPU クレジットバランスがゼロになってから 24 時間にわたって平均 25% の CPU 使用率 (ベースラインを 5% 超過) で実行された場合、さらに 11.52 セント (9.6 セント/VCPU 時間 x 1 vCPU x 5% x 24 時間) が課金されます。料金の詳細については、「Amazon DynamoDB の料金表」を参照してください。

DAX のアクセスコントロール

DynamoDB アクセラレーター (DAX) は DynamoDB と連携して動作し、アプリケーションにキャッシングレイヤーをシームレスに追加するように設計されています。ただし、DAX および DynamoDB には、個別のアクセス制御のメカニズムがあります。どちらのサービスもそれぞれのセキュリティポリシーの実装に AWS Identity and Access Management (IAM) を使用しますが、セキュリティモデルは DAX と DynamoDB で異なります。

両方のセキュリティモデルを理解することを強くお勧めします。そうすることで、DAX を使用するアプリケーションに適切なセキュリティ手段を実装できます。

このセクションでは、DAX が提供するアクセスコントロールメカニズムを説明し、必要に合わせて調整できる IAM ポリシーの例を示します。

DynamoDB では、個別の DynamoDB リソースでユーザーが実行できるアクションを制限する IAM ポリシーを作成できます。たとえば、特定の DynamoDB テーブルに対して、ユーザーに読み込み専用アクションのみを許可するユーザーロールを作成できます。(詳細については、Amazon DynamoDB の Identity and Access Management を参照してください)。対して、DAX のセキュリティモデルでは、クラスターのセキュリティおよびクラスターがユーザーに代わって実行する DynamoDB API アクション機能が中心です。

> ⚠ Warning
>
> 現在 IAM ロールおよびポリシーを使用して DynamoDB テーブルのデータへのアクセスを制限している場合、DAX を使用することで、これらのポリシーが覆される場合があります。たとえば、あるユーザーが、DAX 経由では DynamoDB テーブルにアクセスできても、DynamoDB に直接アクセスすると同じテーブルに対する明示的なアクセスができないという場合があります。詳細については、「Amazon DynamoDB の Identity and Access Management」を参照してください。
>
> DAX では、DynamoDB のデータに対してユーザーレベルの分離を実施していません。代わりに、ユーザーは DAX クラスターにアクセスする際に、そのクラスターの IAM ポリシーのアクセス権限を継承します。そのため、DAX 経由で DynamoDB テーブルにアクセスする際、有効なアクセスコントロールは DAX クラスターの IAM ポリシーのアクセス権限のみです。他のアクセス権限は認識されません。
>
> 隔離が必要な場合は、追加の DAX クラスターを作成して、クラスターごとに IAM ポリシーを限定することをお勧めします。たとえば、複数の DAX クラスターを作成し、各クラスターに 1 つのテーブルのみに対してアクセスを許可することもできます。

DAX 用の IAM サービスロール

DAX クラスターを作成するときに、クラスターを IAM ロールと関連付ける必要があります。これは、クラスターのサービスロールと呼ばれます。

DAXCluster01 という名前の新しい DAX クラスターを作成するとします。DAXServiceRole というサービスロールを作成し、そのロールを DAXCluster01 と関連付けます。DAXServiceRole のポリシーは、DAXCluster01 と対話するユーザーの代理で DAXCluster01 が実行できる DynamoDB アクションを定義します。

サービスロールを作成する際に、DAXServiceRole と DAX サービス自体の信頼関係を指定する必要があります。信頼関係は、どのエンティティがロールを引き受けアクセス権限を利用できるかを決定します。以下は、DAXServiceRole の信頼関係のドキュメントの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

この信頼関係では、DAX クラスターが DAXServiceRole を引き受け、ユーザーに代わって DynamoDB API コールを実行します。

許可される DynamoDB API アクションは、DAXServiceRole にアタッチした IAM ポリシードキュメントに記載されています。以下に、ポリシードキュメントの例を示します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DaxAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
            ]
        }
    ]
}
```

このポリシーでは、DAX は、DynamoDB テーブルに対して必要な DynamoDB API アクションを実行できます。dynamodb:DescribeTable アクションは、DAX がテーブルに関するメタデータを維持するために必要です。その他のアクションは、テーブル内の項目に対して実行される読み込みおよび書き込みアクションです。Books という名前のテーブルは us-west-2 リージョンにあり、AWS アカウント ID 123456789012 によって所有されています。

> **Note**
>
> DAX は、クロスサービスアクセス中に混乱した代理問題を防止するメカニズムをサポートしています。詳細については、IAM ユーザーガイド の 混乱した代理問題 を参照してください。

DAX クラスターのアクセスを許可する IAM ポリシー

DAX クラスターを作成したら、ユーザーが DAX クラスターにアクセスできるように、ユーザーにアクセス許可を付与する必要があります。

例えば、Alice というユーザーに DAXCluster01 へのアクセス許可を付与するとします。最初に、受信者がアクセスできる DAX クラスターと DAX API アクションを定義する IAM ポリシー (AliceAccessPolicy) を作成します。次に、このポリシーをユーザー Alice にアタッチして、アクセスを付与します。

次に示すポリシードキュメントは、受取人に DAXCluster01 のフルアクセスを付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

ポリシードキュメントは DAX クラスターへのアクセスを許可しますが、DynamoDB に対するアクセス権限は付与しません。(DynamoDB のアクセス権限は、DAX サービスロールによって付与されます。)

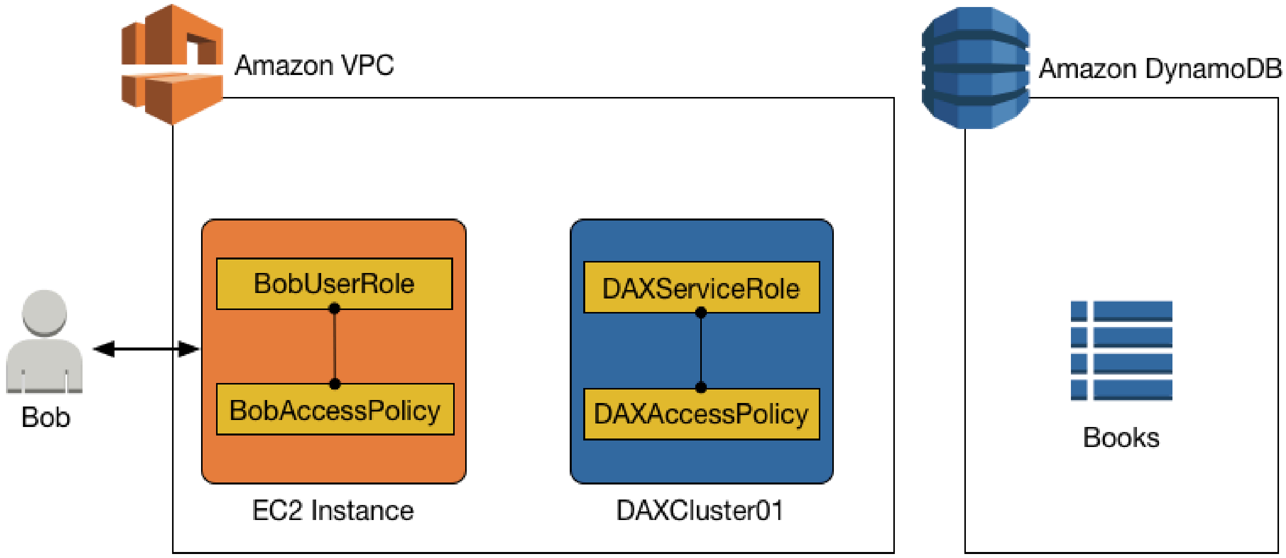
ユーザー Alice 用に、まず上記のポリシードキュメントを持った `AliceAccessPolicy` を作成します。次に、ポリシーを Alice にアタッチします。

> **Note**
>
> ポリシーをユーザーにアタッチする代わりに、IAM ロールにアタッチできます。この方法では、そのロールを引き受けるユーザーのすべてに、ポリシーで定義したアクセス権限があります。

ユーザーポリシーは、DAX サービスロールとともに、受信者が DAX 経由でアクセスできる DynamoDB リソースおよび API アクションを決定します。

導入事例: DynamoDB と DAX にアクセスする

次のシナリオは、DAX で使用する IAM ポリシーについての理解を深めるために役立ちます。(このシナリオは、このセクションの他の部分でも参照されます)。次の図は、このシナリオの大まかな概要を示しています。



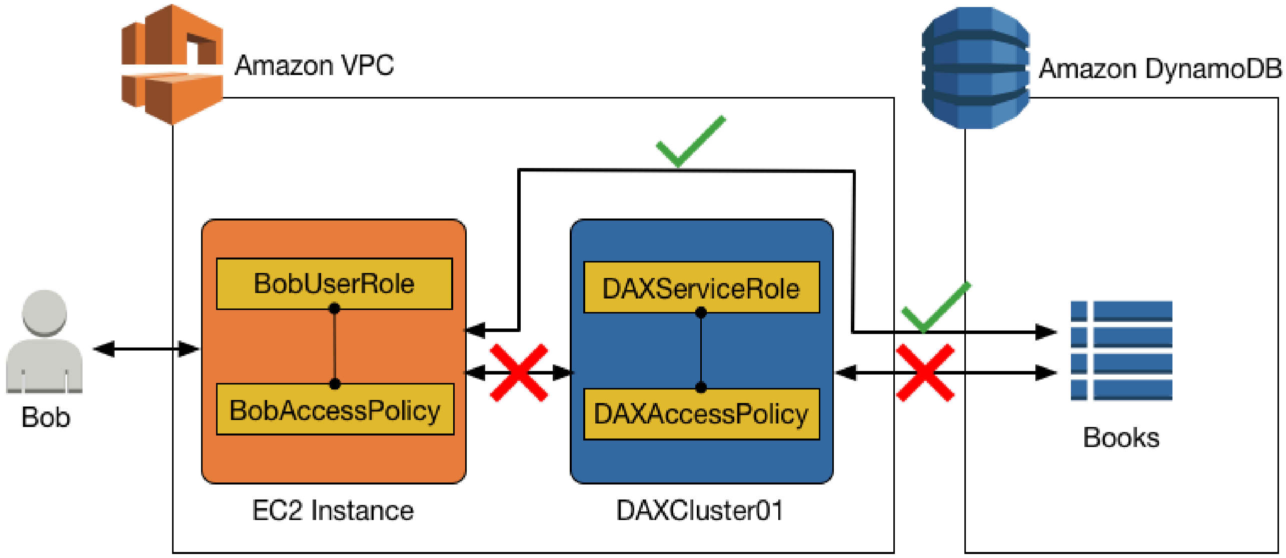
このシナリオには、次のエンティティがあります。

- ユーザー (Bob)。
- IAM ロール (BobUserRole)。Bob は実行時にこのロールを引き受けます。
- IAM ポリシー (BobAccessPolicy)。このポリシーは、BobUserRole にアタッチされています。BobAccessPolicy では、BobUserRole がアクセスできる DynamoDB リソースと DAX リソースが定義されます。
- DAX クラスター (DAXCluster01)。
- IAM サービスロール (DAXServiceRole)。このロールにより、DAXCluster01 は DynamoDB にアクセスできます。
- IAM ポリシー (DAXAccessPolicy)。このポリシーは、DAXServiceRole にアタッチされています。DAXAccessPolicy では、DAXCluster01 がアクセスできる DynamoDB API およびリソースが定義されます。
- DynamoDB テーブル (Books)。

BobAccessPolicy と DAXAccessPolicy のポリシーステートメントの組み合わせにより、Bob が Books に対して何ができるかが決まります。たとえば、Books に (DynamoDB エンドポイントを使用して) 直接アクセスできる、(DAX クラスターを使用して) 間接的にアクセスできる、またはその両

方などです。Books からデータを読み取る、Books にデータを書き込む、またはその両方が可能な場合もあります。

DynamoDB へのアクセスと DAX へのアクセス防止



DynamoDB テーブルに直接アクセスすることを許可する一方で、DAX クラスターを使用した間接的なアクセスを防止することもできます。DynamoDB への直接アクセスについては、BobUserRole のアクセス権限は (ロールにアタッチされている) BobAccessPolicy によって決まります。

DynamoDB (のみ) への読み込み専用アクセス

Bob は BobUserRole で DynamoDB にアクセスできます。このロールにアタッチされた IAM ポリシー (BobAccessPolicy) は、BobUserRole がアクセスできる DynamoDB テーブルと、BobUserRole が呼び出すことができる API を決定します。

BobAccessPolicy に対して、次のポリシードキュメントを検討してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

このドキュメントが BobAccessPolicy にアタッチされていると、BobUserRole は DynamoDB エンドポイントにアクセスし、Books テーブルで読み込み専用のオペレーションを実行できるようになります。

DAX はこのポリシーには含まれていないため、DAX 経由のアクセスは拒否されます。

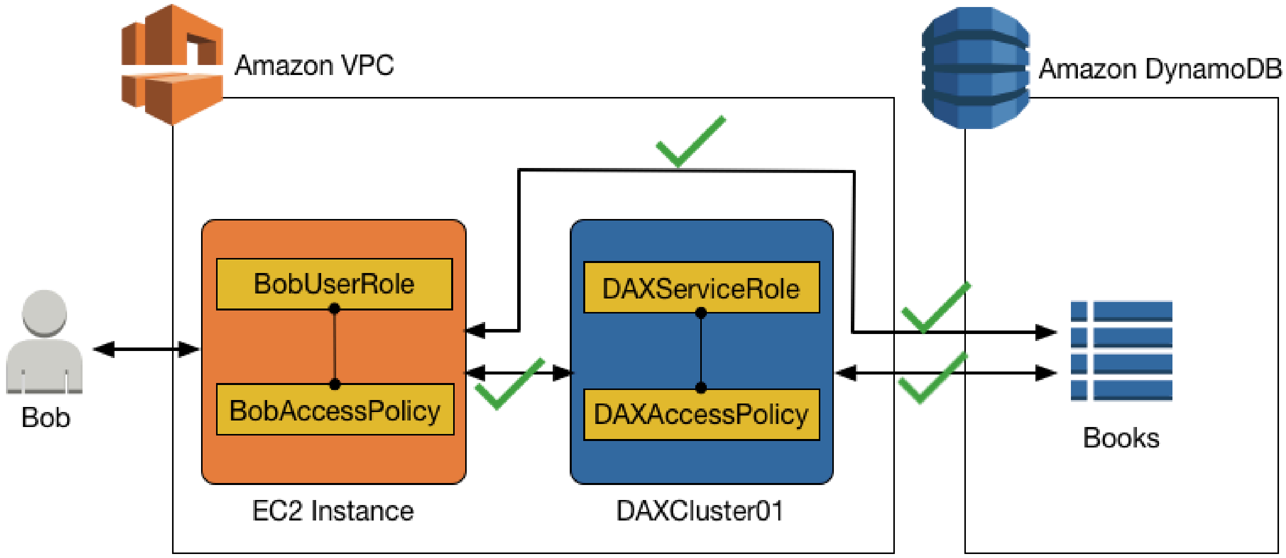
DynamoDB (のみ) への読み込み/書き込みアクセス

BobUserRole に DynamoDB への読み込み/書き込みアクセスが必要な場合、次のポリシーを使用できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

ここでも、DAX はこのポリシーには含まれていないため、DAX 経由のアクセスは拒否されます。

DynamoDB および DAX へのアクセス



DAX クラスターに対するアクセスを許可するには、IAM ポリシーに DAX 固有のアクションを含める必要があります。

次の DAX 固有のアクションは、DynamoDB API の類似した名前のアクションに対応しています。

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax:DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

`dax:EnclosingOperation` 条件キーについても同様です。

DynamoDB への読み込み専用アクセスと DAX への読み込み専用アクセス

Bob には、DynamoDB と DAX の両方から `Books` テーブルへの読み込み専用アクセスが必要であるとします。次のポリシー (`BobUserRole` にアタッチされている) により、このアクセスが付与されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

ポリシーには、DAX アクセス用のステートメント (`DAXAccessStmt`) と DynamoDBaccess 用の別のステートメント (`DynamoDBAccessStmt`) があります。これらのステートメントにより、Bob は、`GetItem`、`BatchGetItem`、`Query`、`Scan` リクエストを `DAXCluster01` に送信できるようになります。

ただし、`DAXCluster01` のサービスロールにも、DynamoDB の `Books` テーブルへの読み込み専用のアクセスが必要になります。`DAXServiceRole` にアタッチされた以下の IAM ポリシーは、この条件を満たします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

DynamoDB への読み込み/書き込みアクセスと DAX 経由の読み込み専用アクセス

特定のユーザーロールに、DAX 経由での読み込み専用アクセスを許可したまま、DynamoDB テーブルへの読み込み/書き込みアクセスを提供できます。

Bob の場合、`BobUserRole` の IAM ポリシーは、`Books` テーブルに対する DynamoDB の読み込み/書き込みアクションを許可しながら、`DAXCluster01` を介した読み込み専用アクションもサポートする必要があります。

`BobUserRole` に対する次のポリシードキュメントの例では、このアクセスが付与されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

さらに、DAXServiceRole には、DAXCluster01 が Books テーブルに対して読み込み専用のアクションを実行できるようにする IAM ポリシーが必要になります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:DescribeTable"
           ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

DynamoDB への読み込み/書き込みアクセスと DAX への読み込み/書き込みアクセス

次に、Bob に DynamoDB から直接、または DAXCluster01 からの間接的な Books テーブルへの読み込み/書き込みアクセスが必要であるとします。BobAccessPolicy にアタッチされている次のポリシードキュメントでは、このアクセスが付与されます。

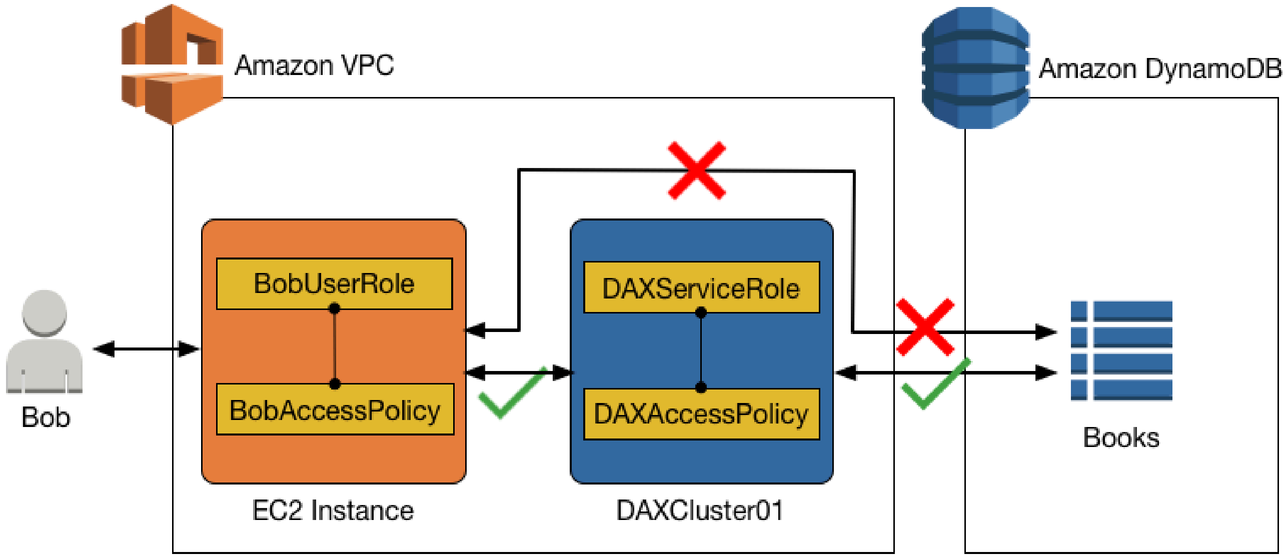
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

さらに、DAXServiceRole には、DAXCluster01 が Books テーブルに対する読み込み/書き込みアクションを許可する IAM ポリシーが必要になります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

DAX 経由での DynamoDB へのアクセスの許可と DynamoDB への直接アクセスの防止

このシナリオでは、Bob は Books テーブルに DAX 経由でアクセスできますが、DynamoDB の Books テーブルに直接アクセスすることはできません。したがって、Bob が DAX にアクセスすると、他の場合ではアクセスできない DynamoDB テーブルへのアクセスも取得します。DAX サービスロール用の IAM ポリシーを設定する際に、ユーザーアクセスポリシー経由で DAX クラスターへのアクセスを付与されたユーザーには、そのポリシーで指定されたテーブルへのアクセスも付与されることに注意してください。この場合、BobAccessPolicy には DAXAccessPolicy で指定されたテーブルへのアクセスが付与されます。



現在 IAM ロールおよびポリシーを使用して DynamoDB テーブルおよびデータへのアクセスを制限している場合、DAX を使用することで、これらのポリシーが覆される場合があります。次のポリシーでは、Bob は DynamoDB テーブルに DAX 経由でアクセスできますが、DynamoDB の同じテーブルに明示的に直接アクセスすることはできません。

`BobAccessPolicy` にアタッチされている次のポリシードキュメント (`BobUserRole`) では、このアクセスが付与されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        }
    ]
}
```

このアクセスポリシーには、DynamoDB に直接アクセスするアクセス許可がありません。

`BobAccessPolicy` と次の `DAXAccessPolicy` により、`BobUserRole` は、DynamoDB テーブル `Books` にアクセスできるようになります。ただし、`BobUserRole` は、`Books` テーブルに直接アクセスすることはできません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

この例に示すように、ユーザーアクセスポリシーおよび DAX クラスターアクセスポリシーのアクセスコントロールを設定する場合は、エンドツーエンドのアクセスを完全に理解し、最小権限の原則が守られていることを確認してください。また、DAX クラスターへのアクセスをユーザーに付与することで、以前に確立されたアクセスコントロールポリシーが覆されないことを確認してください。

保管時の DAX 暗号化

Amazon DynamoDB Accelerator (DAX) 保管時の暗号化は、基になるストレージへの不正アクセスからデータを保護することで、データ保護のレイヤーを追加します。組織のポリシー、業界や政府の規制、またはコンプライアンス要件によって、データを保護するために保管時の暗号化の使用が求められる場合があります。暗号化を使用すると、クラウドにデプロイされたアプリケーションデータの安全性を向上できます。

保管時の暗号化を使用して、DAX によってディスクに保持されるデータが 256 ビット Advanced Encryption Standard (AES-256 暗号化とも呼ばれる) を使用して暗号化されます。DAX は、プライマリノードからリードレプリカへの変更の伝播の一部として、データをディスクに書き込みます。

DAX 保管時の暗号化には、クラスターの暗号化に使用される単一サービスデフォルトキーを管理する AWS Key Management Service (AWS KMS) を自動的に統合します。DAX クラスターを作成するときにサービスデフォルトキーが存在しない場合、AWS KMS は自動的に新しい AWS マネージドキーを作成します。このキーは、この先作成するクラスターの暗号化に使用されます。AWS KMS は、安全で可用性の高いハードウェアとソフトウェアを組み合わせて、クラウド向けに拡張されたキー管理システムを提供します。

データが暗号化されると、DAX はパフォーマンスの影響を最小限に抑えながら、データの復号を透過的に処理します。暗号化を使用するためにアプリケーションを変更する必要はありません。

> **Note**
>
> DAX はすべての DAX オペレーションについて AWS KMS を呼び出すことはしません。DAX は、クラスター起動時にのみキーを使用します。アクセス権限が取り消された場合でも、DAX はクラスターがシャットダウンされるまで引き続きデータにアクセスできます。顧客が指定した AWS KMS キーはサポートされません。

保管時の DAX 暗号化は、次のクラスターノードタイプに使用できます。

ファミリー	ノードの種類
メモリの最適化 (R4 および R5)	dax.r4.large dax.r4.xlarge dax.r4.2xlarge

ファミリー	ノードの種類
	dax.r4.4xlarge dax.r4.8xlarge dax.r4.16xlarge dax.r5.large dax.r5.xlarge dax.r5.2xlarge dax.r5.4xlarge dax.r5.8xlarge dax.r5.12xlarge dax.r5.16xlarge dax.r5.24xlarge
汎用 (T2)	dax.t2.small dax.t2.medium
汎用 (T3)	dax.t3.small dax.t3.medium

> ⚠ Important
>
> 保管時の DAX 暗号化は `dax.r3.*` ノードタイプではサポートされません。

クラスターが作成された後は、保管時の暗号化を有効または無効にすることはできません。作成時に有効でなかった場合には、保管時の暗号化を有効にするためにクラスターを再度作成する必要があります。

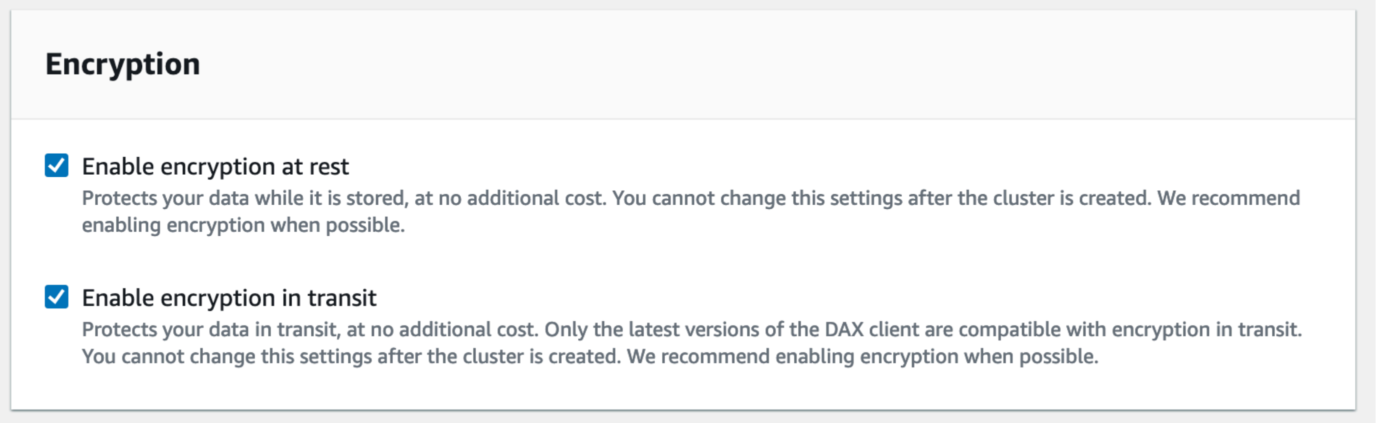
DAX 保管時の暗号化は、追加コストなしで提供されます (AWS KMS 暗号化キーの利用料金適用)。料金に関する詳細については、「Amazon DynamoDB の料金表」を参照してください。

AWS Management Console を使用した保管時の暗号化の有効化

コンソールを使用して、次の手順に従ってテーブルに対する保管時の DAX 暗号化を有効にします。

保管時の DAX 暗号化を有効にするには

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインの、[DAX] の [クラスター] を選択します。
3. [クラスターを作成] を選択します。
4. [クラスター名] に、クラスターの短縮名を入力します。クラスターのすべてのノードに [ノードタイプ] を選択して、クラスターサイズに、**3** ノードを使用します。
5. [暗号化] で、[暗号化の有効化] が選択されていることを確認します。



Encryption

☑ Enable encryption at rest
Protects your data while it is stored, at no additional cost. You cannot change this settings after the cluster is created. We recommend enabling encryption when possible.

☑ Enable encryption in transit
Protects your data in transit, at no additional cost. Only the latest versions of the DAX client are compatible with encryption in transit. You cannot change this settings after the cluster is created. We recommend enabling encryption when possible.

6. IAM ロール、サブネットグループ、セキュリティグループ、およびクラスター設定を選択した後、[クラスターの起動] を選択します。

クラスターが暗号化されていることを確認するには、[クラスター] ペインでクラスターの詳細を参照します。暗号化が [有効] である必要があります。

転送時の DAX 暗号化

Amazon DynamoDB Accelerator (DAX) は、アプリケーションと DAX クラスター間でデータを転送する際の暗号化をサポートするため、厳しい暗号化要件を持つアプリケーションでも DAX を使用できます。

転送中に暗号化を選択するかどうかにかかわらず、アプリケーションと DAX クラスター間のトラフィックは Amazon VPC に残ります。このトラフィックは、クラスターのノードに添付されている VPC 内のプライベート IP を使用して、Elastic Network Interface にルーティングされます。信頼境界として VPC を使用すると、セキュリティグループ、ネットワーク ACL を使用したサブネットセグメンテーション、VPC フローレースなどのスタンダードツールを使用することで、データのセキュリティを大幅に制御できます。転送中の DAX 暗号化では、このベースラインレベルの機密性が高まり、アプリケーションとクラスター間のすべてのリクエストとレスポンスがトランスポートレベルセキュリティ (TLS) によって暗号化され、クラスターへの接続はクラスター x509 証明書の検証によって認証されます。DAX によってディスクに書き込まれたデータも、DAX クラスターを作成する際に保管時の暗号化 を選択すれば暗号化されます。

DAXなら、転送中の暗号化を簡単に使用できます。新しいクラスターを作成するときにこのオプションを選択するだけです。そして最近のバージョンの DAX クライアントアプリケーションに保管します。転送中に暗号化を使用するクラスターは、暗号化されていないトラフィックをサポートしないため、アプリケーションを誤って設定したり、暗号化をバイパスしたりすることはありません。DAX クライアントは、接続を確立するときに、クラスターの x509 証明書を使用してクラスターのアイデンティティを認証し、DAX リクエストが意図した場所に確実に送信されるようにします。DAX クラスターの作成方法はすべて (AWS Management Console、AWS CLI、すべての SDK、AWS CloudFormation)、転送中の暗号化をサポートします。

転送中の暗号化は、既存の DAX クラスターでは有効にできません。既存の DAX アプリケーションで転送中の暗号化を使用するには、転送中の暗号化を有効にした新しいクラスターを作成し、アプリケーションのトラフィックをそのクラスターに移行してから、古いクラスターを削除します。

DAX のサービスにリンクされた IAM ロールの使用

この Amazon DynamoDB Accelerator (DAX) では、AWS Identity and Access Management (IAM) のサービスにリンクされたロールが使用されます。サービスにリンクされたロールは、DAX に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは、DAX で事前定義されており、ユーザーの代わりに必要なサービスから他の AWS のサービスを呼び出すための、すべてのアクセス許可が付与されています。

サービスにリンクされたロールを使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、DAX の設定が簡単になります。サービスにリンクされたロールのアクセス許可は DAX により定義され、他に特別な定義がある場合を除き、DAX のみがそのロールを引き受けることができます。定義されたアクセス権限には、信頼ポリシーとアクセス権限ポリシーが含まれます。このアクセス許可ポリシーを他の IAM エンティティにアタッチすることはできません。

ロールを削除するには、まず関連リソースを削除します。これにより、DAX リソースに対するアクセス権限が意図せずに削除されることがなくなり、リソースを保護できます。

サービスにリンクされたロールをサポートしているその他のサービスの詳細については、IAM ユーザーガイドの「IAM と連携する AWS のサービス」を参照してください。[サービスにリンクされたロール] 列が [はい] になっているサービスを見つけます。該当するサービスにリンクされたロールに関するドキュメントを表示するには、[あり] リンクを選択します。

トピック

DAX のサービスにリンクされたロールにおけるアクセス許可

DAX では、`AWSServiceRoleForDAX` という名前のサービスにリンクされたロールを使用します。このロールにより、ユーザーの DAX クラスターに代わってサービスを呼び出すことを、DAX に許可します。

> **Important**
>
> サービスにリンクされたロール `AWSServiceRoleForDAX` により、DAX クラスターのセットアップと保守が簡単になります。ただし、このロールを使用する前に、各クラスターに DynamoDB へのアクセス許可を付与する必要があります。詳細については、「」を参照してくださいDAX のアクセスコントロール

`AWSServiceRoleForDAX` サービスリンクロールは、ロールの引き受けについて以下のサービスを信頼します。

- `dax.amazonaws.com`

ロールのアクセス許可ポリシーは、DAX に対し、指定したリソースで以下のアクションを実行することを許可します。

- ec2 でのアクション:

- `AuthorizeSecurityGroupIngress`
- `CreateNetworkInterface`
- `CreateSecurityGroup`
- `DeleteNetworkInterface`
- `DeleteSecurityGroup`
- `DescribeAvailabilityZones`
- `DescribeNetworkInterfaces`
- `DescribeSecurityGroups`
- `DescribeSubnets`
- `DescribeVpcs`
- `ModifyNetworkInterfaceAttribute`
- `RevokeSecurityGroupIngress`

サービスにリンクされたロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、アクセス許可を設定する必要があります。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの許可」を参照してください。

IAM エンティティが AWSServiceRoleForDAX サービスにリンクされたロールを作成するには

以下のポリシーステートメントを IAM エンティティのアクセス許可に追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

DAX でサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。サービスにリンクされたロールは、クラスターの作成時に DAX により自動的に作成されます。

> ⚠ Important
>
> 2018 年 2 月 28日より前に DAX サービスを使用していた場合は、サービスにリンクされたロールのサポートが開始された時点で、AWSServiceRoleForDAX ロールが DAX によってアカウントに作成されています。詳細については、IAM ユーザーガイドの「AWS アカウントに新しいロールが表示される」を参照してください。

サービスにリンクされたこのロールを削除したが、再作成する必要がある場合は、同じプロセスで、アカウントにロールを再作成することができます。インスタンスまたはクラスターを作成するたびに、(ユーザーに代わり) DAX により、サービスにリンクされたロールが再作成されます。

DAX のサービスにリンクされたロールの編集

DAX では、サービスにリンクされたロール AWSServiceRoleForDAX を編集することはできません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、IAM ユーザーガイドのサービスにリンクされたロールの編集を参照してください。

DAX でサービスにリンクされたロールの削除

サービスリンクロールを必要とする機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティティを排除できます。ただし、サービスにリンクされたロールを削除する前に、すべての DAX クラスターを削除する必要があります。

サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには、まずそのロールにアクティブなセッションがないことを確認し、そのロールで使用されているリソースをすべて削除する必要があります。

サービスにリンクされたロールにアクティブなセッションがあるかどうかを、IAM コンソールで確認するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

2. IAM コンソールのナビゲーションペインで [ロール] を選択します。次に、AWSServiceRoleForDAX ロールの名前 (チェックボックスではありません) を選択します。
3. 選択したロールの [Summary] ページで、[Access Advisor] タブを選択します。
4. [Access Advisor] タブで、サービスにリンクされたロールの最新のアクティビティを確認します。

> Note
>
> DAX で `AWSServiceRoleForDAX` ロールが使用されているかどうかが不明な場合は、このロールの削除を試みることができます。サービスがロールを使用している場合、削除は失敗し、ロールが使用されているリージョンを表示できます。ロールが使用されている場合は、そのロールを削除する前に DAX クラスターを削除する必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。

`AWSServiceRoleForDAX` ロールを削除する場合は、最初にすべての DAX クラスターを削除する必要があります。

すべての DAX クラスターの削除

以下のいずれかの手順を使用して、各 DAX クラスターを削除します。

DAX クラスターを削除するには (コンソール)

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ナビゲーションペインの、[DAX] で、[クラスター]を選択します。
3. [アクション] を選択し、[削除] を選択します。
4. [Delete cluster confirmation (クラスターの削除の確認)] ボックスで、[削除] を選択します。

DAX クラスターを削除するには (AWS CLI)

AWS CLI コマンドリファレンスの「delete-cluster」を参照してください。

DAX クラスターを削除するには (API)

「Amazon DynamoDB API リファレンス」の「DeleteCluster」を参照してください。

サービスにリンクされたロールの削除

IAM を使用して、サービスにリンクされたロールを手動で削除するには

`AWSServiceRoleForDAX` サービスにリンクされたロールを削除するには、IAM コンソール、IAM CLI、または IAM API を使用します。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

AWS アカウント間での DAX へのアクセス

ある AWS アカウント (アカウント A) で DynamoDB アクセラレーター (DAX) クラスターが実行されていて、別の AWS アカウント (アカウント B) の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスから DAX クラスターにアクセスできる必要があるとします。このチュートリアルでは、アカウント B の IAM ロールを持つ EC2 インスタンスをアカウント B で起動することでこれを実現します。次に、EC2 インスタンスの一時的なセキュリティ認証情報を使用して、アカウント A から IAM ロールを引き受けます。最後に、IAM ロールを引き受けることで得られた一時的なセキュリティ認証情報を使用し、アカウント A の DAX クラスターに対して Amazon VPC ピアリング接続を介してアプリケーション呼び出しを行います。これらのタスクを実行するには、両方の AWS アカウントで管理アクセスが必要です。

> ⚠ Important
>
> DAX クラスターに、別のアカウントから DynamoDB テーブルにアクセスさせることはできません。

トピック

- IAM のセットアップ
- VPC をセットアップする
- クロスアカウントアクセスを許可するように DAX クライアントを変更します

IAM のセットアップ

1. 次のコンテンツを含む `AssumeDaxRoleTrust.json` という名前のテキストファイルを作成します。これにより、Amazon EC2 がお客様に代わって作業できるようになります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. アカウント B で、インスタンスの起動時に Amazon EC2 が使用できるロールを作成します。

```
aws iam create-role \
    --role-name AssumeDaxRole \
    --assume-role-policy-document file://AssumeDaxRoleTrust.json
```

3. 次のコンテンツを含む `AssumeDaxRolePolicy.json` という名前のテキストファイルを作成します。これにより、アカウント B の EC2 インスタンスで実行されているコードが、アカウント A の IAM ロールを引き受けることができるようになります。*accountA* は、アカウント A の実際の ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::accountA:role/DaxCrossAccountRole"
        }
    ]
}
```

4. 作成したロールにそのポリシーを追加します。

```
aws iam put-role-policy \
    --role-name AssumeDaxRole \
    --policy-name AssumeDaxRolePolicy \
    --policy-document file://AssumeDaxRolePolicy.json
```

5. インスタンスプロファイルを作成して、インスタンスがロールを使用できるようにします。

```
aws iam create-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile
```

6. ロールをインスタンスプロファイルに関連付けます。

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AssumeDaxInstanceProfile \
    --role-name AssumeDaxRole
```

7. 次の内容を含む DaxCrossAccountRoleTrust.json という名前のテキストファイルを作成します。これにより、アカウント B がアカウント A ロールを引き受けることができるようになります。*accountB* は、アカウント B の実際の ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::accountB:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. アカウント A で、アカウント B が引き受けることができるロールを作成します。

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. DAX クラスターへのアクセスを許可する DaxCrossAccountPolicy.json という名前のテキストファイルを作成します。*dax-cluster-arn* は、DAX クラスターの正しい Amazon リソースネーム (ARN) に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "dax-cluster-arn"
        }
    ]
}
```

10. アカウント A で、ロールにポリシーを追加します。

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

VPC をセットアップする

1. アカウント A の DAX クラスターのサブネットグループを見つけます。*cluster-name* は、アカウント B がアクセスする必要のある DAX クラスターの名前に置き換えます。

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. その *subnet-group* を使用して、クラスターの VPC を見つけます。

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. その *vpc-id* を使用して、VPC の CIDR を見つけます。

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. アカウント B から、前のステップで見つかったものとは異なる重複しない CIDR を使用して VPC を作成します。次に、少なくとも 1 つのサブネットを作成します。AWS Management Console の VPC 作成ウィザードまたは AWS CLI のいずれかを使用できます。

5. アカウント B から、「VPC ピアリング接続の作成と使用」の説明に従って、アカウント A VPC へのピアリング接続をリクエストします。アカウント A から接続を受け入れます。

6. アカウント B から、新しい VPC のルーティングテーブルを見つけます。`vpc-id` は、アカウント B で作成した VPC の ID に置き換えます。

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. アカウント A の CIDR 宛てのトラフィックを VPC ピアリング接続に送信するルートを追加します。必ず、各#############は、アカウントの正しい値に置き換えます。

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. アカウント A から、以前に見つけた `vpc-id` を使用して DAX クラスターのルートテーブルを見つけます。

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. アカウント A から、アカウント B の CIDR 宛てのトラフィックを VPC ピアリング接続に送信するルートを追加します。各#############は、アカウントの正しい値に置き換えます。

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. アカウント B から、前のステップで作成した VPC で EC2 インスタンスを起動します。それに `AssumeDaxInstanceProfile` を指定します。AWS Management Console または AWS CLI い

ずれかの Launch Wizard を使用できます。インスタンスのセキュリティグループをメモしておきます。

11. アカウント A から、DAX クラスターが使用するセキュリティグループを見つけます。*cluster-name* を DAX クラスターの名前に置き換えることを記憶します。

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. アカウント B で作成した EC2 インスタンスのセキュリティグループからのインバウンドトラフィックを許可するように、DAX クラスターのセキュリティグループを更新します。*##############*は、アカウントの正しい値に置き換えることを記憶します。

```
aws ec2 authorize-security-group-ingress \
    --group-id accountA-security-group-id \
    --protocol tcp \
    --port 8111 \
    --source-group accountB-security-group-id \
    --group-owner accountB-id
```

この時点で、アカウント B の EC2 インスタンスのアプリケーションは、インスタンスプロファイルを使用して `arn:aws:iam::`*`accountA-id`*`:role/DaxCrossAccountRole` ロールを引き受け、DAX クラスターを使用できます。

クロスアカウントアクセスを許可するように DAX クライアントを変更します

> **Note**
>
> AWS Security Token Service (AWS STS) 認証情報は一時的な認証情報です。自動的に更新を処理するクライアントもあれば、認証情報を更新するための追加のロジックを必要とするクライアントもあります。該当するマニュアルの指示に従うことをお勧めします。

Java

このセクションでは、既存の DAX クライアントコードを変更して、クロスアカウント DAX アクセスを許可する方法について説明します。DAX クライアントコードがまだない場合は、「Java および DAX」チュートリアルで実際のコード例を見つけることができます。

1. 次のインポートを追加します。

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. AWS STS から認証情報プロバイダを取得し、DAX クライアントオブジェクトを作成します。必ず、各*#############*は、アカウントの正しい値に置き換えます。

```
AWSSecurityTokenService awsSecurityTokenService =
 AWSSecurityTokenServiceClientBuilder
     .standard()
     .withRegion(region)
     .build();

STSAssumeRoleSessionCredentialsProvider credentials =  new
 STSAssumeRoleSessionCredentialsProvider.Builder("arn:aws:iam::accountA:role/
RoleName", "TryDax")
     .withStsClient(awsSecurityTokenService)
     .build();

DynamoDB client = AmazonDaxClientBuilder.standard()
    .withRegion(region)
    .withEndpointConfiguration(dax_endpoint)
    .withCredentials(credentials)
    .build();
```

.NET

このセクションでは、既存の DAX クライアントコードを変更して、クロスアカウント DAX アクセスを許可する方法について説明します。DAX クライアントコードがまだない場合は、「.NET および DAX」チュートリアルで実際のコード例を見つけることができます。

1. AWSSDK.SecurityToken NuGet パッケージをソリューションに追加します。

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. SecurityToken および SecurityToken.Model パッケージを使用します。

```
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

3. AmazonSimpleTokenService から一時的な認証情報を取得し、ClusterDaxClient オブジェクトを作成します。必ず、各#############は、アカウントの正しい値に置き換えます。

```
IAmazonSecurityTokenService sts = new AmazonSecurityTokenServiceClient();

var assumeRoleResponse = sts.AssumeRole(new AssumeRoleRequest
{
    RoleArn = "arn:aws:iam::accountA:role/RoleName",
                RoleSessionName = "TryDax"
});

Credentials credentials = assumeRoleResponse.Credentials;

var clientConfig = new DaxClientConfig(dax_endpoint, port)
{
    AwsCredentials = assumeRoleResponse.Credentials

};

var client = new ClusterDaxClient(clientConfig);
```

Go

このセクションでは、既存の DAX クライアントコードを変更して、クロスアカウント DAX アクセスを許可する方法について説明します。DAX クライアントコードがまだない場合は、GitHub で実際のコード例を見つけることができます。

1. AWS STS およびセッションパッケージをインポートします。

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sts"
    "github.com/aws/aws-sdk-go/aws/credentials/stscreds"
)
```

2. AmazonSimpleTokenService から一時的な認証情報を取得し、DAX クライアントオブジェクトを作成します。必ず、各#############は、アカウントの正しい値に置き換えます。

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String(region)},
)
if err != nil {
    return nil, err
}

stsClient := sts.New(sess)
arp := &stscreds.AssumeRoleProvider{
                Duration:     900 * time.Second,
                ExpiryWindow: 10 * time.Second,
                RoleARN:      "arn:aws:iam::accountA:role/role_name",
                Client:       stsClient,
                RoleSessionName: "session_name",
        }cfg := dax.DefaultConfig()

cfg.HostPorts = []string{dax_endpoint}
cfg.Region = region
cfg.Credentials = credentials.NewCredentials(arp)
daxClient := dax.New(cfg)
```

Python

このセクションでは、既存の DAX クライアントコードを変更して、クロスアカウント DAX アクセスを許可する方法について説明します。DAX クライアントコードがまだない場合は、「Python および DAX」チュートリアルで実際のコード例を見つけることができます。

1. boto3 をインポートします。

```
import boto3
```

2. sts から一時的な認証情報を取得し、AmazonDaxClient オブジェクトを作成します。必ず、各#############は、アカウントの正しい値に置き換えます。

```
sts = boto3.client('sts')
stsresponse =
 sts.assume_role(RoleArn='arn:aws:iam::accountA:role/
RoleName',RoleSessionName='tryDax')
credentials = botocore.session.get_session()['Credentials']

dax = amazondax.AmazonDaxClient(session, region_name=region,
 endpoints=[dax_endpoint], aws_access_key_id=credentials['AccessKeyId'],
 aws_secret_access_key=credentials['SecretAccessKey'],
 aws_session_token=credentials['SessionToken'])
client = dax
```

Node.js

このセクションでは、既存の DAX クライアントコードを変更して、クロスアカウント DAX アクセスを許可する方法について説明します。DAX クライアントコードがまだない場合は、「Node.js および DAX」チュートリアルで実際のコード例を見つけることができます。必ず、各*############*は、アカウントの正しい値に置き換えます。

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
  return new Promise((resolve, reject) => {
    const sts = new AWS.STS();
    const roleParams = {
      RoleArn: 'arn:aws:iam::accountA:role/RoleName',
      RoleSessionName: 'tryDax',
    };
    sts.assumeRole(roleParams, (err, session) => {
      if(err) {
        reject(err);
      } else {
        resolve({
          accessKeyId: session.Credentials.AccessKeyId,
          secretAccessKey: session.Credentials.SecretAccessKey,
          sessionToken: session.Credentials.SessionToken,
        });
      }
    });
  });
};

const createDaxClient = async() => {
  const credentials = await getCredentials();
  const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
  return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
  client.get(...);
  ...
}).catch((error) => {
  console.log('Caught an error: ' + error);
});
```

DAX クラスターサイジングガイド

このガイドでは、Amazon Amazon DynamoDB Accelerator (DAX) クラスターの適切なサイズとノードタイプを、アプリケーションのために選択する際のアドバイスを提供します。以下の手順では、アプリケーションでの DAX トラフィックの見積もりと、クラスター設定の選択、およびそのテストを行うためのチュートリアルが得られます。

既に DAX クラスターが存在し、そのクラスターに適切な数とサイズのノードが備わっているどうかを評価したい場合は、「DAX クラスターのスケーリング」を参照してください。

トピック

- 概要
- トラフィックの見積もり
- 負荷テスト

概要

新しい DAX クラスターを作成する場合も、既存のクラスターをメンテナンスしている場合も、ワークロードに合わせてクラスターを適切にスケーリングすることが重要です。時間が経過し、アプリリ

ケーションのワークロードが変化するにつれて、スケーリングに関する決定を定期的に再検討して、それらが引き続き適切であることを確認する必要があります。

通常、このプロセスは以下の手順に従います。

1. トラフィックの見積もり。このステップでは、アプリケーションから DAX に送信されるトラフィックのボリュームや、そのトラフィックの性質 (そのオペレーションが読み込みか書き込みか)、および予想されるキャッシュヒット率についての推定を行います。
2. 負荷テスト。この手順では、クラスターを作成し、前のステップからの見積もりをミラーリングするクラスターにトラフィックを送信します。適切なクラスター設定が見つかるまで、この手順を繰り返します。
3. 本番稼働モニタリング。実稼働のアプリケーションで DAX を使用する場合には、クラスターをモニタリングしながら、時間的なワークロードの変動に応じてそのクラスターが正しくスケーリングされていることを、継続的に検証する必要があります。

トラフィックの見積もり

一般的な DAX ワークロードの特性は、以下の 3 つの主要素により表されます。

- キャッシュヒットレート
- 1 秒あたりの読み込みキャパシティーユニット (RCU)
- 1 秒あたりの書き込みキャパシティーユニット (WCU)

キャッシュヒットレートの見積もり

既存の DAX クラスターについては、ItemCacheHits および ItemCacheMisses の Amazon CloudWatch メトリクスを使用することでキャッシュヒット率を確認できます。キャッシュヒット率は ItemCacheHits / (ItemCacheHits + ItemCacheMisses) に等しくなります。ワークロードに Query または Scan オペレーションが含まれている場合は、QueryCacheHits、QueryCacheMisses、ScanCacheHits、および ScanCacheMisses の各メトリクスも確認する必要があります。キャッシュヒット率はアプリケーションごとに異なり、クラスターの有効期限 (TTL) 設定から大きな影響を受けます。DAX を使用するアプリケーションでの標準的なキャッシュヒット率は、85～95% です。

読み込みキャパシティユニットと書き込みキャパシティユニットの見積もり

アプリケーション用の DynamoDB テーブルが既に存在する場合は、ConsumedReadCapacityUnits および ConsumedWriteCapacityUnits の CloudWatch メトリクスを確認します。Sum 統計を使用して、期間の秒数で除算します。

DAX クラスターも既に存在する場合、DynamoDB の ConsumedReadCapacityUnits メトリクスでは、キャッシュミスのみが計算されることに注意してください。DAX クラスターで処理される 1 秒あたりの読み込みキャパシティーユニットの数を把握するには、このメトリクスからの数値をキャッシュミス率 (つまり、1 - キャッシュヒット率) で割ります。

まだ DynamoDB テーブルを用意していない場合は、「読み込みおよび書き込みのキャパシティユニット」のドキュメントを参照して、アプリケーションの推定リクエスト率、リクエストあたりのアクセスした項目数、および項目サイズに基づいて、トラフィックを見積もります。

トラフィックの見積もりを行う際、将来の増大と予想されるピークと予想外のピークを計画して、トラフィックの増加に十分なヘッドルームがクラスターにあることを確認します。

負荷テスト

トラフィックを見積もった後の次の手順は、負荷をかけてクラスター設定をテストすることです。

1. 最初の負荷テストでは、dax.r4.large ノードタイプ、最もコストの低い固定パフォーマンス、メモリ最適化ノードタイプから開始することをお勧めします。
2. 耐障害性の高いクラスターには、3 つのアベイラビリティーゾーンにまたがって分散される少なくとも 3 つのノードが必要です。この場合、アベイラビリティーゾーンが使用できなくなると、有効なアベイラビリティーゾーンの数が 3 分の 1 減少します。最初の負荷テストでは、3 ノードクラスター内の 1 つのアベイラビリティーゾーンの障害をシミュレートする 2 ノードクラスターから開始することをお勧めします。
3. 負荷テスト期間中、持続的なトラフィック (前のステップで見積もったもの) をテストクラスターに駆動します。
4. 負荷テスト中にクラスターのパフォーマンスをモニタリングします。

理想的には、負荷テスト中に駆動するトラフィックプロファイルは、アプリケーションの実際のトラフィックと可能な限り類似している必要があります。これには、オペレーションの分散 (GetItem 70%、Query 25%、PutItem 5% など)、各オペレーションのリクエスト率、リクエストごとにアクセスされた項目数、項目サイズの分散が含まれます。アプリケーションの予想キャッシュヒット率と

同様のキャッシュヒット率を達成するには、テストトラフィックでのキーの分散に細心の注意を払います。

> Note
>
> T2 ノードタイプ (`dax.t2.small` および `dax.t2.medium`) をロードテストする際は注意してください。T2 ノードタイプは、ノードの CPU クレジットバランスに応じて時間の経過とともに変化する、バースト可能な CPU パフォーマンスを提供します。T2 ノードで実行されている DAX クラスターのオペレーションが正常に見える場合でも、いずれかのノードがそのインスタンスのパフォーマンスベースラインを上回ってバーストする場合には、そのノードは蓄積 CPU クレジット残高を消費しています。クレジットで実行中のノードが少ない場合、パフォーマンスは次第にベースラインのパフォーマンスレベルに下がります。

負荷テストにより、DAX クラスターをモニタリングして、そこで使用しているノードタイプが適切なノードタイプであるかどうかを判断します。さらに、負荷テスト中にリクエスト率とキャッシュヒット率をモニタリングして、テストインフラストラクチャが意図したトラフィック量を実際に駆動していることを確認する必要があります。

選択したクラスターインスタンスタイプのネットワークバイト消費量に注意する必要があります。Amazon EC2 インスタンスで使用可能なベースライン帯域幅を超えると、クラスターがアプリケーションのワークロードに耐えられなくなり、スケーリングする必要が出てくるかもしれません。

負荷テストにより、選択したクラスター設定ではアプリケーションのワークロードを維持できないことが示された場合、特にクラスター内のプライマリノードでの高い CPU 使用率や、高い削除率、あるいは高いキャッシュメモリー使用率が見られる場合には、より大きなノードタイプに切り替える必要があります。ヒット率が一貫して高く、書き込みトラフィックに対する読み込みトラフィックの比率が高い場合は、ノードをクラスターに追加することを検討してください。どのような場合により大きなノードタイプを使用するか (垂直スケーリング)、ノードを追加するか (水平スケーリング) の詳細なガイダンスについては、「DAX クラスターのスケーリング」を参照してください。

クラスター設定を変更した後、負荷テストを繰り返す必要があります。

DynamoDB で DAX を使用するためのベストプラクティス

DynamoDB で DAX を使用する場合は、キャッシュのパフォーマンスと信頼性を向上させるためのベストプラクティスとして以下のトピックを参照することをお勧めします。

- DAX クラスターサイジングガイド

- 本番モニタリング

DAX API リファレンス

Amazon DynamoDB Accelerator (DAX) API の詳細については、「Amazon DynamoDB API リファレンス」の「Amazon DynamoDB Accelerator」を参照してください。

DynamoDB テーブルのデータモデリング

データモデリングに進む前に、DynamoDB の基礎を理解することが重要です。DynamoDB は、柔軟なスキーマを可能にする key-value NoSQL データベースです。各項目のキー属性以外の一連のデータ属性は、均一または個別のいずれかにすることができます。DynamoDB のキースキーマは、パーティションキーで項目を一意に識別する単純なプライマリキー形式であるか、パーティションキーとソートキーの組み合わせで項目を一意に定義する複合プライマリキー形式のいずれかです。パーティションキーは、データの物理的な場所を特定して取得するためにハッシュ化されます。そのため、データを均等に分散させるには、カーディナリティが高く、水平方向にスケーラブルな属性をパーティションキーとして選択することが重要です。ソートキー属性は、キースキーマではオプションです。ソートキーを使用すると、1 対多リレーションシップをモデル化し、DynamoDB で項目コレクションを作成できます。ソートキーは範囲キーとも呼ばれます。ソートキーを使用して項目コレクション内の項目をソートし、柔軟な範囲ベースの操作を可能にすることもできます。

DynamoDB のキースキーマの詳細とベストプラクティスについては、以下を参照してください。

- the section called “パーティションとデータ分散”
- the section called “パーティションキーの設計”
- the section called “ソートキーの設計”
- 適切な DynamoDB パーティションキーの選択

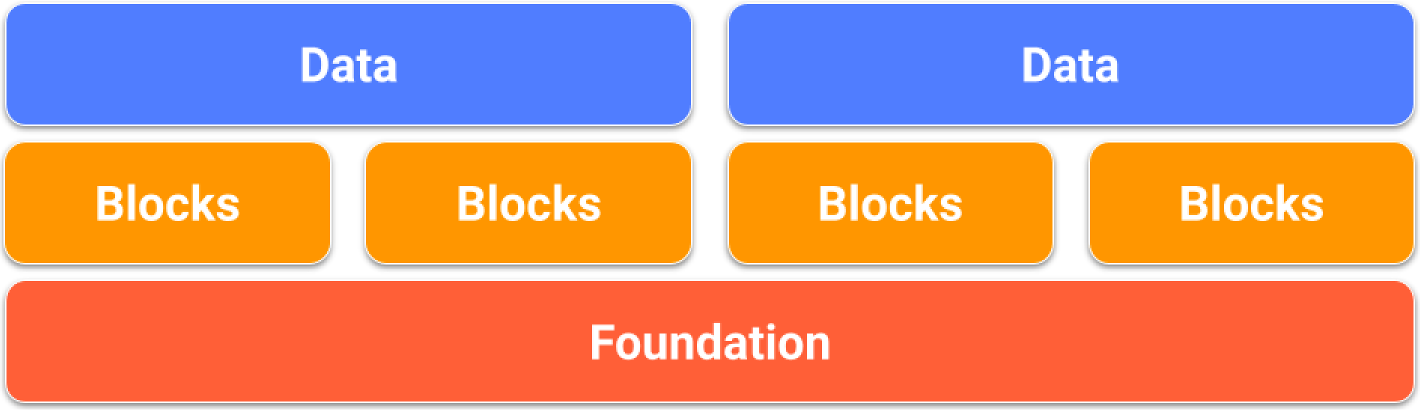
DynamoDB で追加のクエリパターンをサポートするには、多くの場合、セカンダリインデックスが必要になります。セカンダリインデックスは、同じデータがベーステーブルとは異なるキースキーマで整理されているシャドウテーブルです。ローカルセカンダリインデックス (LSI) は、ベーステーブルと同じパーティションキーを共有し、代替のソートキーを使用することでベーステーブルの容量を共有できます。グローバルセカンダリインデックス (GSI) では、ベーステーブルとは異なるパーティションキーと異なるソートキー属性を使用できます。つまり、GSI のスループット管理はベーステーブルとは独立しています。

セカンダリインデックスとベストプラクティスの詳細については、以下を参照してください。

- the section called “インデックスの使用”
- the section called “セカンダリインデックス”

ここで、データモデリングをもう少し詳しく見てみましょう。高度に最適化された柔軟なスキーマを設計するプロセスは、DynamoDB に限らず、どの NoSQL データベースであっても習得が難しいス

キルと言えます。このモジュールの目標は、ユースケースから本番環境へと導く、スキーマ設計のメンタルフローチャートの開発を支援することにあります。まず、あらゆる設計の基本となる、シングルテーブル設計とマルチテーブル設計の選択について簡単に説明します。次に、アプリケーションの組織上またはパフォーマンス上のさまざまな結果を達成するために使用できる多様な設計パターン (構成要素) を確認します。最後に、ユースケースや業界別に異なる完全なスキーマ設計パッケージを紹介します。

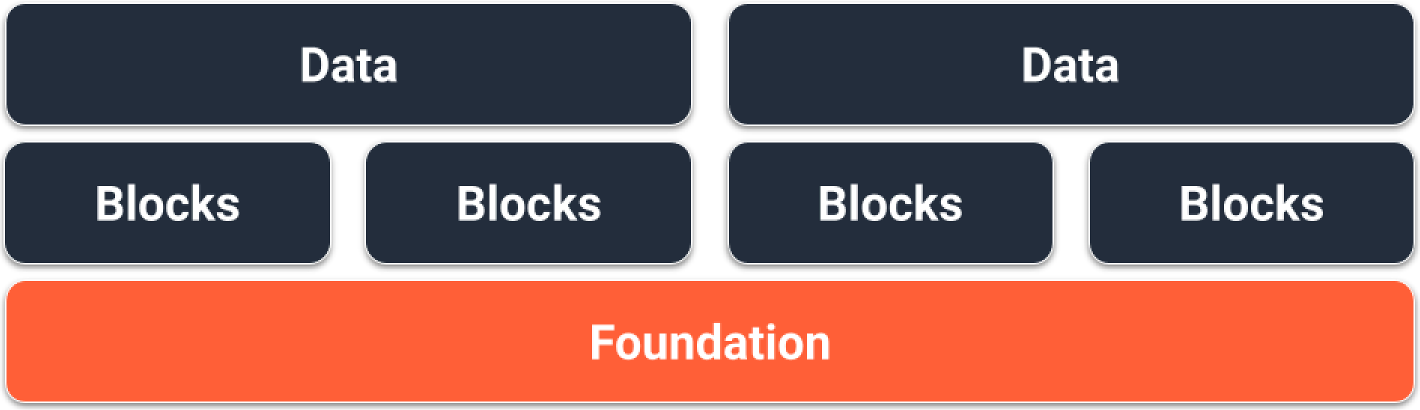


トピック

- DynamoDB のデータモデリングの基盤
- DynamoDB のデータモデリングの構成要素
- DynamoDB のデータモデリングスキーマ設計パッケージ

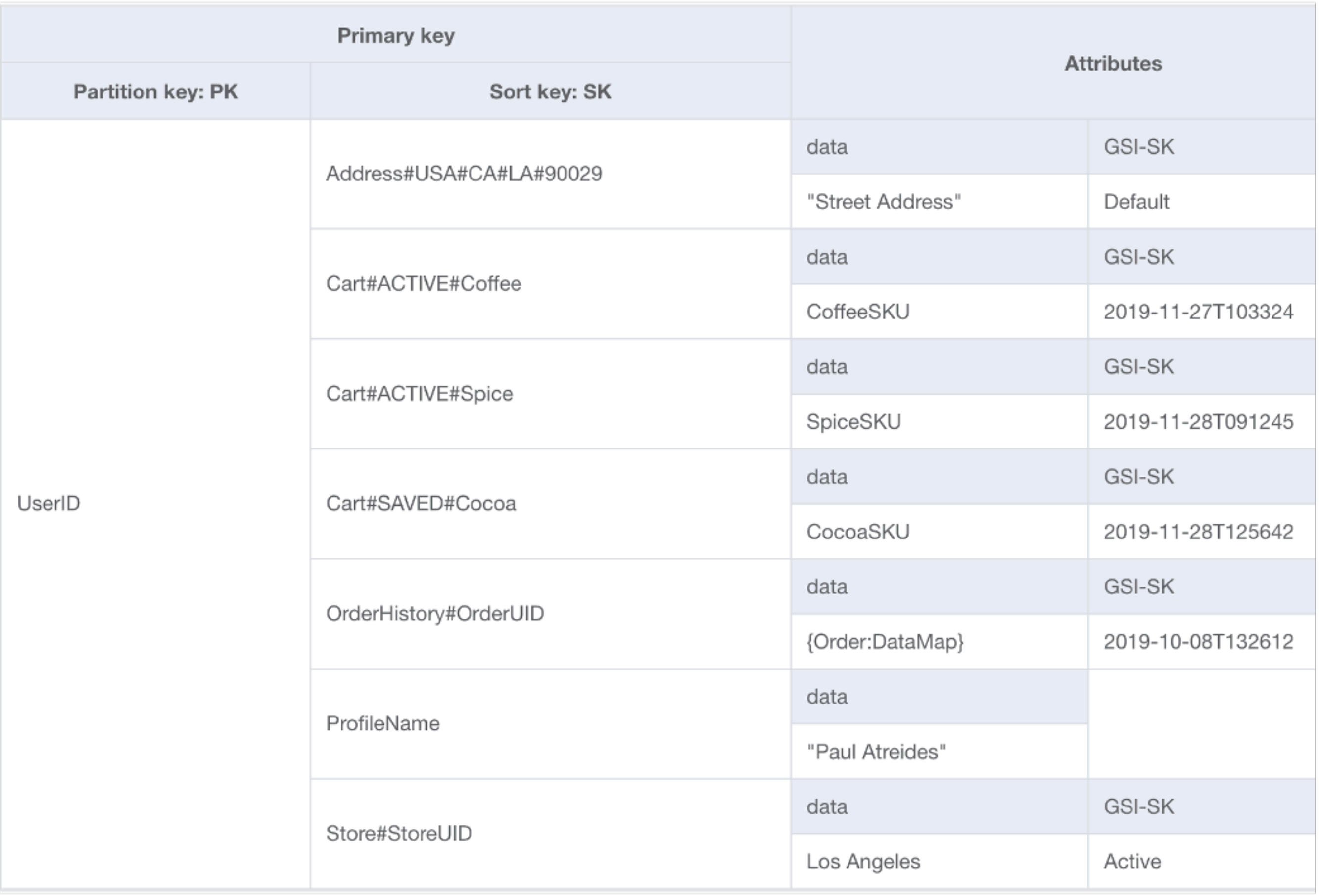
DynamoDB のデータモデリングの基盤

このセクションでは、基盤レイヤーについて、シングルテーブルとマルチテーブルという 2 種類のテーブル設計を検討します。



シングルテーブル設計の基盤

DynamoDB スキーマの基盤の 1 つとして、シングルテーブル設計を選択できます。シングルテーブル設計とは、複数のタイプ (エンティティ) のデータを単一の DynamoDB テーブルに保存できるパターンです。複数のテーブルやテーブル間の複雑な関係を維持する必要をなくすことで、データアクセスパターンを最適化し、パフォーマンスを向上させ、コストを削減することを目的としています。これが可能なのは、DynamoDB は、同じパーティションキーを持つ項目 (項目コレクションと呼ばれます) をいずれも同じパーティションに保存するためです。この設計では、異なる種類のデータが同じテーブル内に項目として保存され、各項目は一意のソートキーで識別されます。



Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

利点

- データの局所性により、1 回のデータベース呼び出しで複数のエンティティタイプのクエリをサポートします。
- 読み取りにかかる全体的な財務コストとレイテンシーコストを削減します。
  - 合計 4 KB 未満の 2 つの項目に対する単一のクエリは、結果整合性のある 0.5 RCU になります。

  - 合計 4 KB 未満の 2 つの項目に対する 2 つのクエリは、結果整合性のある 1 RCU になります (それぞれ 0.5 RCU)。
  - 2 つの別々のデータベース呼び出しを返すまでの時間は、1 回の呼び出しよりも平均して長くなります。
- 管理するテーブルの数が減ります。
  - アクセス許可を複数の IAM ロールやポリシーにわたって管理する必要はありません。
  - テーブルの容量管理はすべてのエンティティにわたって平均化され、通常は消費パターンがより予測可能になります。
  - モニタリングに必要なアラームの数が減ります。
  - カスタマーマネージド暗号化キーは、1 つのテーブルでローテーションするだけで済みます
- テーブルへのトラフィックをスムーズにします。
  - 複数の使用パターンを同じテーブルに集約することで、全体的な使用がよりスムーズになる傾向があり (株価指数のパフォーマンスが個々の株式よりもスムーズになる傾向があるように)、プロビジョニングモードテーブルでより高い使用率を達成するのに適しています。

欠点

- リレーショナルデータベースと比べて逆説的な設計であるため、ラーニングカーブが急になる場合があります。
- データ要件はすべてのエンティティタイプにわたって一貫している必要があります。
  - バックアップは全部かゼロかのどちらかであるため、ミッションクリティカルでないデータは、別のテーブルに保存することを検討してください。
  - テーブルの暗号化はすべての項目間で共有されます。テナントごとに暗号化要件が異なるマルチテナントアプリケーションでは、クライアント側の暗号化が必要になります
  - 履歴データと運用データが混在するテーブルでは、低頻度アクセスストレージクラスを有効にしてもあまりメリットがありません。詳細については、「テーブルクラス」を参照してください。
- エンティティのサブセットのみを処理する必要がある場合でも、すべての変更されたデータは DynamoDB Streams に伝播されます。
  - Lambda を使用する場合は、Lambda イベントフィルターのおかげで請求額に影響しませんが、Kinesis Consumer Libary を使用する場合は、追加費用が発生します。
- GraphQL を使用する場合、シングルテーブル設計の実装はより難しくなります。

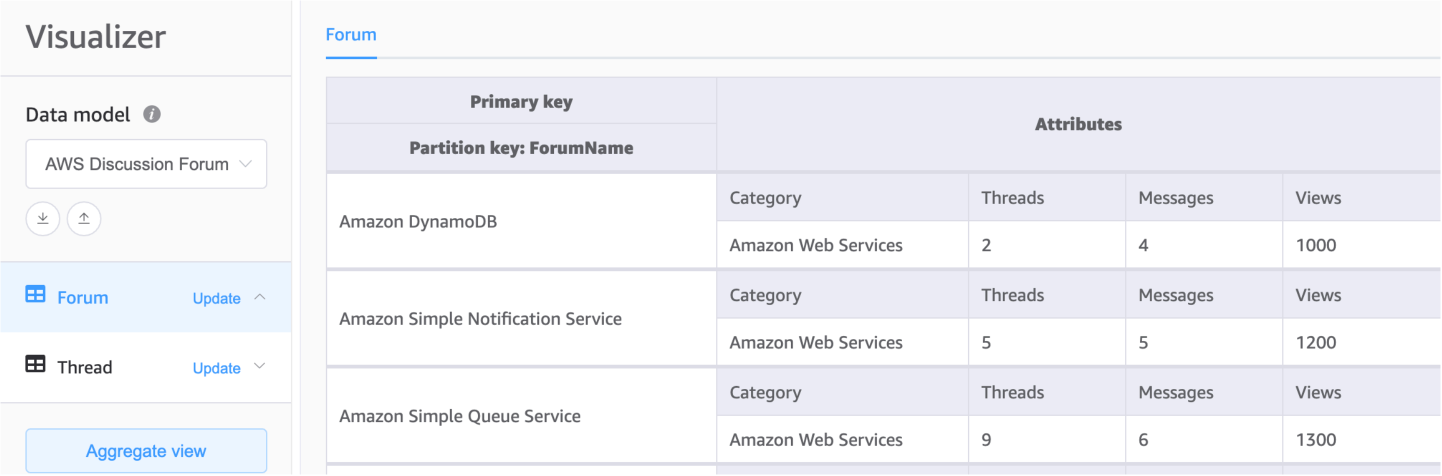
- Java の DynamoDBMapper や Enhanced Client などの上位レベルの SDK クライアントを使用する場合、同じ応答内の項目が異なるクラスに関連付けられる可能性があるため、結果の処理が難しくなりがちです。

どのようなときに使うか

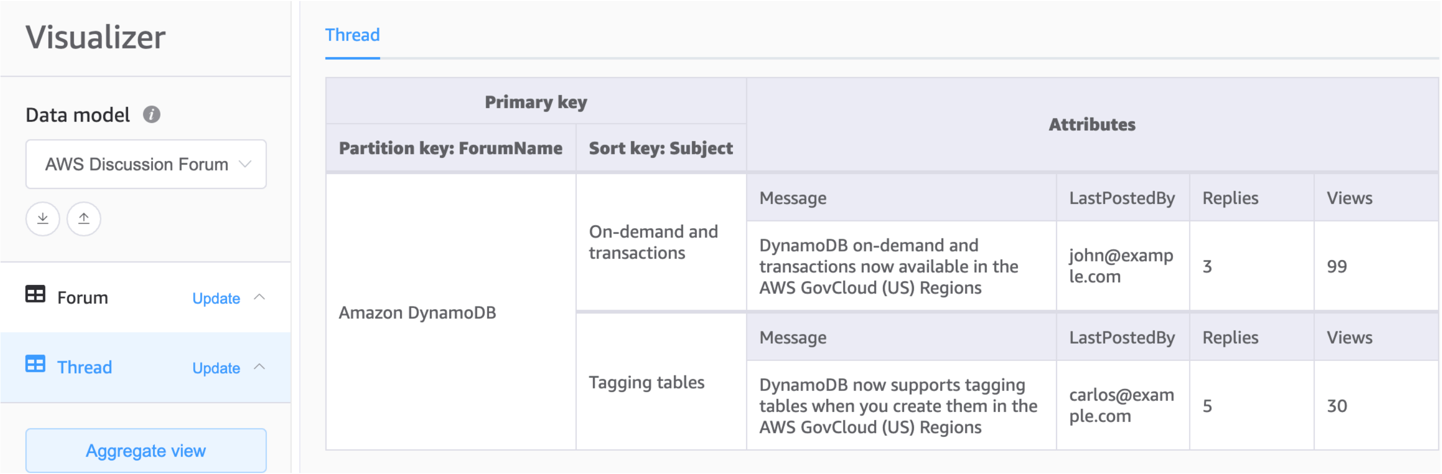
ユースケースが上記のいずれかの欠点によって大きな影響を受ける場合を除いて、シングルテーブル設計は DynamoDB の推奨設計パターンです。ほとんどのお客様にとって、このテーブル設計方法に伴う短期的な課題よりも、長期的な利点が優ります。

マルチテーブル設計の基盤

DynamoDB スキーマの別の基盤として、マルチテーブル設計を選択できます。マルチテーブル設計は、DynamoDB のテーブルごとに 1 つのタイプ (エンティティ) のデータを保存する従来のデータベース設計に近いパターンです。各テーブル内のデータは引き続きパーティションキー別に整理されるため、単一のエンティティタイプ内のパフォーマンスとスケーラビリティは最適化されますが、複数のテーブルにわたるクエリは個別に実行する必要があります。



Primary key Partition key: ForumName	Attributes			
Amazon DynamoDB	Category	Threads	Messages	Views
	Amazon Web Services	2	4	1000
Amazon Simple Notification Service	Category	Threads	Messages	Views
	Amazon Web Services	5	5	1200
Amazon Simple Queue Service	Category	Threads	Messages	Views
	Amazon Web Services	9	6	1300



Primary key		Attributes			
Partition key: ForumName	Sort key: Subject				
Amazon DynamoDB	On-demand and transactions	Message	LastPostedBy	Replies	Views
		DynamoDB on-demand and transactions now available in the AWS GovCloud (US) Regions	john@example.com	3	99
	Tagging tables	Message	LastPostedBy	Replies	Views
		DynamoDB now supports tagging tables when you create them in the AWS GovCloud (US) Regions	carlos@example.com	5	30

利点

- シングルテーブル設計に慣れていないユーザーでも簡単に設計できます。
- 各リゾルバーが単一のエンティティ (テーブル) にマッピングされるため、GraphQL リゾルバーの実装が容易になります。
- エンティティタイプ別に異なるデータ要件が許容されます。
  - ミッションクリティカルな個々のテーブルのバックアップを作成できます
  - テーブル暗号化はテーブルごとに管理できます。テナントごとに暗号化要件が異なるマルチテナントアプリケーションでは、個別のテナントテーブルにより、お客様ごとに独自の暗号化キーを使用できます。
  - 低頻度アクセスストレージクラスは、履歴データを含むテーブルでのみ有効にして、コスト削減のメリットを最大限に実現できます。詳細については、「テーブルクラス」を参照してください。
- 各テーブルには独自の変更データストリームがあり、単一のモノリシックプロセッサではなく、項目の種類ごとに専用の Lambda 関数を設計できます。

欠点

- 複数のテーブルにわたるデータを必要とするアクセスパターンの場合は、DynamoDB からの複数回の読み取りが必要になり、クライアントコードでのデータの処理/結合が必要になることがあります。
- 複数のテーブルの操作とモニタリングには、より多くの CloudWatch アラームが必要であり、各テーブルを個別にスケールする必要があります

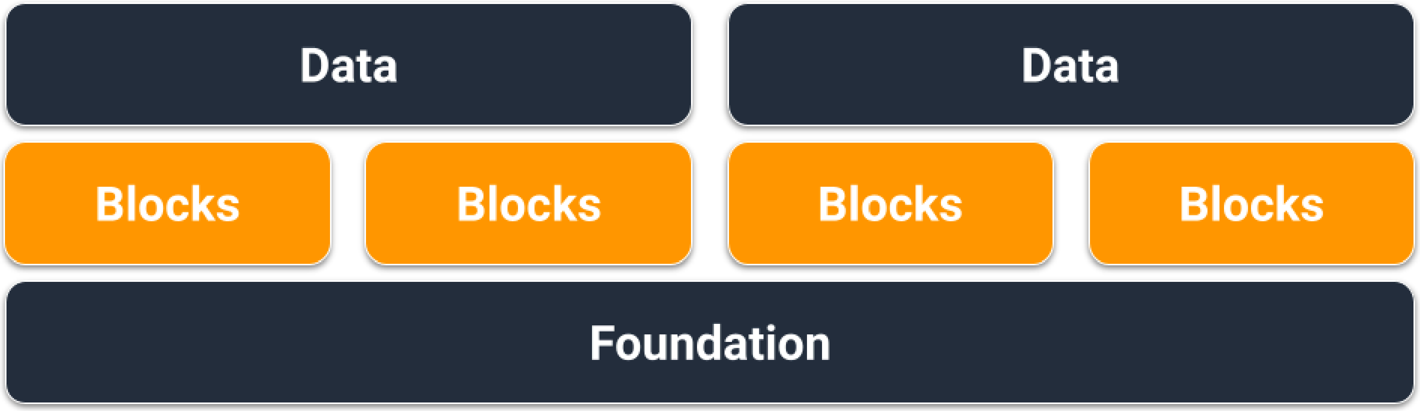
- 各テーブルのアクセス許可を個別に管理する必要があります。今後テーブルを追加する場合は、必要な IAM ロールやポリシーの変更が必要になります。

どのようなときに使うか

アプリケーションのアクセスパターンで複数のエンティティやテーブルをまとめてクエリする必要がない場合は、マルチテーブル設計が適切かつ十分なアプローチです。

DynamoDB のデータモデリングの構成要素

このセクションでは、アプリケーションで使用できる設計パターンを提供するビルディングブロックレイヤーについて説明します。



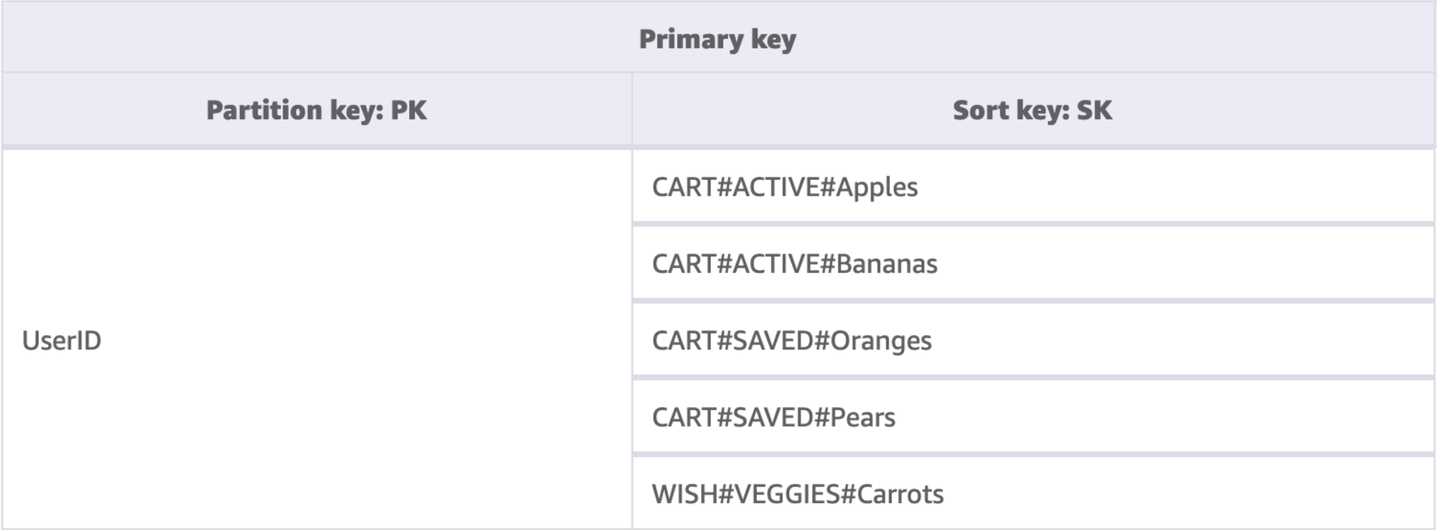
トピック

- 複合ソートキー構成要素
- マルチテナンシー構成要素
- スパースインデックスの構成要素
- Time to Live (有効期限) 構成要素
- Time to Live (アーカイブ用) 構成要素
- 垂直パーティショニング構成要素
- 書き込みシャーディング構成要素

複合ソートキー構成要素

NoSQL と言えば、非リレーショナルと考えがちです。ただし、DynamoDB の場合は、スキーマにリレーションシップを配置できない理由はなく、リレーショナルデータベースやその外部

キーと見た目が異なるだけです。DynamoDB でデータの論理階層を構築するために使用できる最も重要なパターンの 1 つは、複合ソートキーです。最も一般的な設計スタイルは、階層の各レイヤー (親レイヤー > 子レイヤー > 孫レイヤー) をハッシュタグで区切ることです。例えば、`PARENT#CHILD#GRANDCHILD#ETC` と指定します。



Primary key	
Partition key: PK	Sort key: SK
UserID	CART#ACTIVE#Apples
	CART#ACTIVE#Bananas
	CART#SAVED#Oranges
	CART#SAVED#Pears
	WISH#VEGGIES#Carrots

DynamoDB のパーティションキーには、データをクエリするために常に正確な値が必要です。一方、ソートキーには、バイナリツリーをたどる場合と同じように、左から右に部分条件を適用できます。

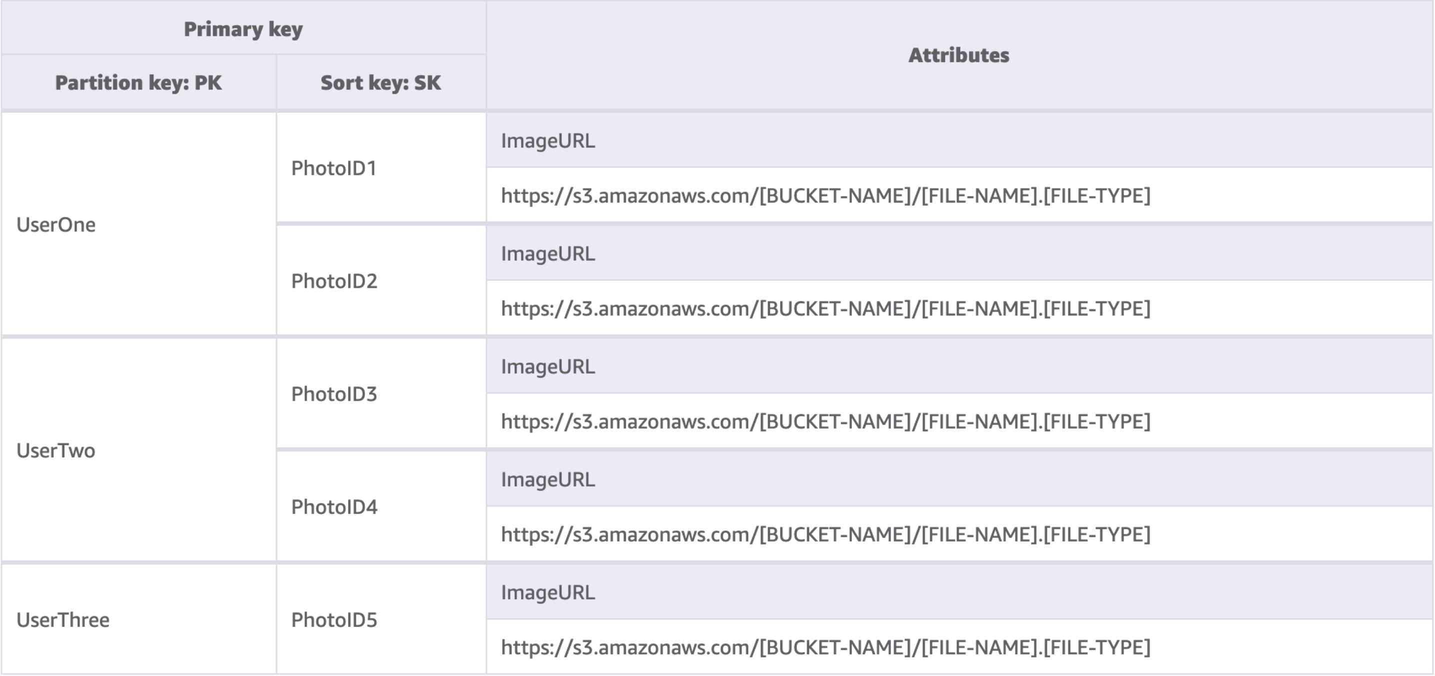
上の例では、e コマースストアのショッピングカートを、ユーザーセッションをまたいで維持する必要があります。ユーザーはログインするたびに、あとで買うために保存した商品を含むショッピングカート全体を確認したい場合があります。ただし、レジに進むときは、アクティブなカート内の商品のみをロードして購入できるようにする必要があります。これらの `KeyConditions` は、どちらもカートのソートキーを明示的に要求するため、追加のウィッシュリストのデータは読み取り時に DynamoDB によって単に無視されます。保存した商品とアクティブな商品はどちらも同じカートの一部ですが、アプリケーションの異なる部分ごとに異なる方法で扱う必要があります。この場合、アプリケーションの部分ごとに必要なデータのみを取得する最適な方法は、ソートキーのプレフィックスに `KeyCondition` を適用することです。

この構成要素の主な特徴

- 関連項目は相互にローカルに保存されるため、データへのアクセスが効率的になります。
- `KeyCondition` 式を使用すると、階層のサブセットを選択的に取得できるため、無駄な RCU がなくなります。
- アプリケーションの異なる部分ごとに異なるプレフィックスを適用して項目を保存できるため、項目の上書きや書き込みの競合を防ぐことができます。

マルチテナンシー構成要素

多くのお客様が DynamoDB を使用してマルチテナントアプリケーションのデータをホストしています。このようなシナリオでは、1 つのテナントからのすべてのデータをテーブルの独自の論理パーティションに保持するようにスキーマを設計する必要があります。これには、項目コレクションという概念を活用します。項目コレクションとは、DynamoDB テーブル内で同じパーティションキーを持つすべての項目を表す用語です。DynamoDB におけるマルチテナンシーへのアプローチの詳細については、「DynamoDB のマルチテナンシー」を参照してください。



Primary key		Attributes
Partition key: PK	Sort key: SK	
UserOne	PhotoID1	ImageURL https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID2	ImageURL https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserTwo	PhotoID3	ImageURL https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID4	ImageURL https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserThree	PhotoID5	ImageURL https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]

この例では、ユーザー数が数千人に及ぶ可能性がある写真ホスティングサイトを運営しています。各ユーザーは、最初は自分のプロファイルにのみ写真をアップロードし、デフォルトでは他のユーザーの写真を見ることはできません。各ユーザーからの API コールの承認に分離レベルを追加して、各自のパーティションのデータのみを要求していることを確認するのが理想的ですが、スキーマレベルでは一意のパーティションキーで十分です。

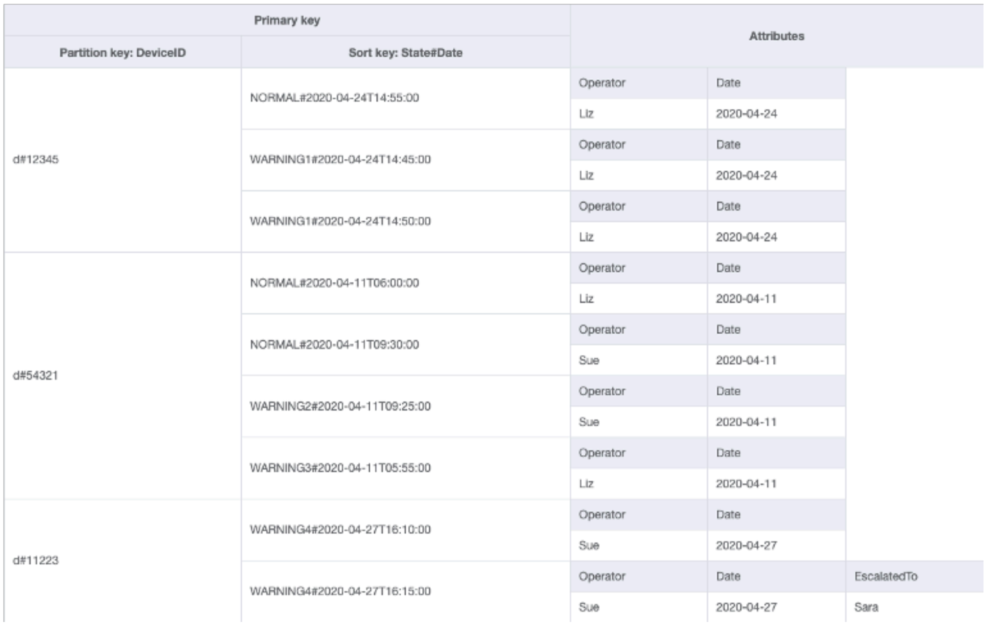
この構成要素の主な特徴

- 1 人のユーザーやテナントが読み取ることができるデータ量は、各自のパーティション内の項目の合計量と同じ量に制限されます。
- アカウントの閉鎖やコンプライアンス要求に伴うテナントのデータの削除は、適切かつ安価に行うことができます。単に、パーティションキーがテナント ID と等しいクエリを実行し、返されたプライマリキーごとに `DeleteItem` 操作を実行します。

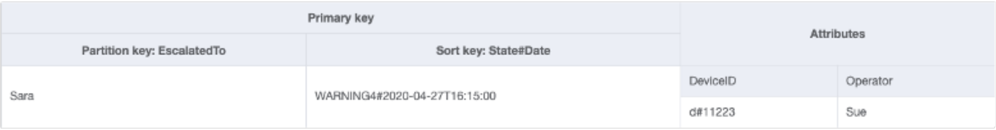
> **Note**
>
> マルチテナンシーを念頭に置いて設計されているため、1 つのテーブル全体でさまざまな暗号化キープロバイダーを使用してデータを安全に分離できます。AWSAmazon DynamoDB 用の Database Encryption SDK を使用すると、DynamoDB ワークロードにクライアント側の暗号化を組み込むことができます。属性レベルの暗号化を実行すると、特定の属性値を DynamoDB テーブルに保存する前に暗号化したり、データベース全体を事前に復号せずに暗号化された属性を検索したりできます。

スパースインデックスの構成要素

アクセスパターンによっては、まれな項目や、ステータスを受け取る項目 (応答のエスカレーションが必要) に一致する項目の検索が必要になる場合があります。これらの項目については、データセット全体を定期的にクエリするのではなく、グローバルセカンダリインデックス (GSI) を利用できます。GSI ではデータがまばらに読み込まれるという事実が役立ちます。つまり、インデックスに定義されている属性を持つベーステーブルの項目だけがインデックスにレプリケートされます。



Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara



Primary key		Attributes	
Partition key: EscalatedTo	Sort key: State#Date		
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator
		d#11223	Sue

この例は、現場の各デバイスが定期的にステータスを報告している IOT のユースケースを示しています。ほとんどのレポートでは、デバイスから何の問題もないことが報告されると予想されますが、場合によっては障害が発生し、修理技術者へのエスカレーションが必要になる場合があります。エ

スカレーションを含むレポートでは、EscalatedTo 属性が項目に追加されますが、それ以外の場合は追加されません。この例の GSI は EscalatedTo でパーティション化されており、GSI はベーステーブルからキーを引き継ぐため、どの DeviceID がいつ障害を報告したかは依然として確認できます。

DynamoDB の場合、読み取りは書き込みよりも安価であり、スパースインデックスは、特定の種類の項目のインスタンスがめったに発生しないが、これらを見つけるための読み取りをよく行うユースケースでは非常に強力なツールです。

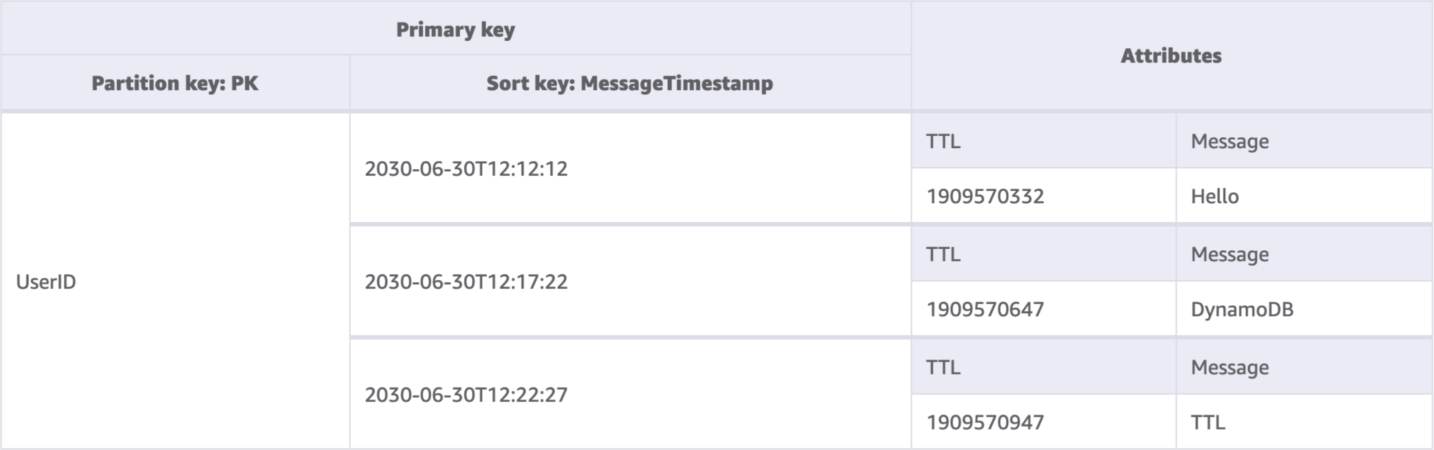
この構成要素の主な特徴

- スパース GSI の書き込みコストとストレージコストは、キーパターンに一致する項目にのみ適用されるため、GSI のコストは、すべての項目をレプリケートする他の GSI よりも大幅に低くなる可能性があります。
- 複合ソートキーを引き続き使用して、目的のクエリに一致する項目をさらに絞り込むこともできます。例えば、ソートキーにタイムスタンプを使用して、過去 X 分間 (SK > 5 minutes ago, ScanIndexForward: False) に報告された障害のみを表示できます。

Time to Live (有効期限) 構成要素

ほとんどのデータは、特定の期限まで、プライマリデータストアに保持する価値があると考えられます。DynamoDB からのデータのエージングアウトを容易にするために、Time to Live (TTL) と呼ばれる機能があります。TTL 機能を使用すると、エポックタイムスタンプ (経過済み) を持つ項目のために、モニタリングする必要がある特定の属性をテーブルレベルで定義できます。これにより、期限切れのレコードをテーブルから無料で削除できます。

> Note
>
> グローバルテーブルバージョン 2019.11.21 (現行) を使用しており、Time to Live 機能も使用している場合、DynamoDB は TTL による削除をすべてのレプリカテーブルにレプリケートします。最初の TTL による削除の場合、TTL の有効期限切れが発生したリージョンでは、書き込みキャパシティが消費されません。ただし、レプリカテーブルにレプリケートされた TTL による削除の場合、各レプリカリージョンではレプリケートされた書き込みキャパシティが消費され、該当する料金が適用されます。



Primary key		Attributes	
Partition key: PK	Sort key: MessageTimestamp		
UserID	2030-06-30T12:12:12	TTL	Message
		1909570332	Hello
	2030-06-30T12:17:22	TTL	Message
		1909570647	DynamoDB
	2030-06-30T12:22:27	TTL	Message
		1909570947	TTL

この例は、存続期間の短いメッセージをユーザーが作成できるように設計したアプリケーションを示しています。DynamoDB でメッセージを作成すると、アプリケーションコードによって TTL 属性が 7 日後の日付に設定されます。約 7 日後に、DynamoDB はこれらの項目のエポックタイムスタンプが過去のものであることを確認し、削除します。

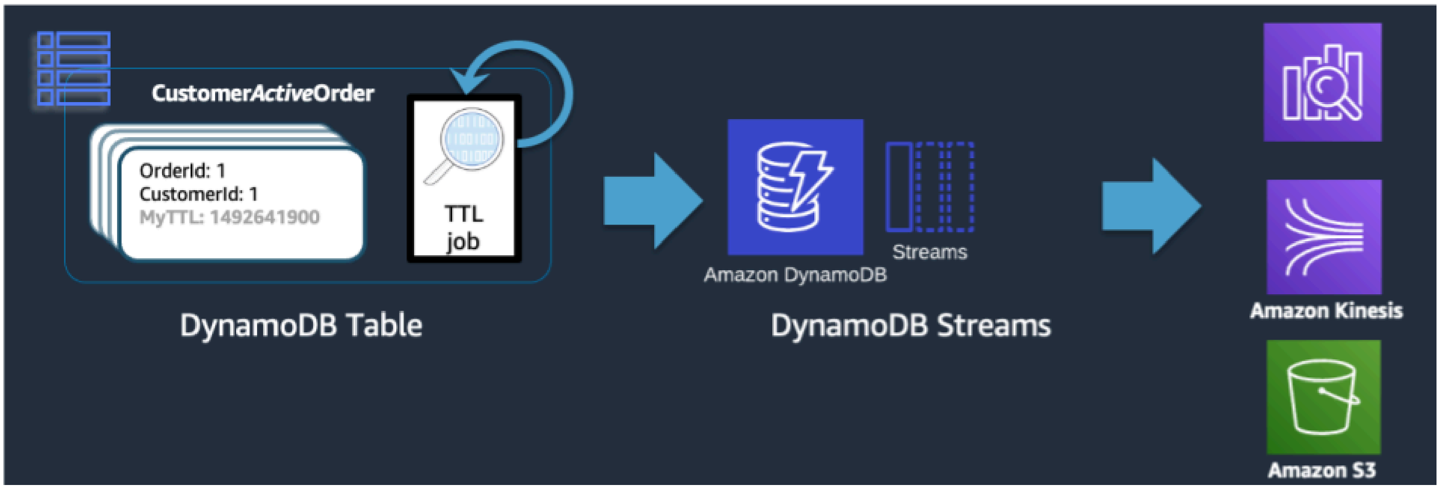
TTL による削除は無料であるため、この機能を使用してテーブルから履歴データを削除することを強くお勧めします。これにより、毎月の全体的なストレージ料金が削減され、クエリによって取得されるデータが少なくなるため、ユーザーの読み取りコストも削減される可能性があります。TTL はテーブルレベルで有効になっていますが、どの項目またはエンティティに対して TTL 属性を作成するか、またエポックタイムスタンプをどれだけ将来に設定するかはユーザーが指定できます。

この構成要素の主な特徴

- TTL による削除はバックグラウンドで実行され、テーブルのパフォーマンスには影響しません
- TTL は約 6 時間ごとに実行される非同期プロセスですが、期限切れのレコードの削除には 48 時間超かかる場合があります。
  - 古いデータを 48 時間以内にクリーンアップする必要がある場合は、ロックレコードや状態管理などのユースケースで TTL による削除に依存しないでください。
- TTL 属性には有効な属性名を指定できますが、値は数値型でなければなりません。

Time to Live (アーカイブ用) 構成要素

TTL は DynamoDB から古いデータを削除する効果的なツールですが、多くのユースケースでは、データのアーカイブをプライマリデータストアよりも長期間保存する必要があります。この場合、TTL によるレコードの時間指定削除を利用して、期限切れのレコードを長期データストアにプッシュできます。



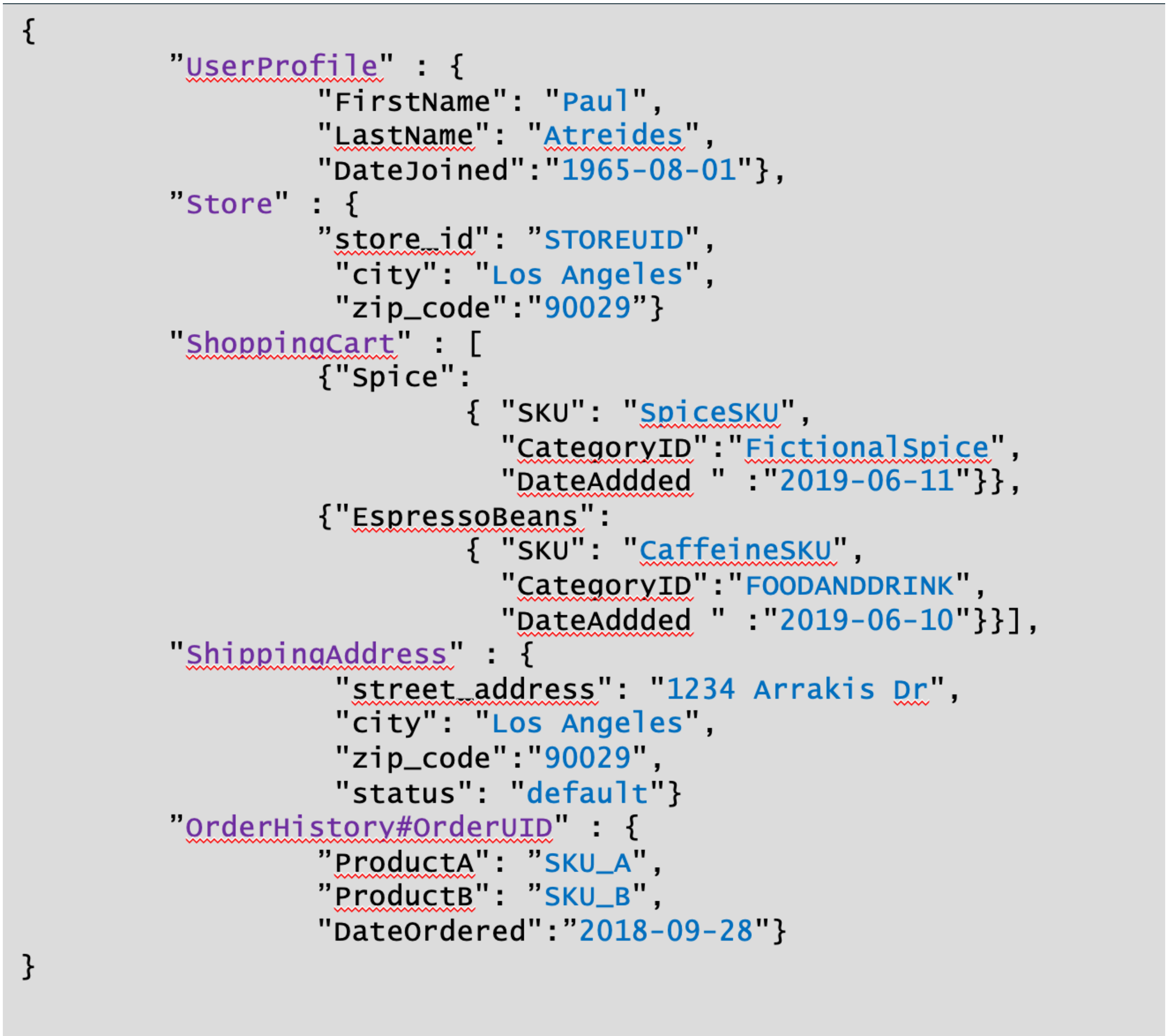
DynamoDB が TTL を使用して削除を実行すると、その操作は引き続き `Delete` イベントとして DynamoDB Stream にプッシュされます。ただし、DynamoDB の TTL 自体が削除を実行する場合は、`principal:dynamodb` のストリームレコードの属性を利用できます。DynamoDB Stream への Lambda サブスクライバーを使用すると、DynamoDB プリンシパル属性にのみイベントフィルターを適用することで、このフィルターに一致するすべてのレコードが S3 Glacier などのアーカイブストアにプッシュされるようになります。

この構成要素の主な特徴

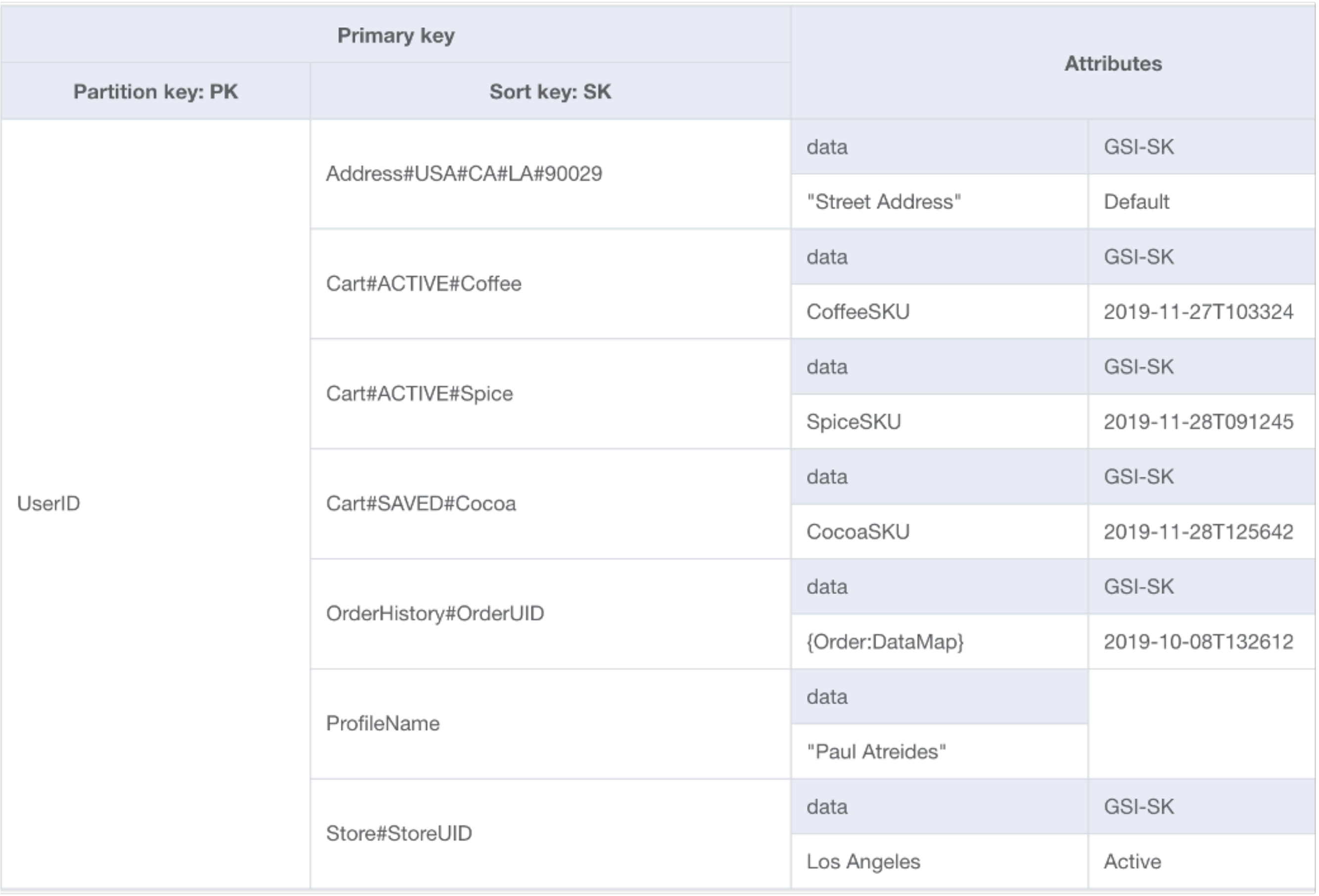
- 履歴項目に対する DynamoDB の低レイテンシーの読み取りが不要になったら、それらを S3 Glacier などのコールドストレージサービスに移行することで、ストレージコストを大幅に削減すると同時に、ユースケースのデータコンプライアンスのニーズを満たすことができます。
- データが Amazon S3 に保存されている場合、Amazon Athena や Redshift Spectrum などのコスト効率の高い分析ツールを使用してデータの履歴分析を行うことができます。

垂直パーティショニング構成要素

ドキュメントモデルデータベースに精通しているユーザーなら、すべての関連データを 1 つの JSON ドキュメントに保存するという考え方に慣れているでしょう。DynamoDB は JSON データ型をサポートしていますが、ネストされた JSON に対する `KeyConditions` の実行はサポートしていません。`KeyConditions` は、ディスクから読み取るデータの量とクエリが実際に消費する RCU の数を決定するものであるため、これに伴って大規模な非効率が生じる可能性があります。DynamoDB の書き込みと読み取りをより適切に最適化するには、ドキュメントの個々のエンティティを個別の DynamoDB 項目に分割することをお勧めします。これは垂直パーティショニングとも呼ばれます。



```
{
        ”UserProfile" : {
                "FirstName": "Paul",
                "LastName": "Atreides",
                "DateJoined":"1965-08-01"},
        ”Store" : {
                ”store_id": ”STOREUID",
                 "city": "Los Angeles",
                 "zip_code":"90029”}
        "ShoppingCart" : [
                {"Spice":
                        { "SKU": "SpiceSKU",
                          "CategoryID":"FictionalSpice",
                          "DateAddded " :"2019-06-11"}},
                {"EspressoBeans":
                        { "SKU": "CaffeineSKU",
                          "CategoryID":"FOODANDDRINK",
                          "DateAddded " :"2019-06-10"}}],
        "ShippingAddress" : {
                 "street_address": "1234 Arrakis Dr",
                 "city": "Los Angeles",
                 "zip_code":"90029",
                 "status": "default"}
        ”OrderHistory#OrderUID" : {
                ”ProductA": ”SKU_A",
                ”ProductB": ”SKU_B",
                "DateOrdered":”2018-09-28"}
}
```



Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

上の垂直パーティショニングは、シングルテーブル設計を示す重要な実例ですが、必要に応じて複数のテーブルに実装することもできます。DynamoDB は書き込みを 1 KB 単位で請求するため、理想的には各項目が 1 KB 未満になるようにドキュメントを分割します。

この構成要素の主な特徴

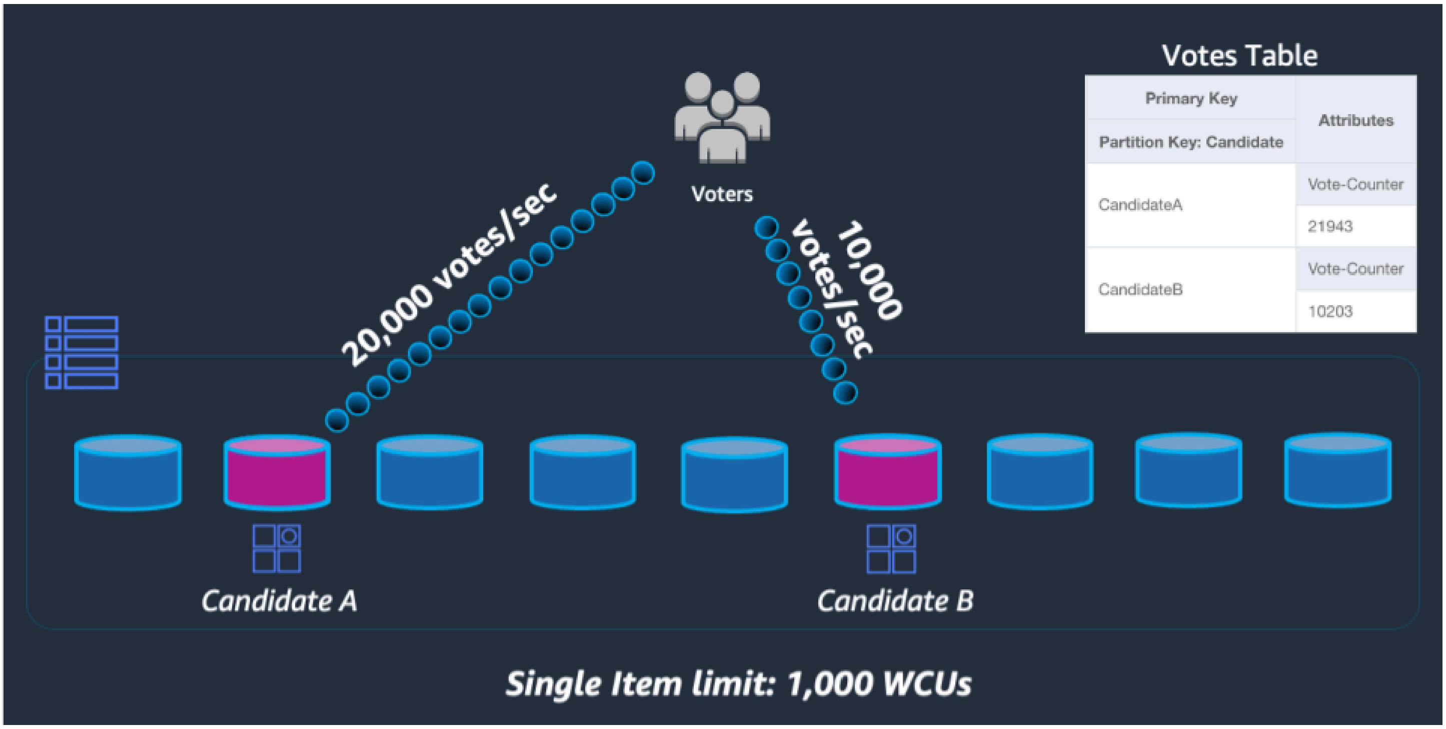
- データ関係の階層はソートキーのプレフィックスによって維持されるため、必要に応じて単一のドキュメント構造をクライアント側で再構築できます。
- データ構造の個々のコンポーネントを個別に更新できるため、小さな項目の更新はわずか 1 WCU で済みます。
- ソートキー `BeginsWith` を使用すると、アプリケーションは 1 回のクエリで類似したデータを取得し、読み取りコストを集約して総コスト/レイテンシーを削減できます。
- サイズの大きいドキュメントは、DynamoDB の個別項目のサイズ制限である 400 KB を簡単に超えてしまう可能性があり、垂直パーティショニングはこの制限を回避するのに役立ちます。

書き込みシャーディング構成要素

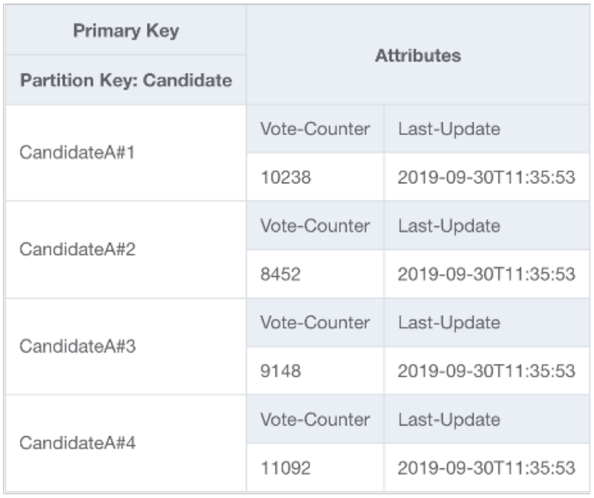
DynamoDB に設定されている数少ないハード制限の 1 つは、単一の物理パーティションが 1 秒あたりに維持できるスループットに対する制限です (必ずしも単一のパーティションキーとは限りません)。現在、これらの制限は以下のとおりです。

- 1000 WCU (または 1 秒あたり 1000 <=1KB 項目の書き込み) および 3000 RCU (または 1 秒あたり 3000 <=4KB の読み取り) の強力な整合性のある読み込み、または
- 1 秒あたり 6000 <=4KB の結果整合性のある読み込み

テーブルに対するリクエストがこれらの制限のいずれかを超えると、エラーが `ThroughputExceededException` のクライアント SDK に送り返されます。これは一般的にスロットリングと呼ばれます。この制限を超える読み取り操作を必要とするユースケースは、ほとんどの場合、DynamoDB の前に読み取りキャッシュを配置することで最適に処理されます。ただし、書き込み操作には書き込みシャーディングと呼ばれるスキーマレベルの設計が必要です。



Primary Key	Attributes
Partition Key: Candidate	
CandidateA	Vote-Counter
	21943
CandidateB	Vote-Counter
	10203



Primary Key	Attributes	
Partition Key: Candidate		
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53

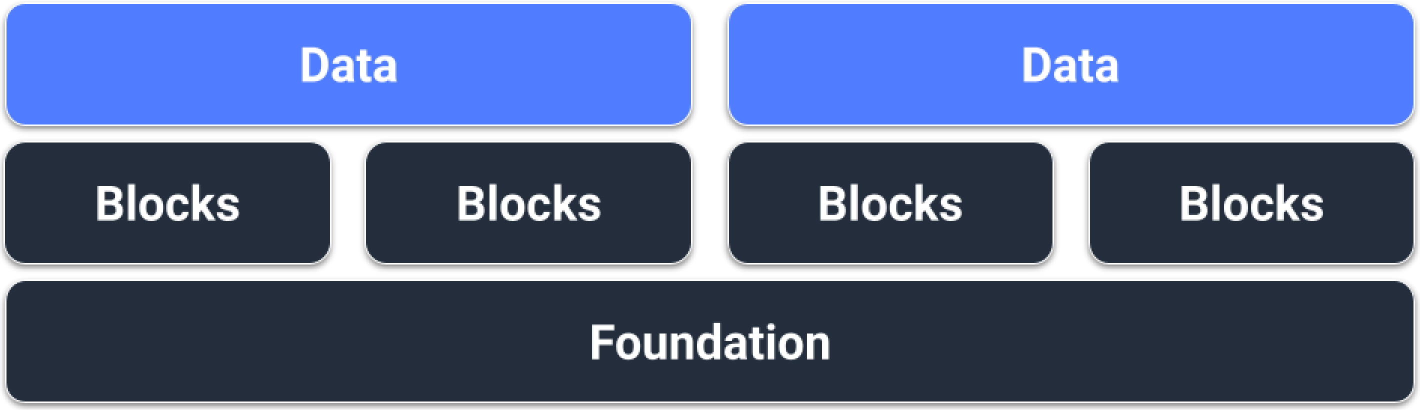
この問題を解決するには、アプリケーションの `UpdateItem` コードで各候補者のパーティションキーの末尾にランダムな整数を追加します。ランダム整数ジェネレータの範囲の上限は、特定の候補者の 1 秒あたりの予想書き込み量を 1000 で割った値と一致するか、それを超える必要があります。1 秒あたり 20,000 の投票数をサポートするには、rand(0,19) のようになります。データは個別の論理パーティションに保存されているため、読み取り時にデータを結合し直す必要があります。投票総数はリアルタイムである必要はないため、X 分ごとにすべての投票パーティションを読み取るようにスケジュールした Lambda 関数では、候補者ごとに集計をときどき行い、ライブ読み取り用の 1 つの投票集計レコードに書き戻すことができます。

この構成要素の主な特徴

- 特定のパーティションキーに対する書き込みスループットがきわめて高く、これを避けられないユースケースでは、書き込み操作を複数の DynamoDB パーティションに人為的に分散できます。
- GSI でのスロットリングはベーステーブルへの書き込み操作にバックプレッシャーを与えるため、カーディナリティの低いパーティションキーを持つ GSI でも、このパターンを利用する必要があります。

DynamoDB のデータモデリングスキーマ設計パッケージ

このセクションでは、データレイヤーについて説明し、DynamoDB テーブル設計で使用できるさまざまな例を示します。各例では、ユースケース、アクセスパターン、アクセスパターンの実現方法、スキーマの最終的な形について説明します。



前提条件

DynamoDB のスキーマを設計する前に、まずスキーマがサポートする必要のあるユースケースに関する前提条件となるデータを収集する必要があります。リレーショナルデータベースとは異なり、DynamoDB はデフォルトでシャーディングされます。つまり、データはバックグラウンドで複数のサーバーに保存されるため、データの局所性を考慮して設計することが重要です。スキーマ設計ごとに次のリストを作成する必要があります。

- エンティティのリスト (ER 図)
- 各エンティティの推定ボリュームとスループット
- サポートが必要なアクセスパターン (クエリと書き込み)
- データ保持要件

トピック

- DynamoDB でのソーシャルネットワークのスキーマ設計
- DynamoDB でのゲームプロファイルスキーマの設計
- DynamoDB の苦情管理システムのスキーマ設計
- DynamoDB での定期払いスキーマの設計
- DynamoDB でのデバイスステータス更新のモニタリング
- DynamoDB をオンラインショップのデータストアとして使用する

DynamoDB でのソーシャルネットワークのスキーマ設計

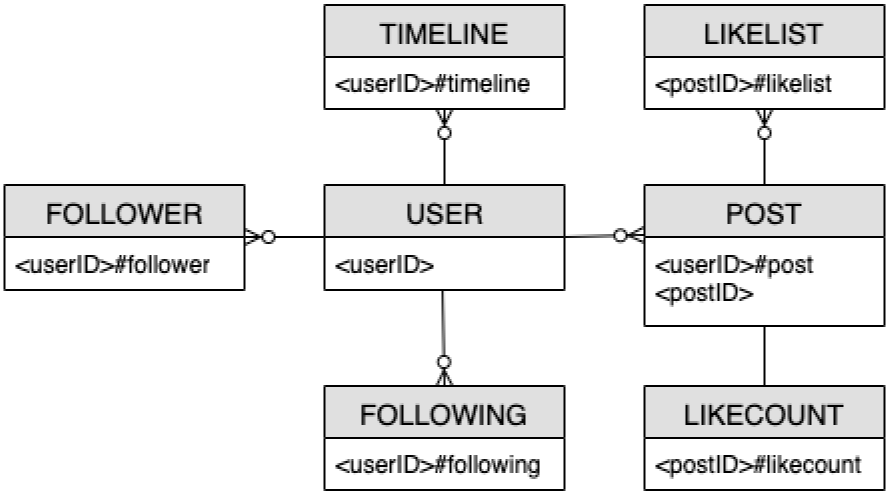
ソーシャルネットワークのビジネスユースケース

このユースケースでは、DynamoDB をソーシャルネットワークとして使用する方法について説明します。ソーシャルネットワークは、さまざまなユーザーが相互に交流できるオンラインサービスです。これから設計するソーシャルネットワークでは、ユーザーの投稿、フォロワー、フォローしている相手、およびフォローしている相手による投稿で構成されるタイムラインをユーザーが表示できるようにします。このスキーマ設計のアクセスパターンは以下のとおりです。

- 特定のユーザー ID のユーザー情報を取得する。
- 特定のユーザー ID のフォロワーリストを取得する。
- 特定のユーザー ID のフォローリストを取得する。
- 特定のユーザー ID の投稿リストを取得する。
- 特定の投稿 ID による投稿に「いいね!」したユーザーのリストを取得する。
- 特定の投稿 ID の「いいね!」数を取得する。
- 特定のユーザー ID のタイムラインを取得する。

ソーシャルネットワークエンティティ関係図

これは、ソーシャルネットワークのスキーマ設計に使用するエンティティ関係図 (ERD) です。



ソーシャルネットワークのアクセスパターン

これらは、ソーシャルネットワークのスキーマ設計のために検討するアクセスパターンです。

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

ソーシャルネットワークのスキーマ設計の進化

DynamoDB は NoSQL データベースであるため、結合 (複数のデータベースのデータを結合する操作) は実行できません。DynamoDB に慣れていないお客様は、必要がない場合でも、リレーショナルデータベース管理システム (RDBMS) の設計理念 (エンティティごとにテーブルを作成するなど) を DynamoDB に適用する可能性があります。DynamoDB のシングルテーブル設計の目的は、アプリケーションのアクセスパターンに従って事前に結合された形式でデータを書き込み、追加の計算を行わずにそのデータをすぐに使用することにあります。詳細については、「DynamoDB のシングルテーブル設計とマルチテーブル設計」を参照してください。

それでは、すべてのアクセスパターンに対処するためにスキーマ設計をどのように進化させるかをステップ別に見ていきましょう。

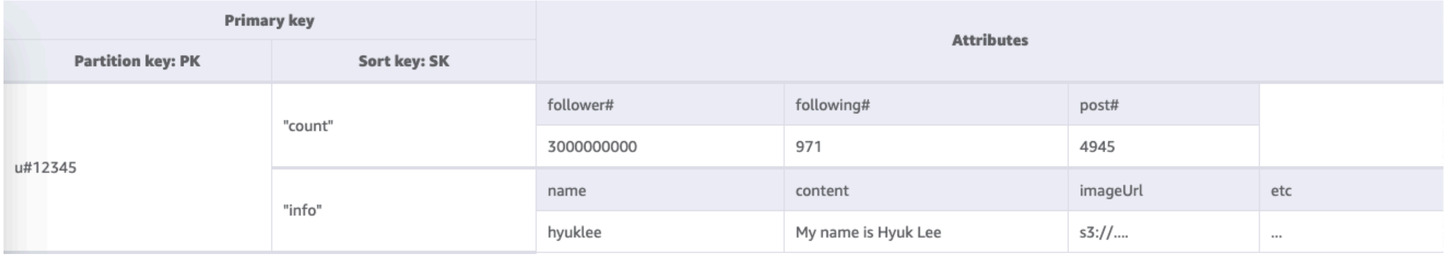
ステップ 1: アクセスパターン 1 (**`getUserInfoByUserID`**) に対処する

特定のユーザーに関する情報を取得するには、キー条件を `PK=<userID>` にしてベーステーブルに `Query` を実行する必要があります。クエリ操作では、結果をページ分割できます。これは、ユーザーに多数のフォロワーがいる場合に便利です。Query の詳細については、「DynamoDB のクエリオペレーション」を参照してください。

この例では、ユーザーの「count」と「info」の 2 種類のデータを追跡します。ユーザーの「count」は、ユーザーのフォロワー数、フォローしている相手の数、相手が作成した投稿の数を反映します。ユーザーの「info」は、名前などの個人情報を反映します。

これら 2 種類のデータは、以下の 2 つの項目で表されます。ソートキー (SK) に「count」が含まれている項目は、「info」が含まれている項目よりも変更される可能性が高くなります。DynamoDB は、更新前と更新後に表示される項目のサイズを考慮し、消費されるプロビジョニングされたスループットに、これらの項目サイズの大きい方を反映します。したがって、項目の属性の一部だけを更新した場合でも、`UpdateItem` は、プロビジョニングされたスループットの総量 (前後の項目サイズの

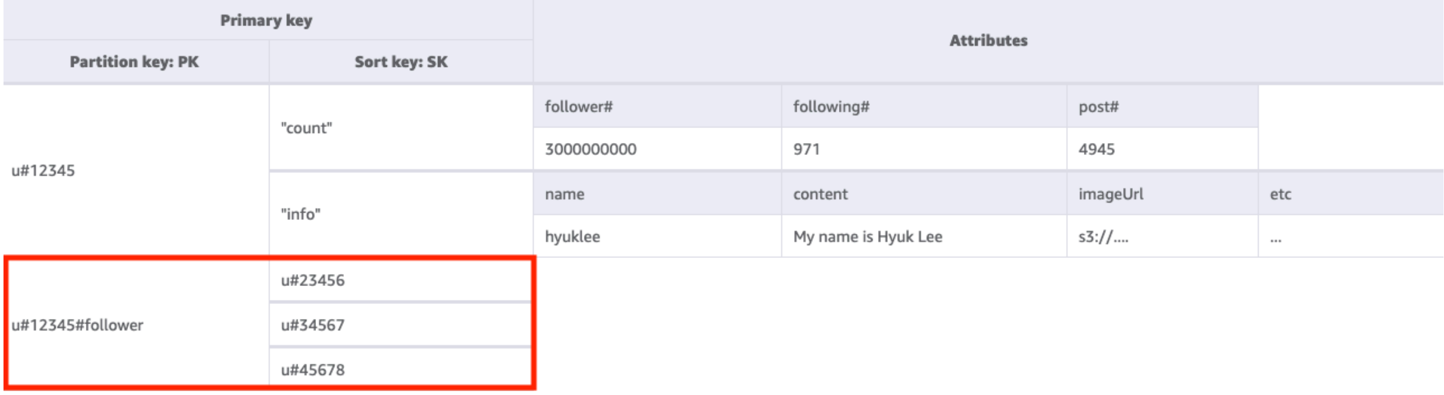
大きい方) を消費します。単一の Query 操作で項目を取得し、UpdateItem を使用して既存の数値属性の加算または減算ができます。



Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...

ステップ 2: アクセスパターン 2 (**getFollowerListByUserID**) に対処する

特定のユーザーのフォロワーリストを取得するには、キー条件を PK=<userID>#follower にしてベーステーブルに Query を実行する必要があります。

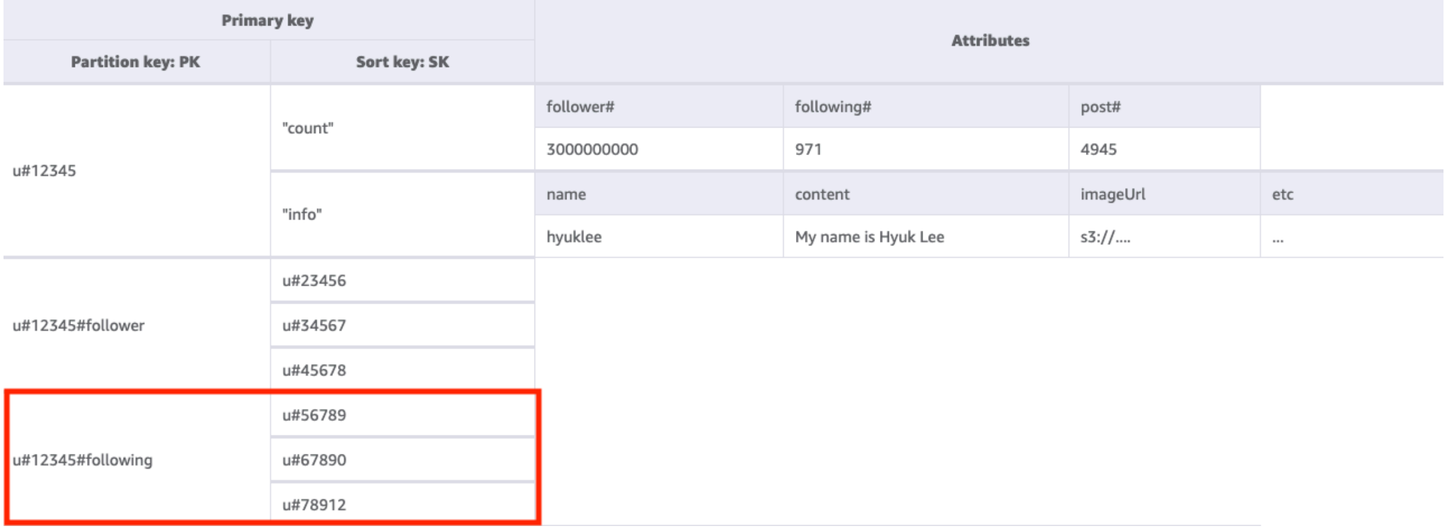


Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				

ステップ 3: アクセスパターン 3 (**getFollowingListByUserID**) に対処する

特定のユーザーがフォローしている相手のリストを取得するには、キー条件を PK=<userID>#following にしてベーステーブルに Query を実行する必要があります。次に、TransactWriteItems 操作を使用して複数のリクエストをグループ化し、以下のことを実行できます。

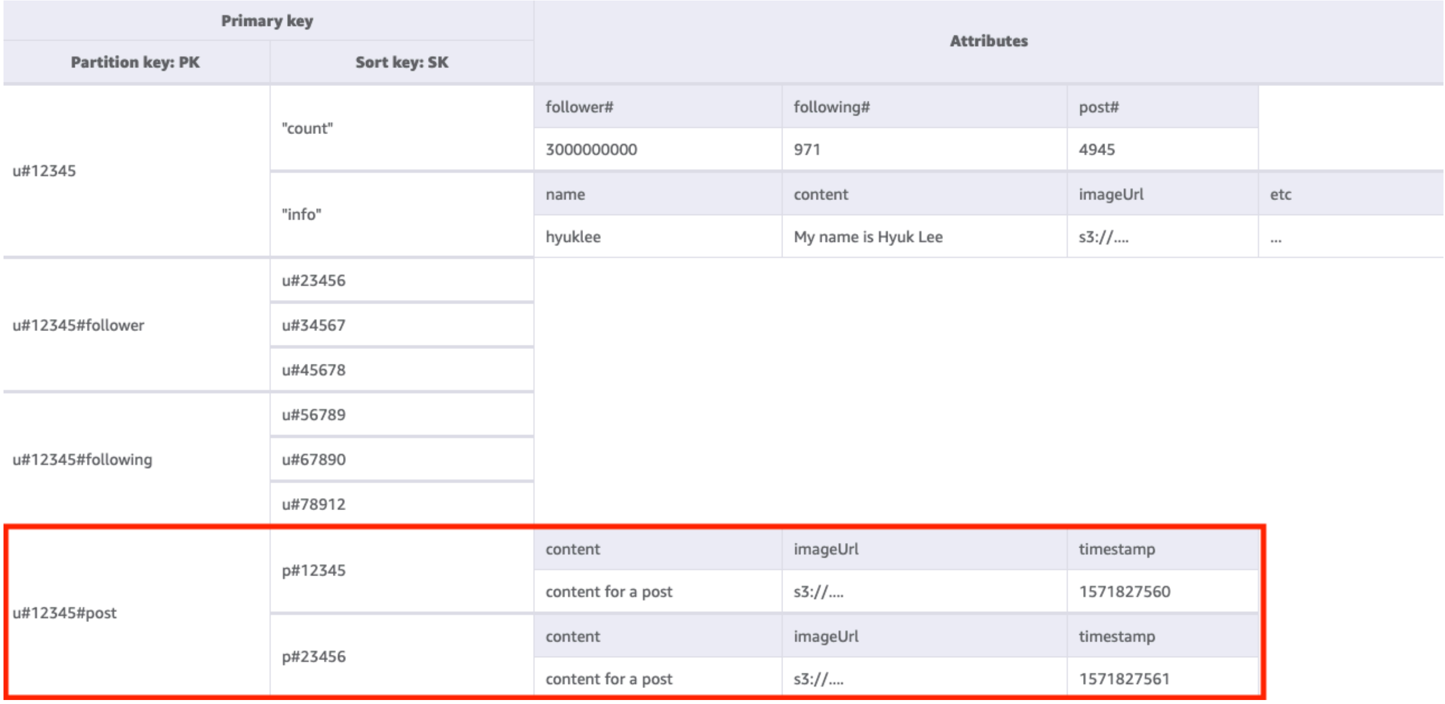
- ユーザー A をユーザー B のフォロワーリストに追加し、ユーザー B のフォロワー数を 1 つ増やします。
- ユーザー B をユーザー A のフォロワーリストに追加し、ユーザー A のフォロワー数を 1 つ増やします



Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				

ステップ 4: アクセスパターン 4 (`getPostListByUserID`) に対処する

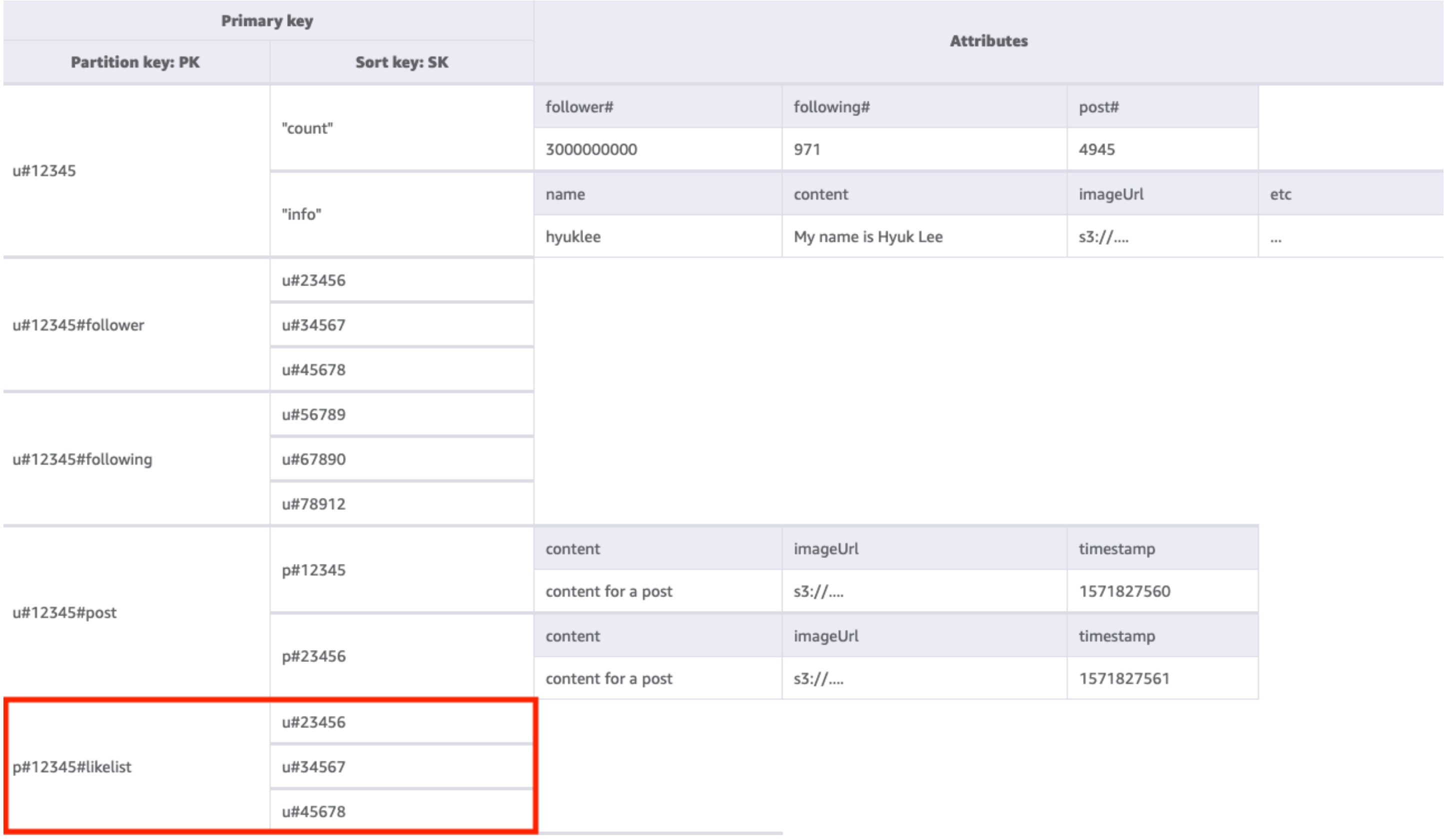
特定のユーザーが作成した投稿のリストを取得するには、キー条件を `PK=<userID>#post` にしてベーステーブルに `Query` を実行する必要があります。ここで注意すべき重要な点の 1 つは、ユーザーの postID はインクリメンタルでなければならないということです。つまり、2 番目の PostID 値は 1 番目の PostID 値よりも大きくなければなりません (ユーザーは自分の投稿を並べ替えて表示したいはずです)。これを行うには、Universally Unique Lexicographically Sortable Identifier (ULID) などの時間値に基づいて postID を生成できます。



Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	

ステップ 5: アクセスパターン 5 (`getUserLikesByPostID`) に対処する

特定のユーザーの投稿に「いいね!」を付けた人のリストを取得するには、キー条件を PK=<postID>#likelist にしてベーステーブルに Query を実行する必要があります。このアプローチは、アクセスパターン 2 (getFollowerListByUserID) とアクセスパターン 3 (getFollowingListByUserID) でフォロワーリストとフォローしている相手のリストを取得するときに使用したのと同じパターンです。

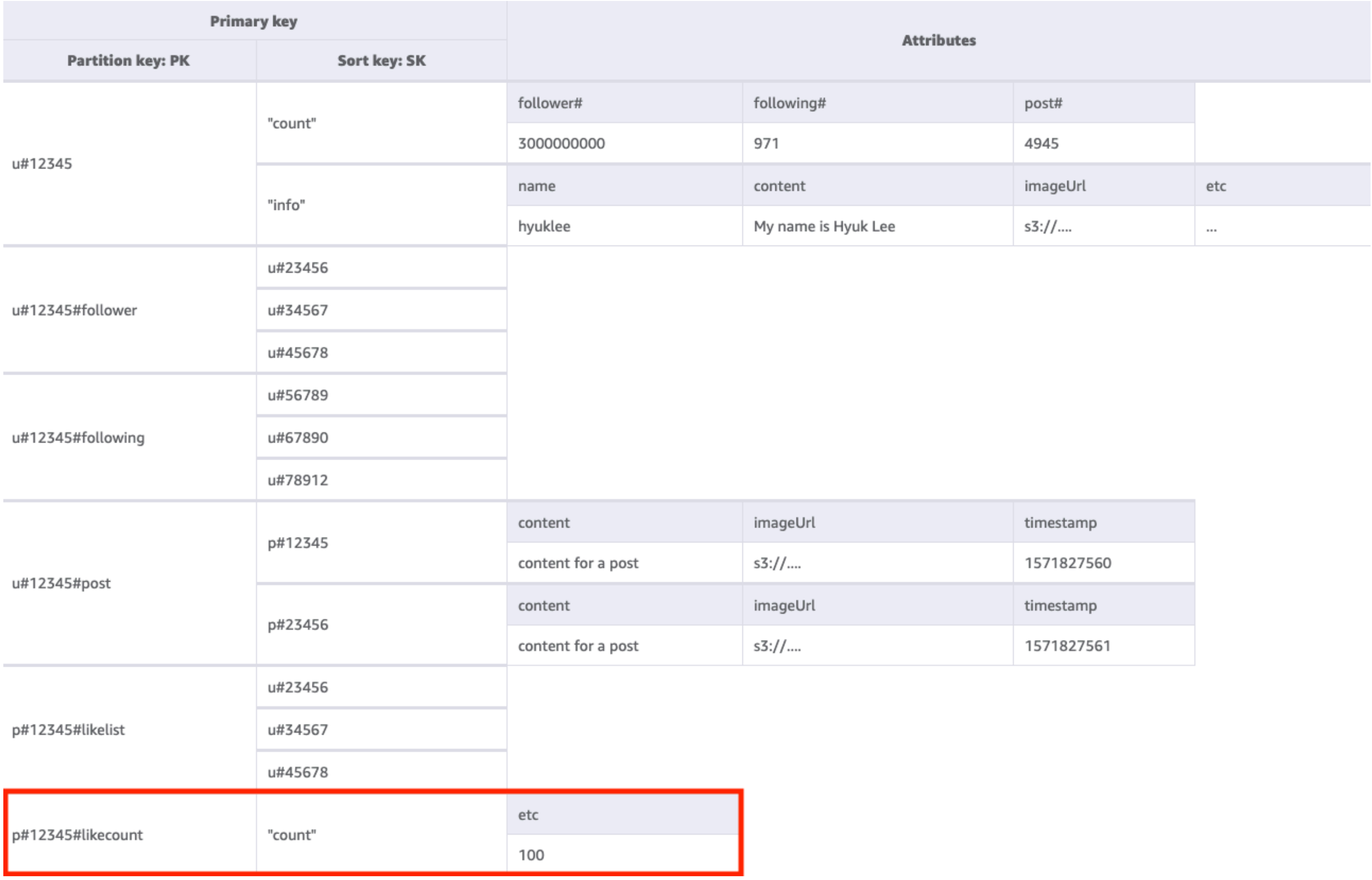


Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				

ステップ 6: アクセスパターン 6 (**getLikeCountByPostID**) に対処する

特定の投稿の「いいね!」の数を取得するには、キー条件を PK=<postID>#likecount にしてベーステーブルに GetItem 操作を実行する必要があります。このアクセスパターンでは、パーティションのスループットが 1 秒あたり 1000 WCU を超えるとスロットリングが発生するため、フォロワーの多いユーザー (有名人など) が投稿を作成するたびにスロットリングの問題が発生する可能性があります。この問題は DynamoDB が原因ではなく、DynamoDB がソフトウェアスタックの最後にあるため、DynamoDB に現れるだけです。

すべてのユーザーに「いいね!」の数を同時に表示することが本当に必要か、それとも時間の経過と共に徐々に表示できるかを評価する必要があります。一般的に、投稿の「いいね!」の数はすぐに 100% 正確である必要はありません。この戦略を実装するには、アプリケーションと DynamoDB の間にキューを置き、更新が定期的に行われるようにします。

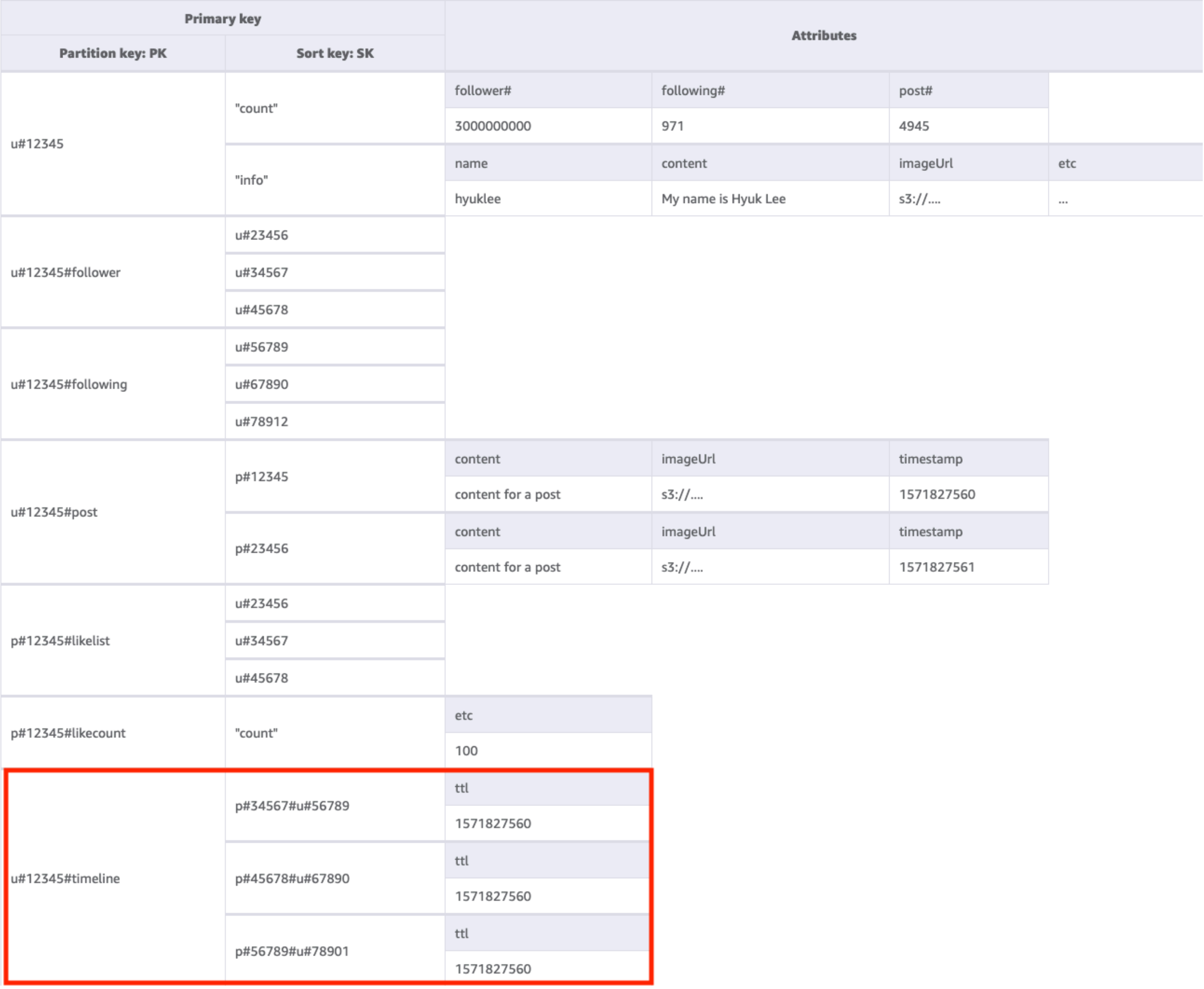


Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			

ステップ 7: アクセスパターン 7 (**`getTimelineByUserID`**) に対処する

特定のユーザーのタイムラインを取得するには、キー条件を `PK=<userID>#timeline` にしてベーステーブルに `Query` 操作を実行する必要があります。ユーザーのフォロワーが自分の投稿を同期的に表示する必要があるシナリオを考えてみましょう。ユーザーが投稿を書くたびに、フォロワーリストが読み取られ、useIDと postID がすべてのフォロワーのタイムラインキーにゆっくりと入力されます。次に、アプリケーションを起動すると、`Query` 操作でタイムラインキーを読み取り、新しい項目に対して `BatchGetItem` 操作を使用して userID と postID の組み合わせをタイムライン画面に入力できます。API コールでタイムラインを読み取ることはできませんが、投稿が頻繁に編集される可能性がある場合、これはより費用対効果の高いソリューションです。

タイムラインは最近の投稿を表示する場所なので、古い投稿をクリーンアップする方法が必要です。WCU を使用して削除する代わりに、DynamoDB の TTL 機能を使用して無料で削除できます。



Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

すべてのアクセスパターンと各アクセスパターンにスキーマ設計で対処する方法を次の表にまとめています。

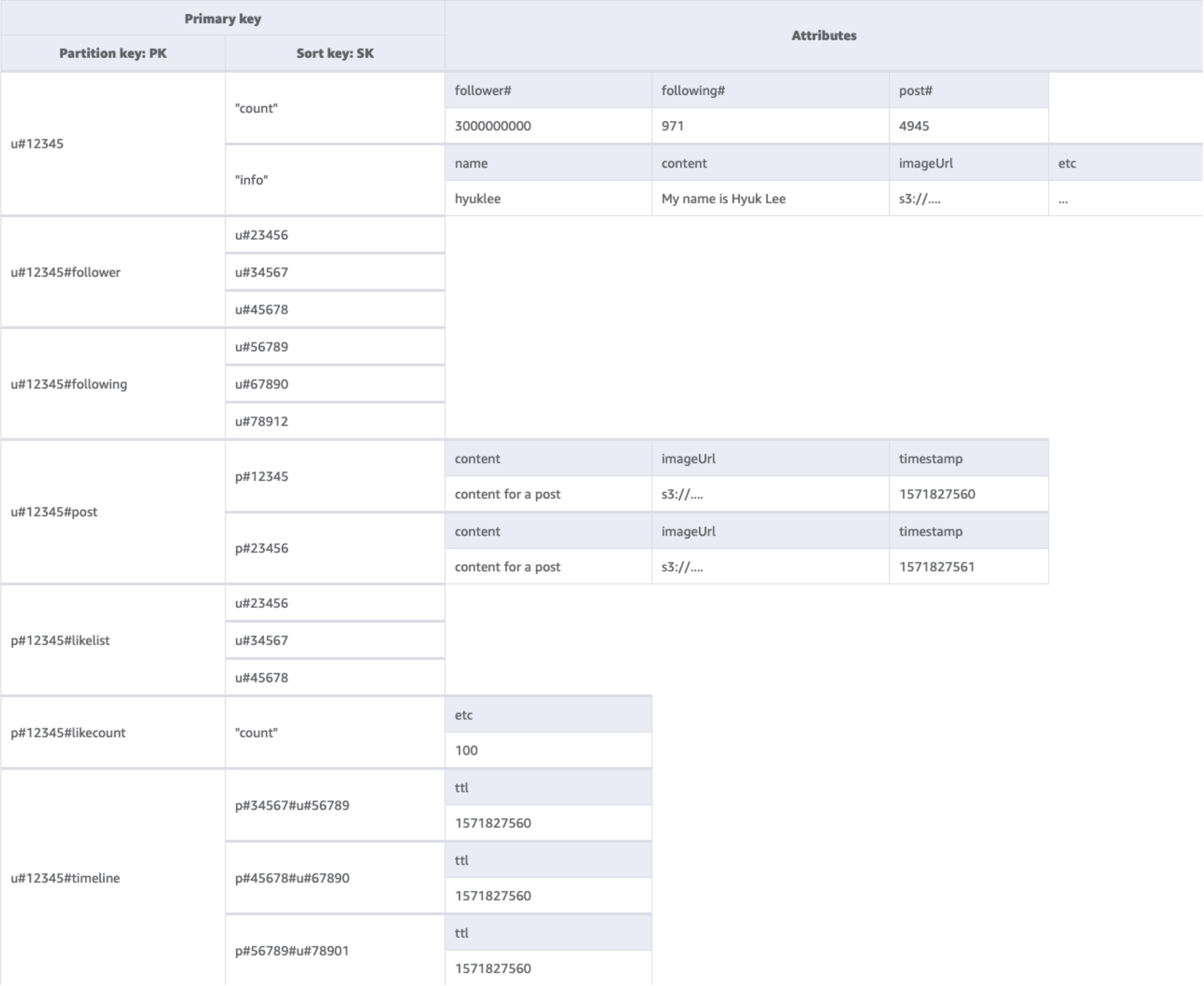
アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
getUserInfoByUserID	ベーステーブル	Query	PK=<userID>		

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
getFollowerListByUserID	ベーステーブル	Query	PK=<userID>#follower		
getFollowingListByUserID	ベーステーブル	Query	PK=<userID>#following		
getPostListByUserID	ベーステーブル	Query	PK=<userID>#post		
getUserLikesByPostID	ベーステーブル	Query	PK=<postID>#likelist		
getLikeCountByPostID	ベーステーブル	GetItem	PK=<postID>#likecount		
getTimelineByUserID	ベーステーブル	Query	PK=<userID>#timeline		

ソーシャルネットワークの最終スキーマ

これが最終的なスキーマ設計です。このスキーマ設計を JSON ファイルとしてダウンロードするには、GitHub の DynamoDB の例を参照してください。

ベーステーブル:



Primary key		Attributes			
Partition key: PK	Sort key: SK				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

このスキーマ設計での NoSQL Workbench の使用

この最終スキーマを、DynamoDB のデータモデリング、データ視覚化、クエリ開発機能を提供するビジュアルツールである NoSQL Workbench にインポートして、新しいプロジェクトを詳しく調べたり編集したりできます。使用を開始するには、次の手順に従います。

1. NoSQL Workbench をダウンロードします。詳細については、「the section called “ダウンロード”」を参照してください。
2. 上記の JSON スキーマファイルをダウンロードします。このファイルは既に NoSQL Workbench モデル形式になっています。

3. JSON スキーマファイルを NoSQL Workbench にインポートします。詳細については、「the section called “既存のモデルのインポート”」を参照してください。
4. NOSQL Workbench にインポートしたら、データモデルを編集できます。詳細については、「the section called “既存モデルの編集”」を参照してください。
5. データモデルの視覚化、サンプルデータの追加、CSV ファイルからのサンプルデータのインポートを行うには、NoSQL Workbench のデータビジュアライザー機能を使用します。

DynamoDB でのゲームプロファイルスキーマの設計

ゲームプロファイルのビジネスユースケース

このユースケースでは、DynamoDB を使用してゲームシステムのプレイヤープロファイルを保存する方法について説明します。ユーザー (この場合はプレイヤー) は、多くの最新のゲーム、特にオンラインゲームを操作する前に、プロファイルを作成する必要があります。ゲームプロファイルには通常、以下が含まれます。

- ユーザー名などの基本情報
- 項目や装備などのゲームデータ
- タスクやアクティビティなどのゲーム記録
- 友達リストなどのソーシャル情報

このアプリケーションのきめ細かなデータクエリアクセス要件を満たすため、プライマリキー (パーティションキーとソートキー) には汎用名 (PK と SK) を使用するため、以下に示すように、さまざまなタイプの値でオーバーロードできます。

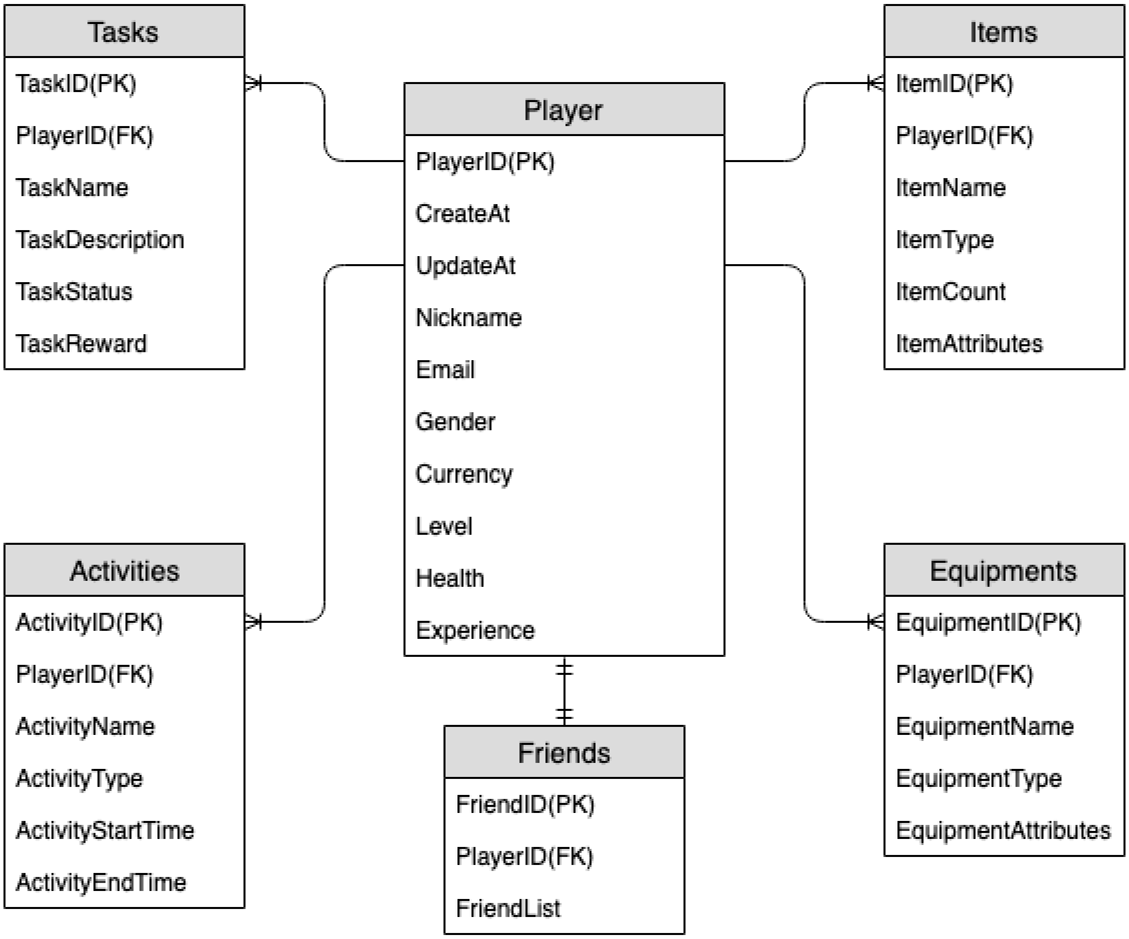
このスキーマ設計のアクセスパターンは以下のとおりです。

- ユーザーの友達リストを取得する。
- プレイヤーのすべての情報を取得する。
- ユーザーの項目リストを取得する。
- ユーザーの項目リストから特定の項目を取得する。
- ユーザーのキャラクターを更新する。
- ユーザーの項目数を更新する。

ゲームプロファイルのサイズは、ゲームによって異なります。大きな属性値は圧縮することで、DynamoDB の項目制限内に収め、コストを削減できます。スループット管理戦略は、プレイヤー数、1 秒あたりにプレイされるゲーム数、ワークロードの季節性などのさまざまな要因によって異なります。新しくリリースされたゲームの場合、通常、プレイヤーの数や人気度は不明であるため、オンデマンドスループットモードから始めます。

ゲームプロファイルエンティティ関係図

次に示すのは、ゲームプロファイルのスキーマ設計に使用するエンティティ関係図 (ERD) です。



ゲームプロファイルのアクセスパターン

これらは、ソーシャルネットワークのスキーマ設計のために検討するアクセスパターンです。

- `getPlayerFriends`
- `getPlayerAllProfile`
- `getPlayerAllItems`
- `getPlayerSpecificItem`

- updateCharacterAttributes
- updateItemCount

ゲームプロファイルスキーマ設計の進化

上記の ERD から、これは 1 対多リレーションシップタイプのデータモデリングであることがわかります。DynamoDB では、1 対多のデータモデルを項目コレクションに整理できます。これは、複数のテーブルを作成して外部キーでリンクする従来のリレーショナルデータベースとは異なります。項目コレクションは、同じパーティションキー値を共有していてもソートキー値が異なる項目のグループです。項目コレクション内の各項目には、他の項目と区別する固有のソートキー値があります。これを念頭に置いて、エンティティタイプごとに HASH 値と RANGE 値のパターンを使用してみましょう。

まず、PK や SK などの汎用名を使用して、さまざまなタイプのエンティティを同じテーブルに保存し、モデルを将来にわたって使用できるようにします。読みやすくするために、データのタイプを示すプレフィックスを含めたり、Entity_type または Type という任意の属性を含めたりできます。現在の例では、player で始まる文字列を PK として player_ID を保存し、SK のプレフィックスとして entity name# を使用します。さらに、このデータがどのエンティティタイプであるかを示す Type 属性を追加します。これにより、将来的にはより多くのエンティティタイプの保存をサポートできるようになり、GSI オーバーローディングやスパース GSI などの高度なテクノロジーを使用して、より多くのアクセスパターンに対応できるようになります。

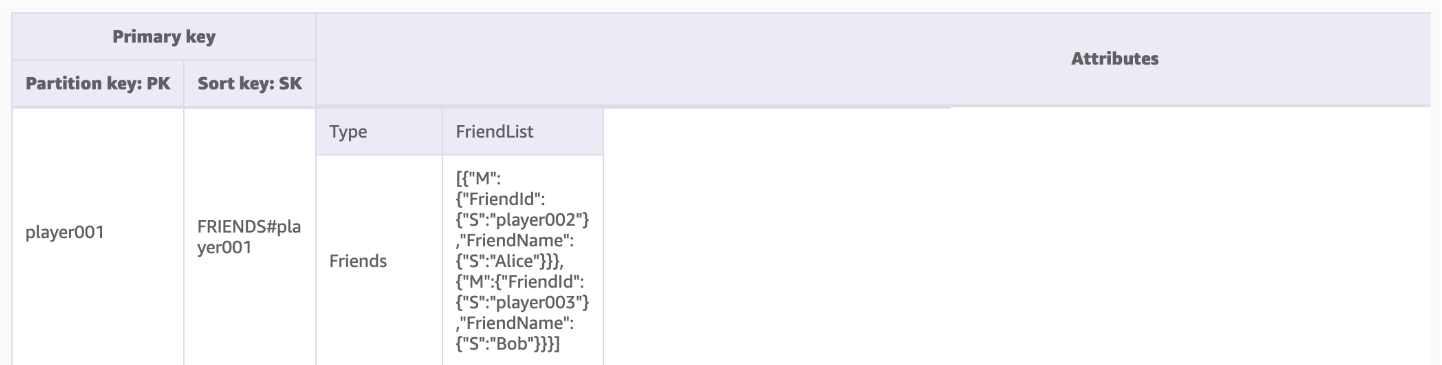
アクセスパターンの実装を始めましょう。プレイヤーの追加や装備の追加などのアクセスパターンは PutItem 操作を通じて実現できるため、無視して構いません。このドキュメントでは、上記の一般的なアクセスパターンに焦点を当てます。

ステップ 1: アクセスパターン 1 (**getPlayerFriends**) に対処する

このステップではアクセスパターン 1 (getPlayerFriends) に対処します。現在の設計の場合、友達の関係はシンプルで、ゲーム内の友達の数は少なくなっています。わかりやすくするために、リストデータ型を使用して友達リストを保存します (1:1 モデリング)。この設計では、GetItem を使用してこのアクセスパターンに対処します。GetItem 操作では、特定の項目を取得するためのパーティションキーとソートキーの値を明示的に指定します。

ただし、ゲームに多数の友達がいて、友達間の関係が複雑な場合 (招待コンポーネントと承諾コンポーネントの両方で友達の関係が双方向であるなど)、友達リストのサイズを無制限にスケールするには、多対多リレーションシップを使用して各友達を個別に保存する必要があります。また、友達関係の変更に伴って複数の項目を同時に操作する必要がある場合は、DynamoDB トランザクションを

使用して複数のアクションをグループ化し、単一の全部かゼロかの TransactWriteItems 操作または TransactGetItems 操作として送信できます。



Primary key		Attributes	
Partition key: PK	Sort key: SK		
player001	FRIENDS#player001	Type	FriendList
		Friends	[{"M":{"FriendId":{"S":"player002"},"FriendName":{"S":"Alice"}}},{"M":{"FriendId":{"S":"player003"},"FriendName":{"S":"Bob"}}}]

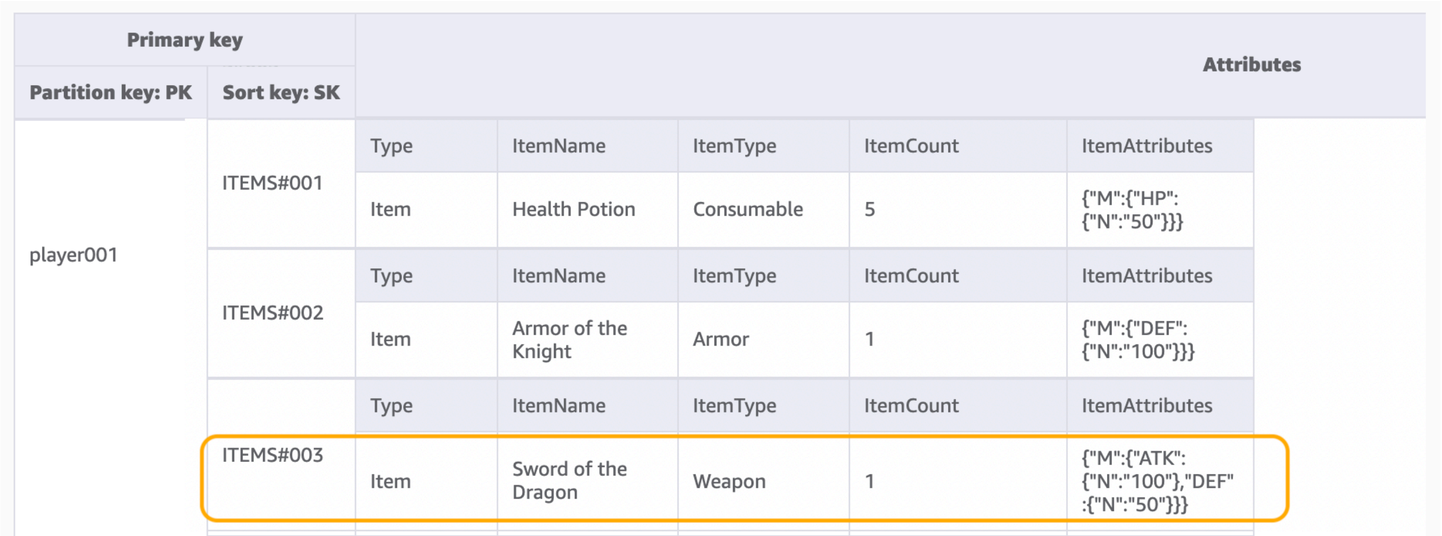
ステップ 2: アクセスパターン 2 (**getPlayerAllProfile**),3 (**getPlayerAllItems**)、4 (**getPlayerSpecificItem**) に対処する

このステップを使用して、アクセスパターン 2 (getPlayerAllProfile)、3 (getPlayerAllItems)、4 (getPlayerSpecificItem) に対処します。これら 3 つのアクセスパターンに共通しているのは、Query 操作を使用する範囲クエリです。クエリの範囲によっては、実際の開発でよく使用されるキー条件およびフィルター式を使用します。

Query 操作では、パーティションキーとして 1 つの値を指定し、このパーティションキー値を持つすべての項目を取得します。このようにしてアクセスパターン 2 (getPlayerAllProfile) を実装します。オプションで、ソートキー条件式、つまりテーブルから読み取る項目を決定する文字列を追加できます。アクセスパターン 3 (getPlayerAllItems) は、ソートキー begins_with ITEMS# というキー条件を追加することで実装します。さらに、アプリケーション側の開発を簡略化するために、フィルター式を使用してアクセスパターン 4 (getPlayerSpecificItem) を実装できます。

Weapon カテゴリの項目をフィルタリングするフィルター式を使用した疑似コードの例を次に示します。

```
filterExpression: "ItemType = :itemType"
expressionAttributeValues: {":itemType": "Weapon"}
```



Primary key		Attributes				
Partition key: PK	Sort key: SK					
player001	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}
		Type	ItemName	ItemType	ItemCount	ItemAttributes
	ITEMS#003	Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}

Note

フィルター式は、クエリの完了後、結果がクライアントに返される前に適用されます。したがって、Query は、フィルター式の有無にかかわらず、同じ量の読み取りキャパシティを消費します。

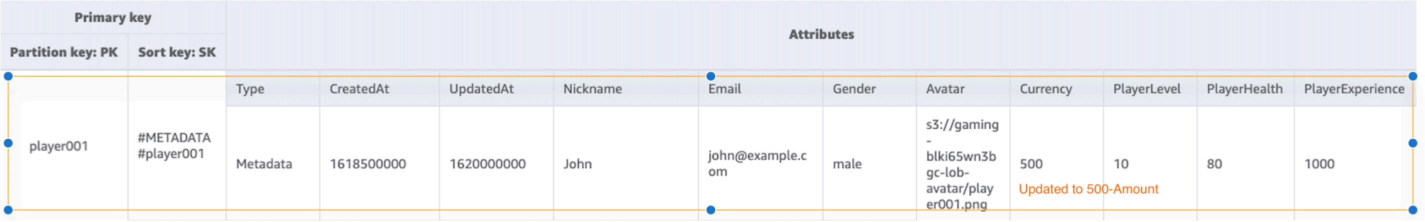
アクセスパターンとして、大きなデータセットをクエリし、大量のデータをフィルタリングして一部のデータのみを保持する場合、適切なアプローチは DynamoDB パーティションキーとソートキーをより効果的に設計することです。例えば、前述の特定の ItemType を取得する例で、各プレイヤーに多数の項目があり、特定の ItemType をクエリするのが一般的なアクセスパターンである場合は、ItemType を複合キーとして SK に取り込む方が効率的です。データモデルは ITEMS#ItemType#ItemId のようになります。

ステップ 3: アクセスパターン 5 (**updateCharacterAttributes**) および 6 (**updateItemCount**) に対処する

このステップを使用して、アクセスパターン 5 (updateCharacterAttributes) と 6 (updateItemCount) に対処します。プレイヤーが通貨を減らしたり、項目内の特定の武器の数量を変更したりするなど、キャラクターを変更する必要がある場合は、UpdateItem を使用してこれらのアクセスパターンを実装します。プレイヤーの通貨を更新しながら最低額を下回らないようにするには、the section called “条件式” を追加して残高が最低額以上である場合にのみ残高を減らすことができます。擬似コードの例を次に示します。

```
UpdateExpression: "SET currency = currency - :amount"
```

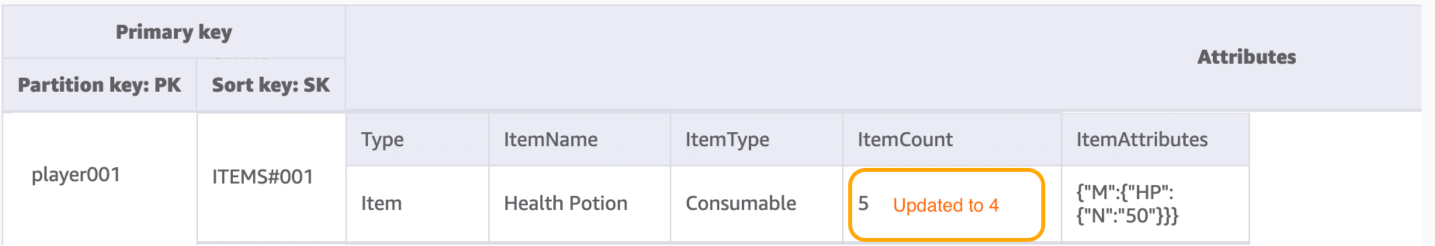
```
ConditionExpression: "currency >= :minAmount"
```



Primary key		Attributes										
Partition key: PK	Sort key: SK											
		Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
player001	#METADATA #player001	Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500 Updated to 500-Amount	10	80	1000

DynamoDB で開発し、アトミックカウンタを使用してインベントリを減らす場合、楽観的ロックを使用することで冪等性を確保できます。アトミックカウンタの擬似コードの例を次に示します。

```
UpdateExpression: "SET ItemCount = ItemCount - :incr"
expression-attribute-values: '{":incr":{"N":"1"}}'
```



Primary key		Attributes				
Partition key: PK	Sort key: SK					
		Type	ItemName	ItemType	ItemCount	ItemAttributes
player001	ITEMS#001	Item	Health Potion	Consumable	5 Updated to 4	{"M":{"HP":{"N":"50"}}}

さらに、プレイヤーが通貨で項目を購入するシナリオでは、プロセス全体で通貨の差し引きと項目の追加を同時に行う必要があります。DynamoDB トランザクションを使用すると、複数のアクションをグループ化し、単一の全部かゼロかの `TransactWriteItems` 操作または `TransactGetItems` 操作として送信できます。`TransactWriteItems` は、単一の全部かゼロかの操作で最大 100 の書き込みアクションをグループ化する、同期かつ冪等性の書き込み操作です。アクションはアトミックに完了するため、すべてが成功するか、どれも成功しません。トランザクションは、通貨の重複や消滅のリスクを排除するのに役立ちます。トランザクションの詳細については、「DynamoDB トランザクションの例」を参照してください。

すべてのアクセスパターンと各アクセスパターンにスキーマ設計で対処する方法を次の表にまとめています。

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
getPlayerFriends	ベーステーブル	GetItem	PK=PlayerID	SK=“FRIENDS#playerID”	

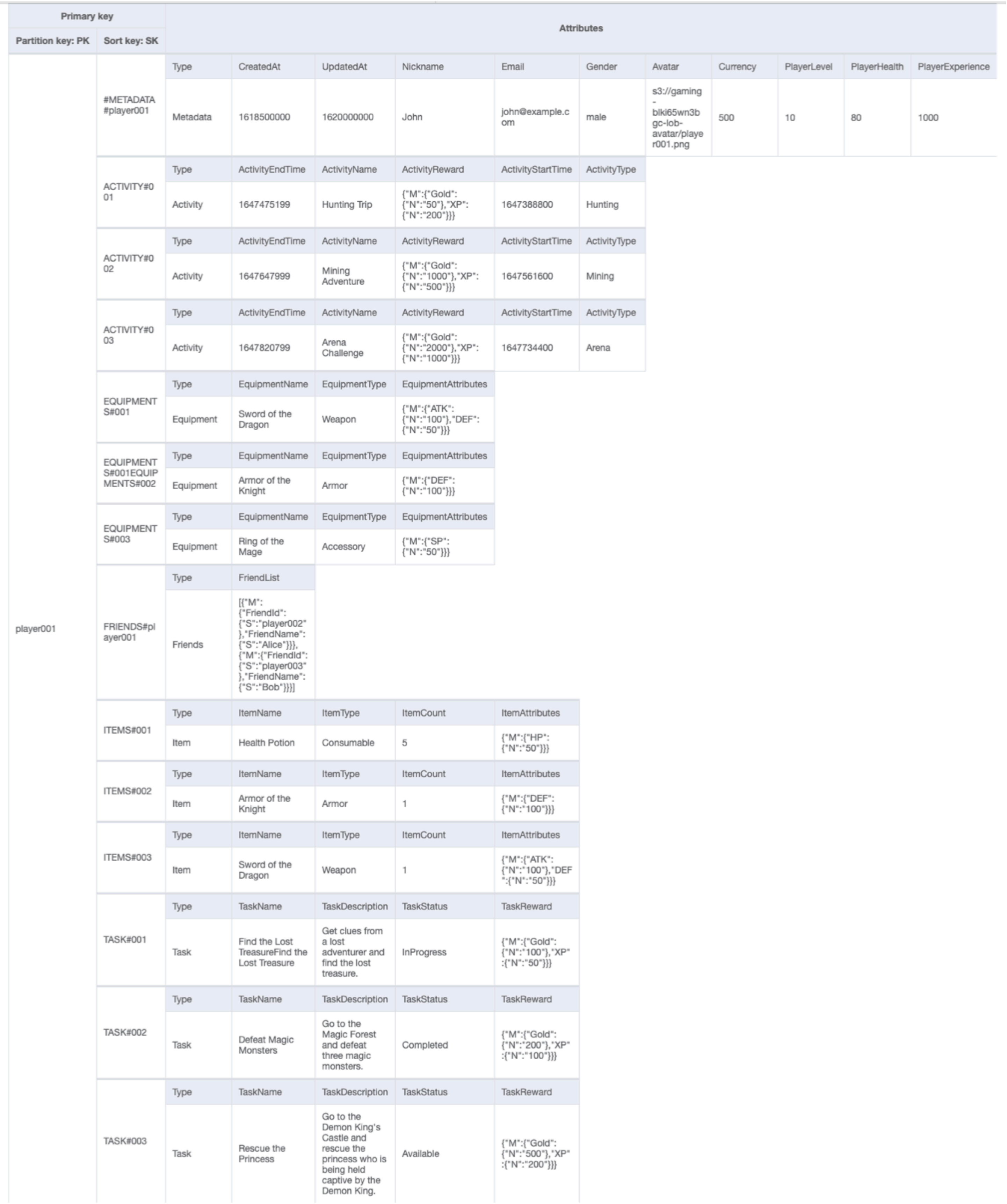
アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
getPlayerAllProfile	ベーステーブル	Query	PK=PlayerID		
getPlayerAllItems	ベーステーブル	Query	PK=PlayerID	SK begins_with “ITEMS#”	
getPlayerSpecificItem	ベーステーブル	Query	PK=PlayerID	SK begins_with “ITEMS#”	filterExpression: "ItemType = :itemType " expressionAttributeValues: { ":itemType": "Weapon" }
updateCharacterAttributes	ベーステーブル	UpdateItem	PK=PlayerID	SK=“#METADATA#playerID”	UpdateExpression: "SET currency = currency - :amount" ConditionExpression: "currency >= :minAmount"

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
updateItemCount	ベーステーブル	UpdateItem	PK=PlayerID	SK =“ITEMS#ItemID”	update-expression: "SET ItemCount = ItemCount - :incr" expression-attribute-values: '{":incr":{"N":"1"}}'

ゲームプロファイルの最終スキーマ

これが最終的なスキーマ設計です。このスキーマ設計を JSON ファイルとしてダウンロードするには、GitHub の DynamoDB の例を参照してください。

ベーステーブル:



Primary key		Attributes										
Partition key: PK	Sort key: SK											
player001	#METADATA#player001	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
		Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500	10	80	1000
	ACTIVITY#001	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647475199	Hunting Trip	{"M":{"Gold":{"N":"50"},"XP":{"N":"200"}}}	1647388800	Hunting					
	ACTIVITY#002	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647647999	Mining Adventure	{"M":{"Gold":{"N":"1000"},"XP":{"N":"500"}}}	1647561600	Mining					
	ACTIVITY#003	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647820799	Arena Challenge	{"M":{"Gold":{"N":"2000"},"XP":{"N":"1000"}}}	1647734400	Arena					
	EQUIPMENTS#001	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Sword of the Dragon	Weapon	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}							
	EQUIPMENTS#001EQUIPMENTS#002	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Armor of the Knight	Armor	{"M":{"DEF":{"N":"100"}}}							
	EQUIPMENTS#003	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Ring of the Mage	Accessory	{"M":{"SP":{"N":"50"}}}							
	FRIENDS#player001	Type	FriendList									
		Friends	[{"M":{"FriendId":{"S":"player002"},"FriendName":{"S":"Alice"}}},{"M":{"FriendId":{"S":"player003"},"FriendName":{"S":"Bob"}}}]									
	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}						
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}						
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}						
	TASK#001	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Find the Lost TreasureFind the Lost Treasure	Get clues from a lost adventurer and find the lost treasure.	InProgress	{"M":{"Gold":{"N":"100"},"XP":{"N":"50"}}}						
	TASK#002	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Defeat Magic Monsters	Go to the Magic Forest and defeat three magic monsters.	Completed	{"M":{"Gold":{"N":"200"},"XP":{"N":"100"}}}						
	TASK#003	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Rescue the Princess	Go to the Demon King's Castle and rescue the princess who is being held captive by the Demon King.	Available	{"M":{"Gold":{"N":"500"},"XP":{"N":"200"}}}						

このスキーマ設計での NoSQL Workbench の使用

この最終スキーマを、DynamoDB のデータモデリング、データ視覚化、クエリ開発機能を提供するビジュアルツールである NoSQL Workbench にインポートして、新しいプロジェクトを詳しく調べたり編集したりできます。使用を開始するには、次の手順に従います。

1. NoSQL Workbench をダウンロードします。詳細については、「the section called “ダウンロード”」を参照してください。
2. 上記の JSON スキーマファイルをダウンロードします。このファイルは既に NoSQL Workbench モデル形式になっています。
3. JSON スキーマファイルを NoSQL Workbench にインポートします。詳細については、「the section called “既存のモデルのインポート”」を参照してください。
4. NOSQL Workbench にインポートしたら、データモデルを編集できます。詳細については、「the section called “既存モデルの編集”」を参照してください。
5. データモデルの視覚化、サンプルデータの追加、CSV ファイルからのサンプルデータのインポートを行うには、NoSQL Workbench のデータビジュアライザー機能を使用します。

DynamoDB の苦情管理システムのスキーマ設計

苦情管理システムのビジネスユースケース

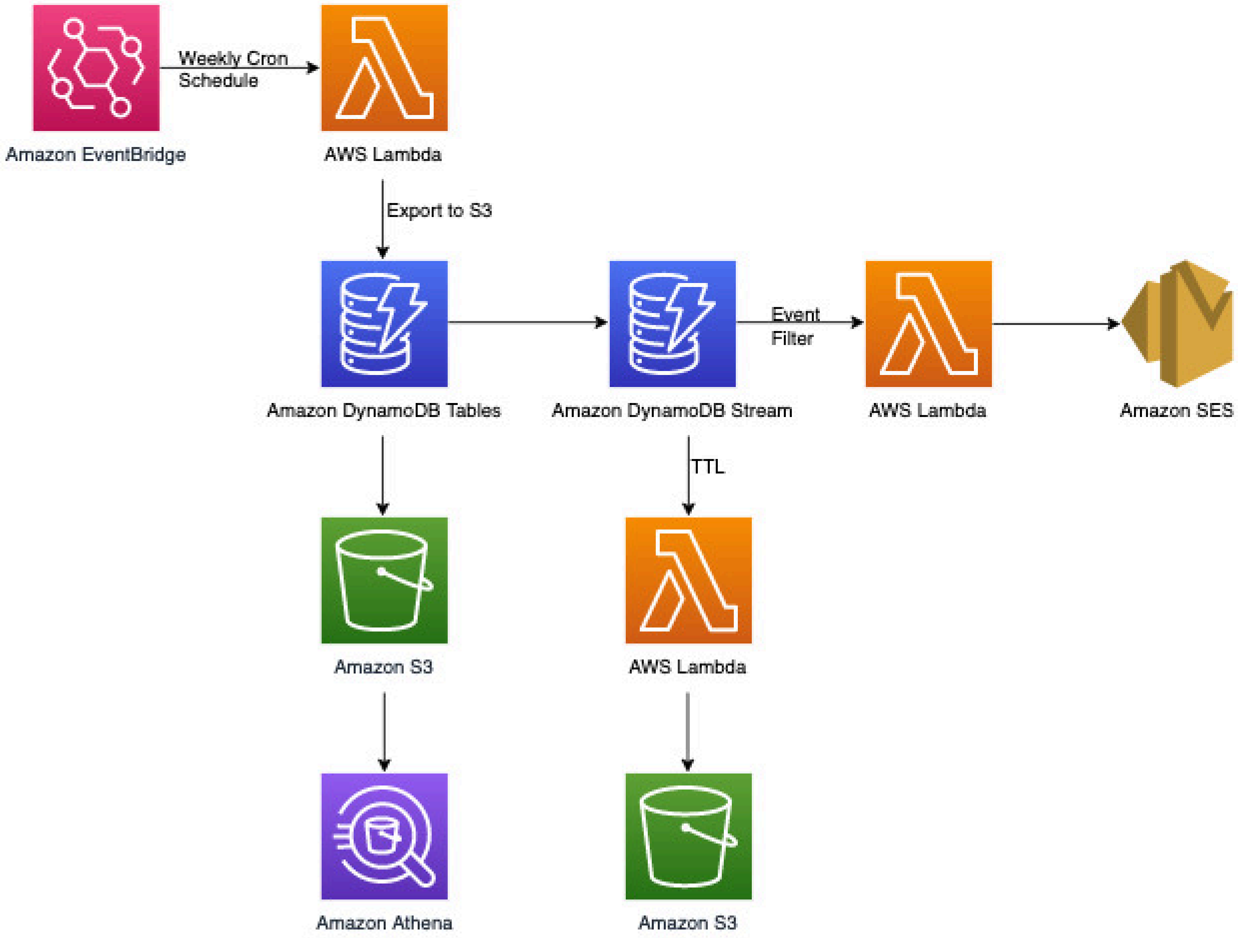
DynamoDB は、苦情管理システム (またはコンタクトセンター) のユースケースに最適なデータベースです。関連するほとんどのアクセスパターンが key-value ベースのトランザクションルックアップであるためです。このシナリオの一般的なアクセスパターンは、次のとおりです。

- 苦情を作成および更新する
- 苦情をエスカレートする
- 苦情に関するコメントを作成および閲覧する
- 顧客からのすべての苦情を取得する
- エージェントからのすべてのコメントを取得し、すべてのエスカレーションを取得する

一部のコメントには、苦情または解決策を説明する添付ファイルが付いている場合があります。これらはすべて key-value アクセスパターンですが、苦情に新しいコメントが追加されたときに通知を送信したり、分析クエリを実行して重大度 (またはエージェントのパフォーマンス) 別に苦情の分布を毎週調べたりするなど、追加の要件が伴う場合があります。ライフサイクル管理やコンプライアンス

に関連するその他の要件としては、苦情を 3 年間記録した後に苦情データをアーカイブすることが挙げられます。

苦情管理システムのアーキテクチャ図



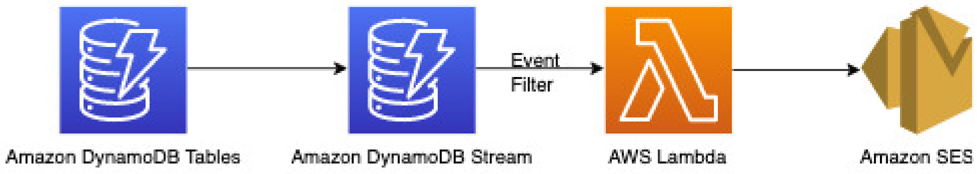
後述の DynamoDB データモデリングセクションで扱う key-value トランザクションアクセスパターンとは別に、非トランザクション要件が 3 つあります。上のアーキテクチャ図は、次の 3 つのワークフローに分けることができます。

1. 苦情に新しいコメントが追加されたときに通知を送信する
2. 週次データに対して分析クエリを実行する
3. 3 年より前のデータをアーカイブする

それぞれを詳しく見ていきましょう。

苦情に新しいコメントが追加されたときに通知を送信する

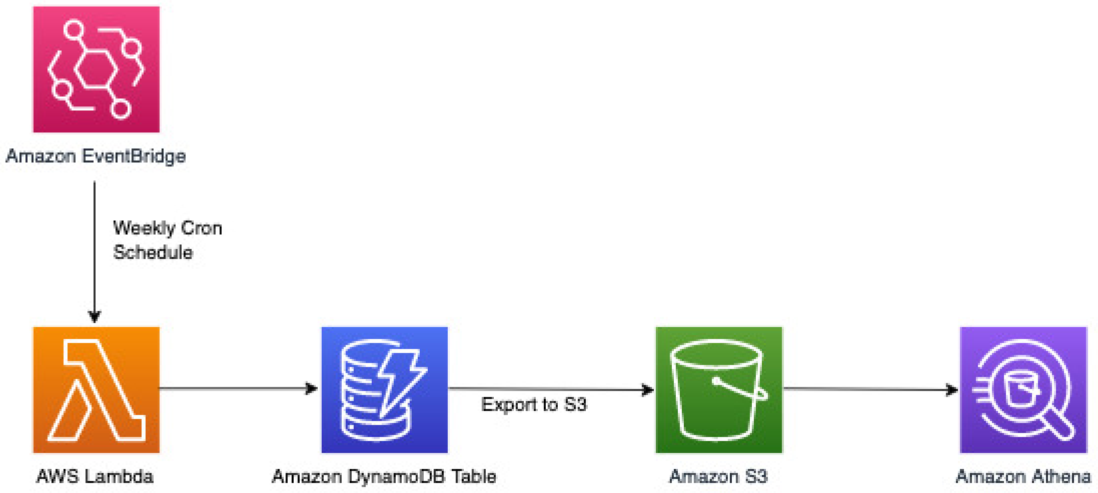
この要件を満たすには、次のワークフローを使用できます。



DynamoDB ストリームは、DynamoDB テーブルに対するすべての書き込みアクティビティを記録する変更データキャプチャメカニズムです。これらの変更の一部またはすべてに応じてトリガーされるように Lambda 関数を設定できます。Lambda トリガーにイベントフィルターを設定して、ユースケースに関係のないイベントを除外できます。この例では、フィルターを使用して、新しいコメントが追加されたときにのみ Lambda をトリガーし、関連する E メール ID (AWS Secret Manager またはその他の認証情報ストアから取得可能) に通知を送信できます。

週次データに対して分析クエリを実行する

DynamoDB は、オンライントランザクション処理 (OLTP) を主眼とするワークロードに適しています。分析が必要な他の 10～20% のアクセスパターンについては、マネージド機能の Amazon S3 へのエクスポートを使用してデータを S3 にエクスポートできます。この機能を使用しても、DynamoDB テーブルのライブトラフィックには影響がありません。次のワークフローをご覧ください。

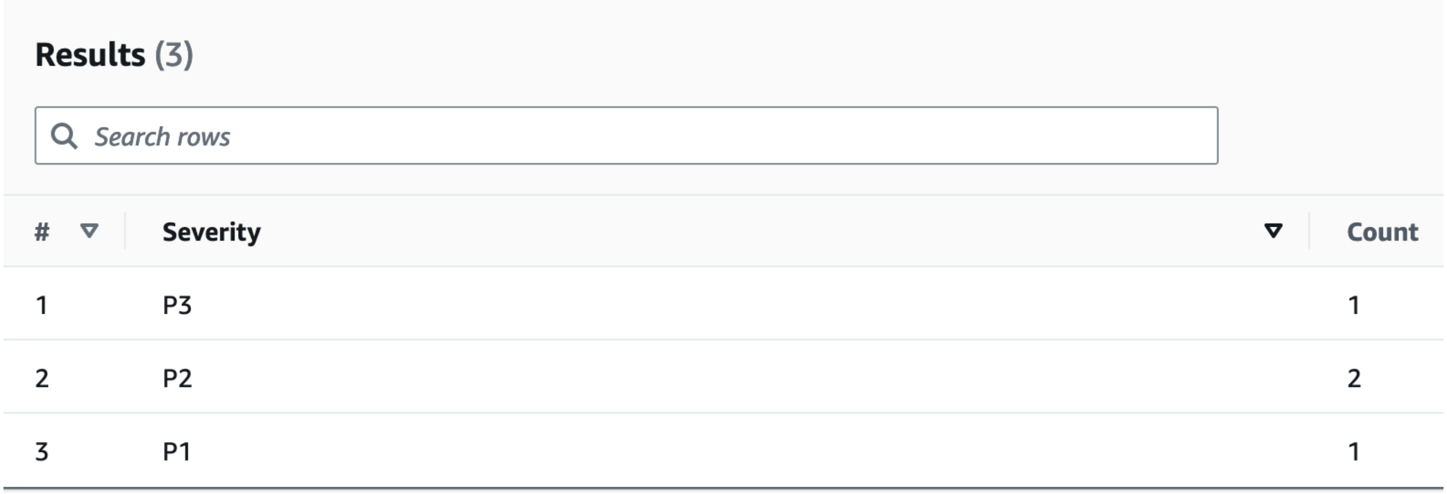


Amazon EventBridge を使用すると、AWS Lambda をスケジュールどおりにトリガーできます。これにより、Lambda を定期的に呼び出すように cron 式を設定できます。Lambda は `ExportToS3` API コールを呼び出し、DynamoDB データを S3 に保存します。この S3 データ

に Amazon Athena などの SQL エンジンからアクセスし、テーブルのライブトランザクションワークロードには影響を与えることなく、DynamoDB データに対して分析クエリを実行できます。重大度レベル別に苦情数を調べる Athena クエリの例は、次のようになります。

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

この Athena クエリは次の結果を返します。



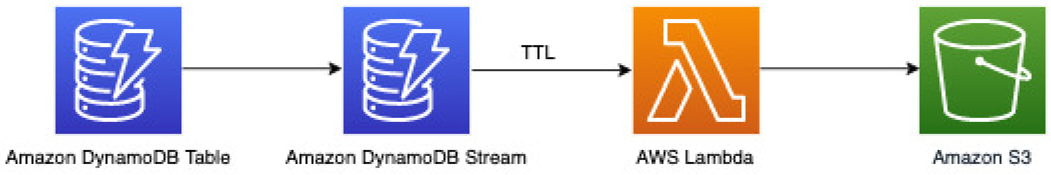
Results (3)

Search rows

#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

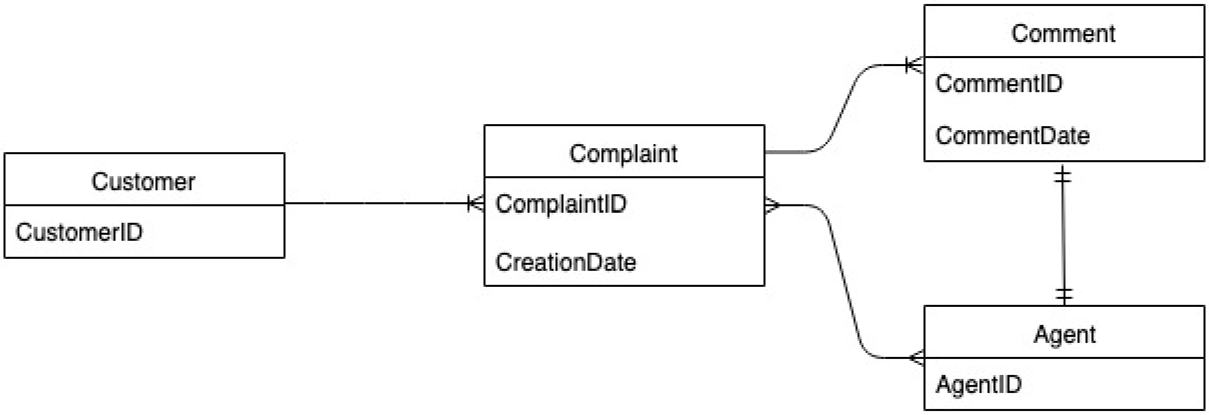
3 年より前のデータをアーカイブする

DynamoDB の Time to Live (TTL) 機能を使用して、追加費用なしで、DynamoDB テーブルから古いデータを削除できます (ただし、2019.11.21 (現行) バージョンのグローバルテーブルレプリカは除きます。TTL 削除が他のリージョンにレプリケートされると書き込み容量が消費されるためです)。このデータを DynamoDB ストリームで表示し、使用できます。その後に、Amazon S3 にアーカイブできます。この要件のワークフローは次のとおりです。



苦情管理システムのエンティティ関係図

次に示すのは、苦情管理システムのスキーマ設計に使用するエンティティ関係図 (ERD) です。



苦情管理システムのアクセスパターン

以下は、苦情管理システムのスキーマ設計で検討するアクセスパターンです。

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID
4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID (新しいものから古いものへの順)
13. getCommentsByAgentID (2 つの日付間)

苦情管理システムのスキーマ設計の進化

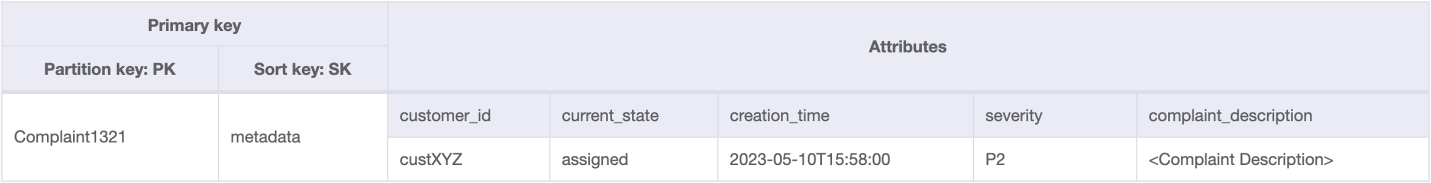
これは苦情管理システムであるため、ほとんどのアクセスパターンはプライマリエンティティとしての苦情を中心に展開します。`ComplaintID` のカーディナリティが高いと、基盤となるパーティションにデータが均等に分散されます。また、特定したアクセスパターンの最も一般的な検索条件となり

ます。したがって、ComplaintID は、このデータセットのパーティションキー候補として適しています。

ステップ 1: アクセスパターン 1 (**createComplaint**)、2 (**updateComplaint**)、3 (**updateSeveritybyComplaintID**)、4 (**getComplaintByComplaintID**) に対処する

「metadata」(または「AA」) という汎用ソートキー値を使用して、CustomerID、State、Severity、CreationDate などの苦情固有の情報を保存できます。PK=ComplaintID と SK=“metadata” でシングルトンオペレーションを使用して、以下を行います。

1. PutItem で新しい苦情を作成する
2. UpdateItem で苦情メタデータの重大度やその他のフィールドを更新する
3. GetItem で苦情のメタデータを取得する



Primary key		Attributes				
Partition key: PK	Sort key: SK					
Complaint1321	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	assigned	2023-05-10T15:58:00	P2	<Complaint Description>

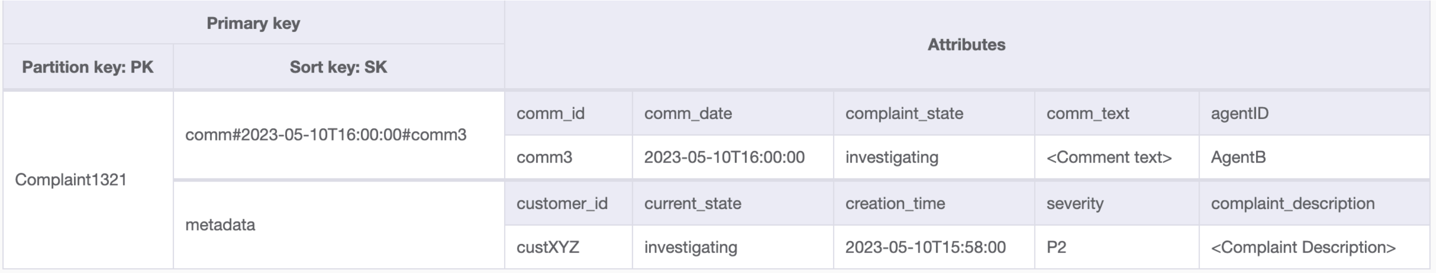
ステップ 2: アクセスパターン 5 (**addCommentByComplaintID**) に対処する

このアクセスパターンでは、苦情と苦情に関するコメントとの間に 1 対多リレーションシップモデルが必要です。ここでは、垂直パーティショニング手法を通じてソートキーを使用し、さまざまなタイプのデータを含む項目コレクションを作成します。アクセスパターン 6 (getAllCommentsByComplaintID) と 7 (getLatestCommentByComplaintID) に注目すると、コメントを時間順に並べ替える必要があることがわかります。また、複数のコメントを同時に受け取ることができるため、複合ソートキー手法を使用して時間と CommentID をソートキー属性に追加できます。

このようなコメント競合の可能性に対処する他のオプションとしては、タイムスタンプの粒度を増やすか、Comment_ID を使用する代わりに増分数値をサフィックスとして追加します。この場合、範囲ベースの操作を可能にするために、コメントに対応する項目のソートキー値の前に「comm#」を付けます。

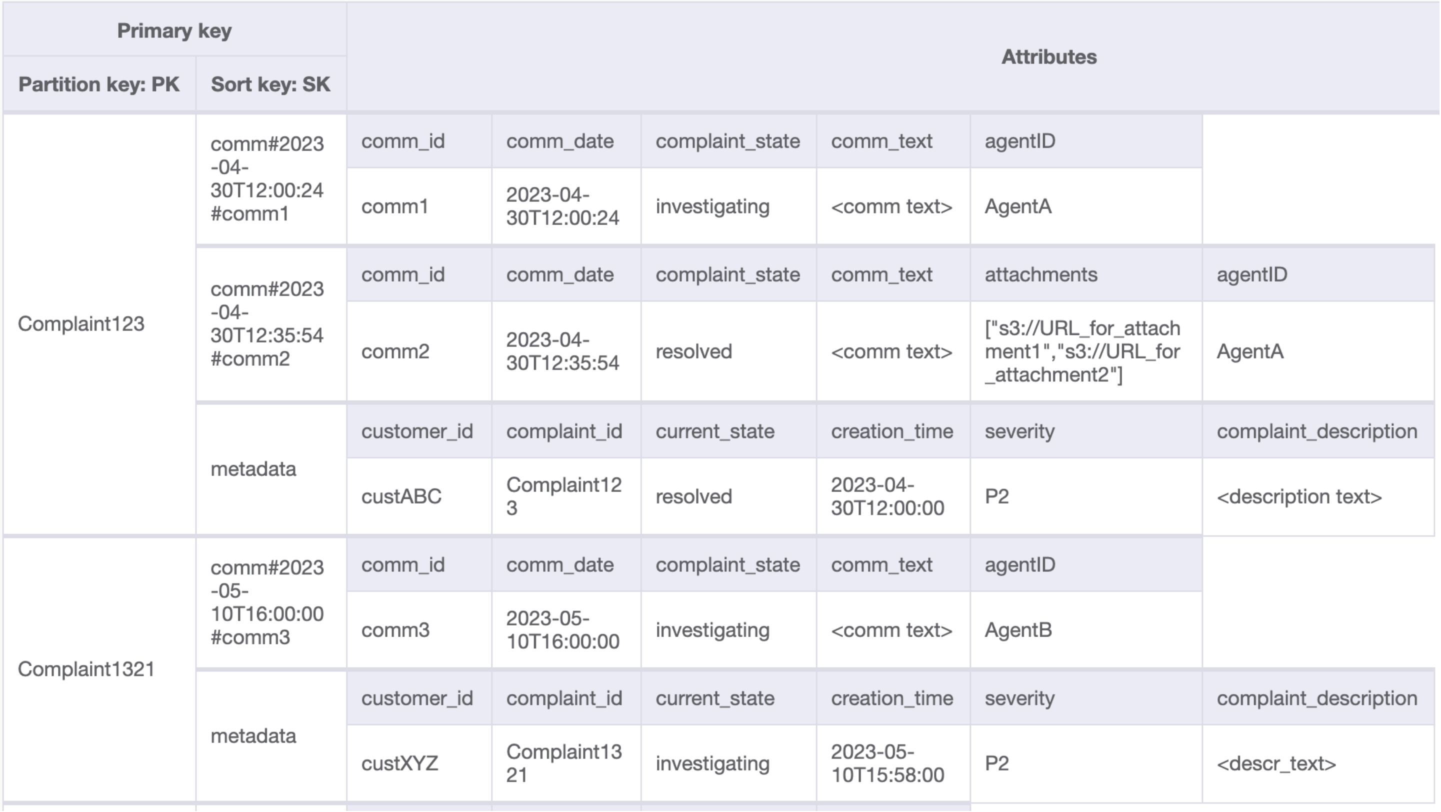
また、苦情メタデータの currentState に、新しいコメントを追加したときの状態が反映されることを確認する必要があります。コメントの追加は、エージェントへの苦情の割り当てや苦情の解決などを示す場合があります。コメントの追加と現在の状態の更新を苦情メタデータにバンドルするため

に、オールオアナッシング方式で、TransactWriteItems API を使用します。結果として得られるテーブルの状態は次のようになります。



Primary key		Attributes				
Partition key: PK	Sort key: SK					
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID
		comm3	2023-05-10T16:00:00	investigating	<Comment text>	AgentB
	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	investigating	2023-05-10T15:58:00	P2	<Complaint Description>

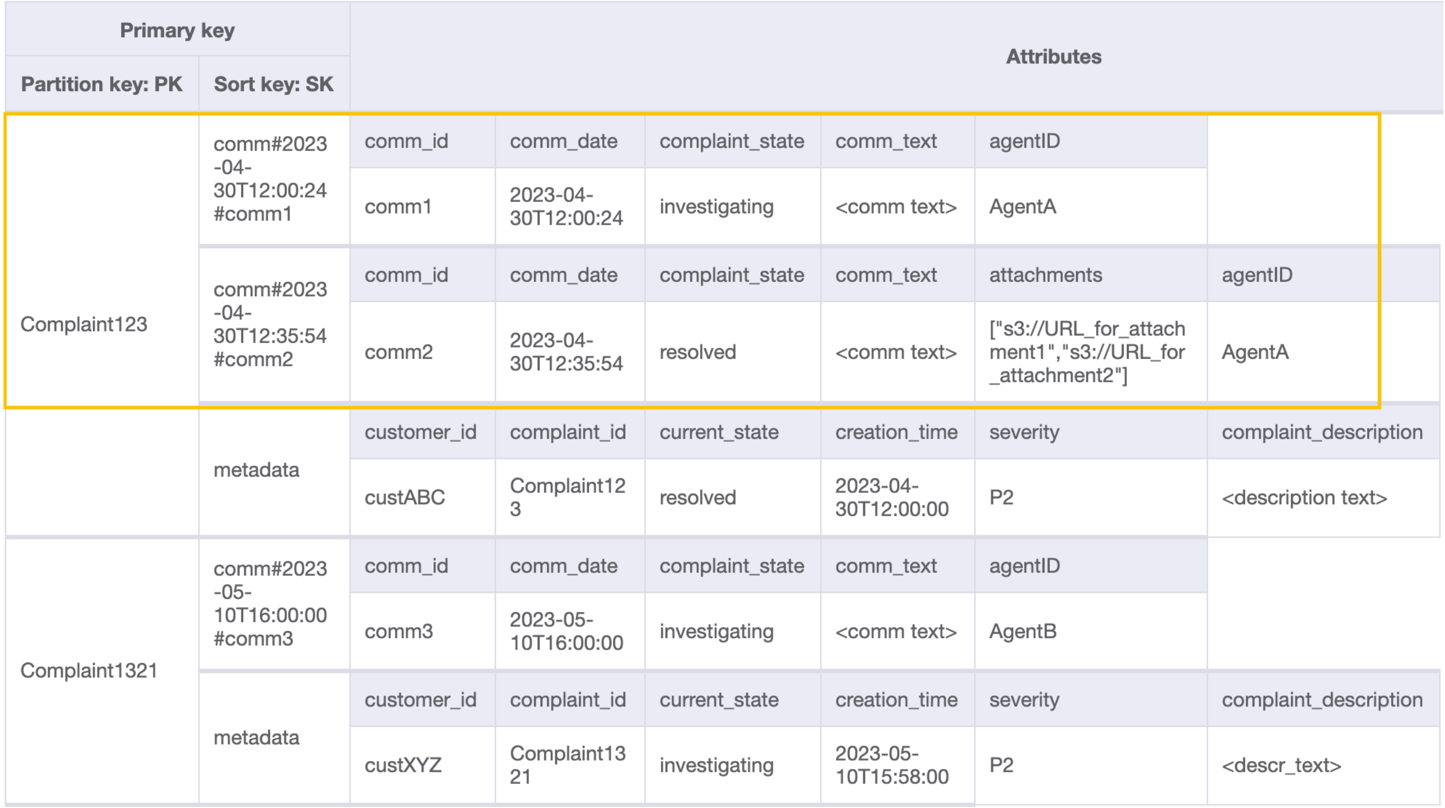
テーブルにさらにデータを追加し、PK とは別のフィールドとして `ComplaintID` も追加しましょう。`ComplaintID` に追加のインデックスが必要な場合に備えて、モデルを将来的に保証するためです。また、一部のコメントには添付ファイルが含まれている場合があることに注意してください。添付ファイルは、Amazon Simple Storage Service に保存し、その参照または URL のみを DynamoDB で管理します。コストとパフォーマンスを最適化するために、トランザクションデータベースはできるだけリーンな状態に保つことがベストプラクティスです。これで、データは次のようになります。



Primary key		Attributes					
Partition key: PK	Sort key: SK						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

ステップ 3: アクセスパターン 6 (**`getAllCommentsByComplaintID`**) と 7 (**`getLatestCommentByComplaintID`**) に対処する

苦情に関するすべてのコメントを取得するには、ソートキーに対する query オペレーションで begins_with 条件を使用できます。メタデータエントリを読み取るために追加の読み取り容量を消費したり、関連する結果をフィルタリングするためのオーバーヘッドを生じたりせずに、このようなソートキー条件を設定することで、必要なものだけを読み取ることができます。例えば、PK=Complaint123 および SK begins_with comm# を使用した query オペレーションは、メタデータエントリをスキップして、以下を返します。

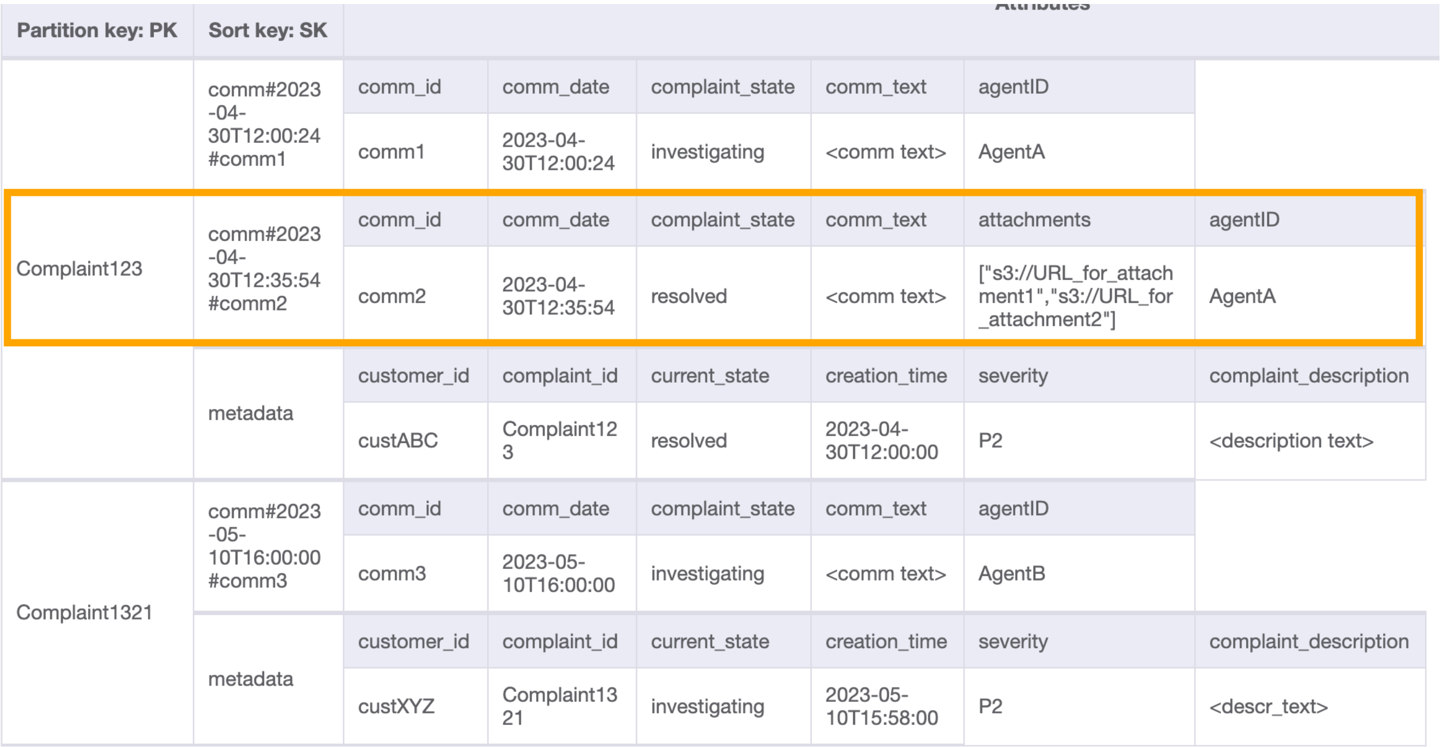


Primary key		Attributes					
Partition key: PK	Sort key: SK						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

パターン 7 (getLatestCommentByComplaintID) では苦情に関する最新のコメントが必要であるため、次の 2 つのクエリパラメータを追加しましょう。

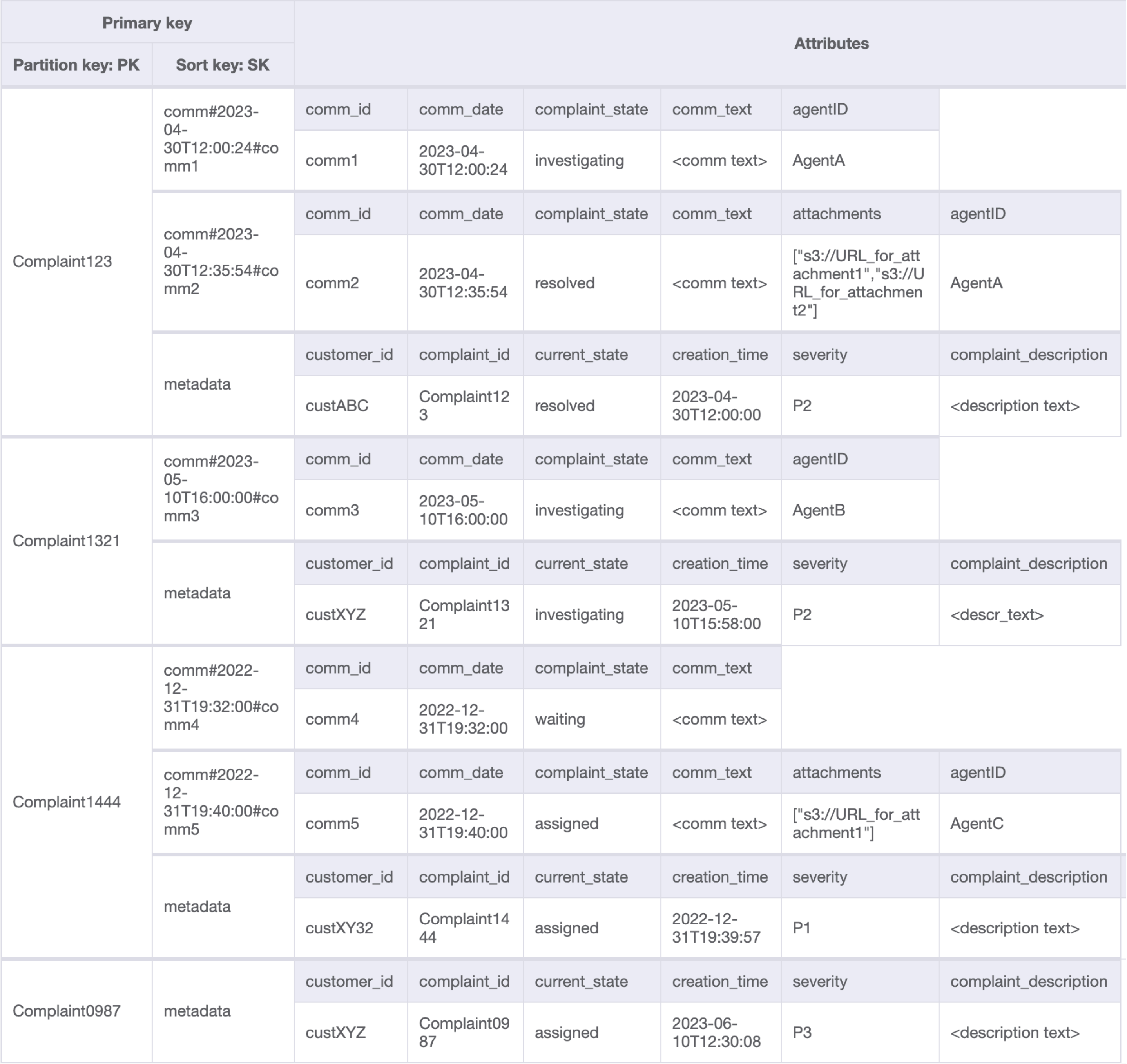
1. 結果を降順にソートするために False に設定する ScanIndexForward
2. 最新のコメントを (1つだけ) 取得するために 1 に設定する Limit

アクセスパターン 6 (getAllCommentsByComplaintID) と同じように、ソートキー条件として begins_with comm# を使用してメタデータエントリを省略します。これで、クエリオペレーションで PK=Complaint123 と SK=begins_with comm# に加えて、ScanIndexForward=False と Limit を使用し、この設計でアクセスパターン 7 を実行できるようになりました。結果として、次のターゲット項目が返されます。



Partition key: PK	Sort key: SK	Attributes					
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

さらにダミーデータをテーブルに追加してみましょう。



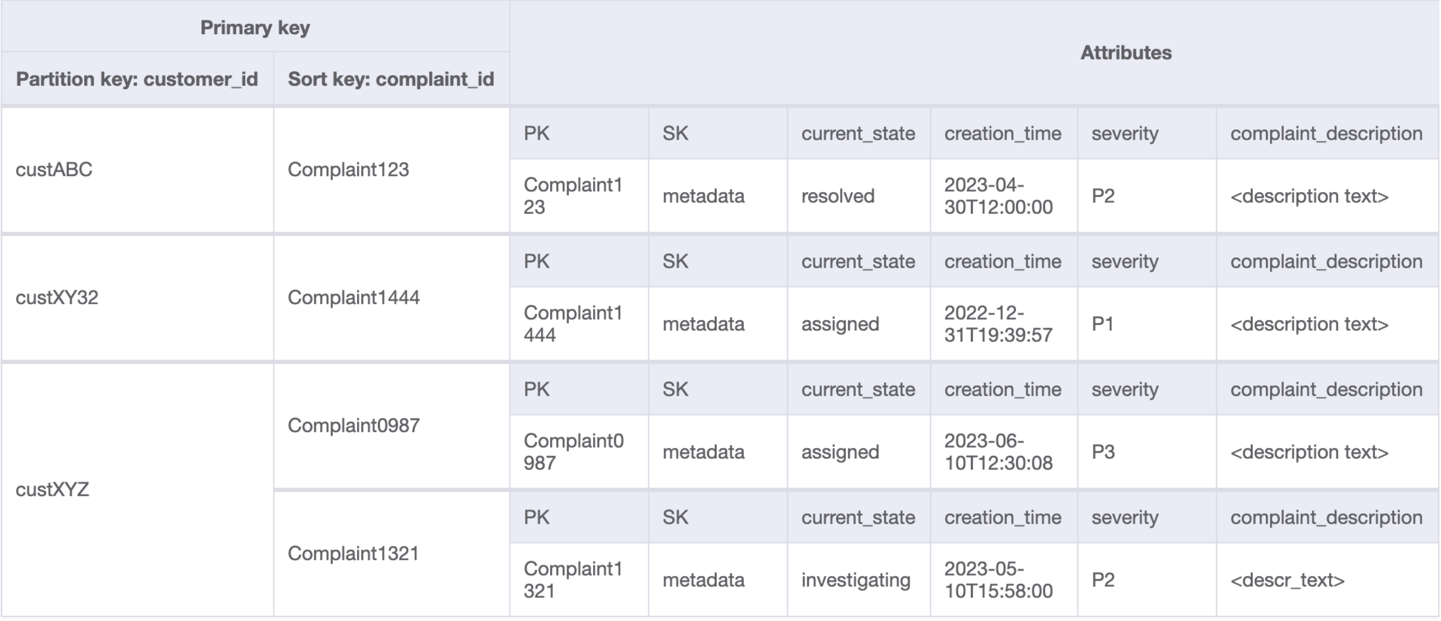
Primary key		Attributes					
Partition key: PK	Sort key: SK						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text		
		comm4	2022-12-31T19:32:00	waiting	<comm text>		
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>

ステップ 4: アクセスパターン 8 (**getAComplaintbyCustomerIDAndComplaintID**) と 9 (**getAllComplaintsByCustomerID**) に対処する

アクセスパターン 8 (getAComplaintbyCustomerIDAndComplaintID) と 9 (getAllComplaintsByCustomerID) では、新しい検索条件 CustomerID を導入します。これを既存のテーブルから取得するには、すべてのデータを読み取って当の CustomerID の関連項目をフィルタリングするために、コストの高い Scan が必要になります。この検索をより効

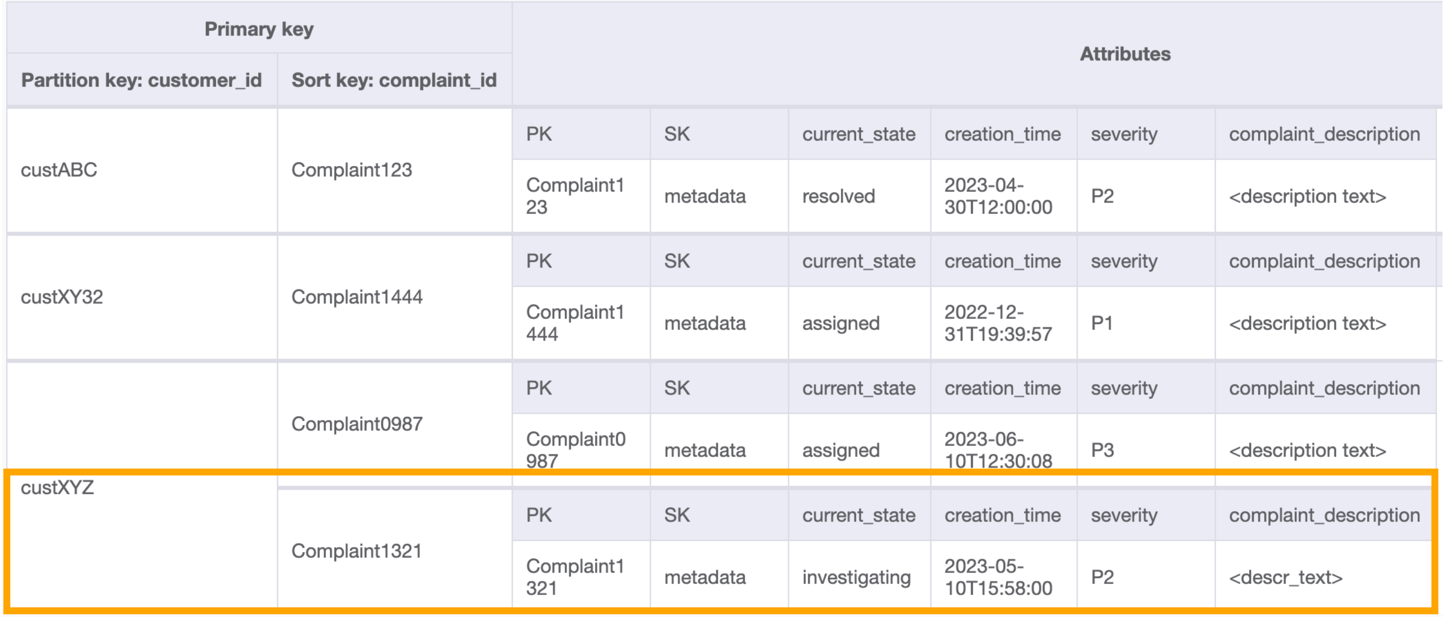
率的にするには、CustomerID をパーティションキーとして[グローバルセカンダリインデックス (GSI)](#) を作成できます。顧客と苦情の 1 対多リレーションシップおよびアクセスパターン 9 (getAllComplaintsByCustomerID) を考慮すると、ComplaintID がソートキーの適切な候補となります。

GSI のデータは次のようになります。



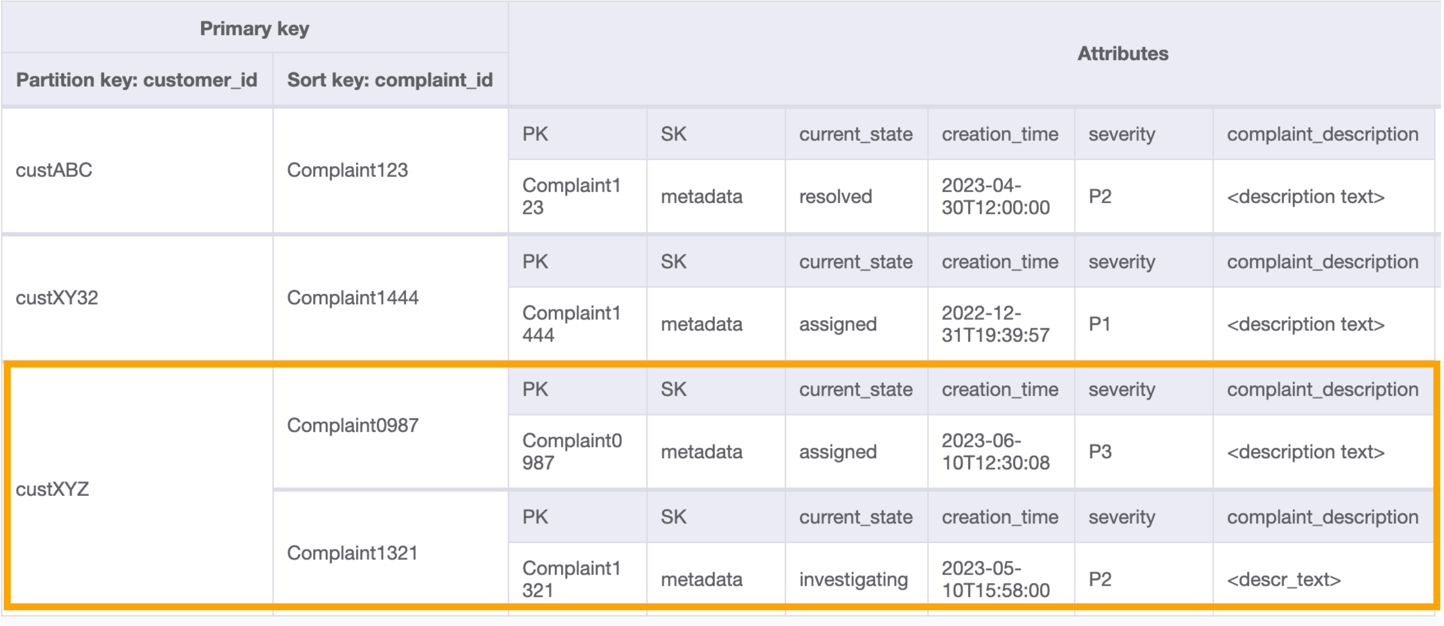
Primary key		Attributes					
Partition key: customer_id	Sort key: complaint_id						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

この GSI のクエリ例はアクセスパターン 8 (getAComplaintbyCustomerIDAndComplaintID) で customer_id=custXYZ、sort key=Complaint1321 となります。結果は次のようになります。



Primary key		Attributes					
Partition key: customer_id	Sort key: complaint_id						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

アクセスパターン 9 (`getAllComplaintsByCustomerID`) でお客様の苦情をすべて取得する場合、GSI のクエリはパーティションキー条件が `customer_id=custXYZ` になります。結果は次のようになります。

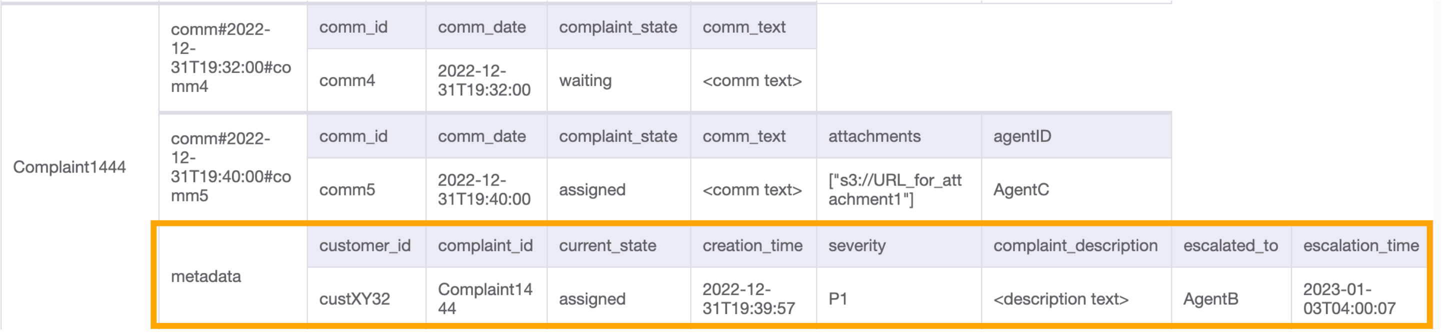


Primary key		Attributes					
Partition key: customer_id	Sort key: complaint_id						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

ステップ 5: アクセスパターン 10 (**`escalateComplaintByComplaintID`**) に対処する

このアクセスでは、エスカレーションのディメンションを導入します。苦情をエスカレーションするには、`UpdateItem` を使用して `escalated_to` や `escalation_time` などの属性を既存の苦情メタデータ項目に追加します。DynamoDB のスキーマ設計は柔軟であるため、キー以外の属性のセットをさまざまな項目間で均一にすることも、個別にすることもできます。次の例を参照してください。

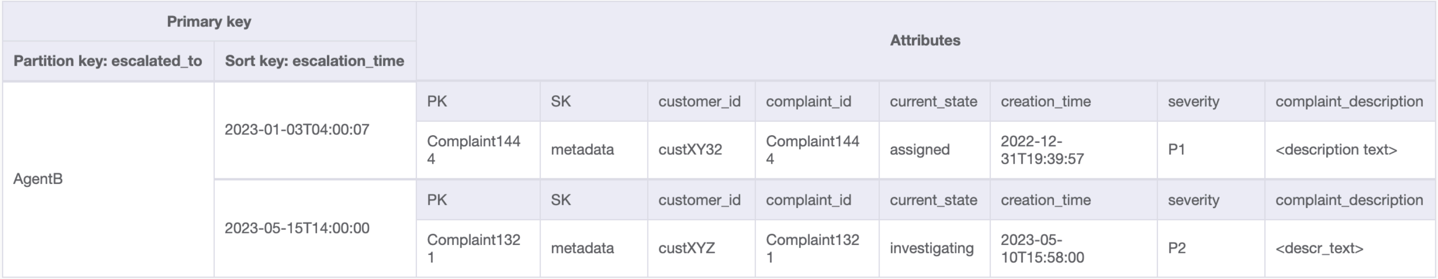
```
UpdateItem with PK=Complaint1444, SK=metadata
```



Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07

ステップ 6: アクセスパターン 11 (**`getAllEscalatedComplaints`**) と 12 (**`getEscalatedComplaintsByAgentID`**) に対処する

データセット全体からエスカレーションされると予想される苦情はほんの一握りです。したがって、エスカレーション関連の属性にインデックスを作成すると、効率的な検索と費用対効果の高い GSI ストレージが実現します。これを実現するには、スパースインデックス手法を活用できます。パーティションキーを escalated_to、ソートキーを escalation_time とする GSI は、次のようになります。

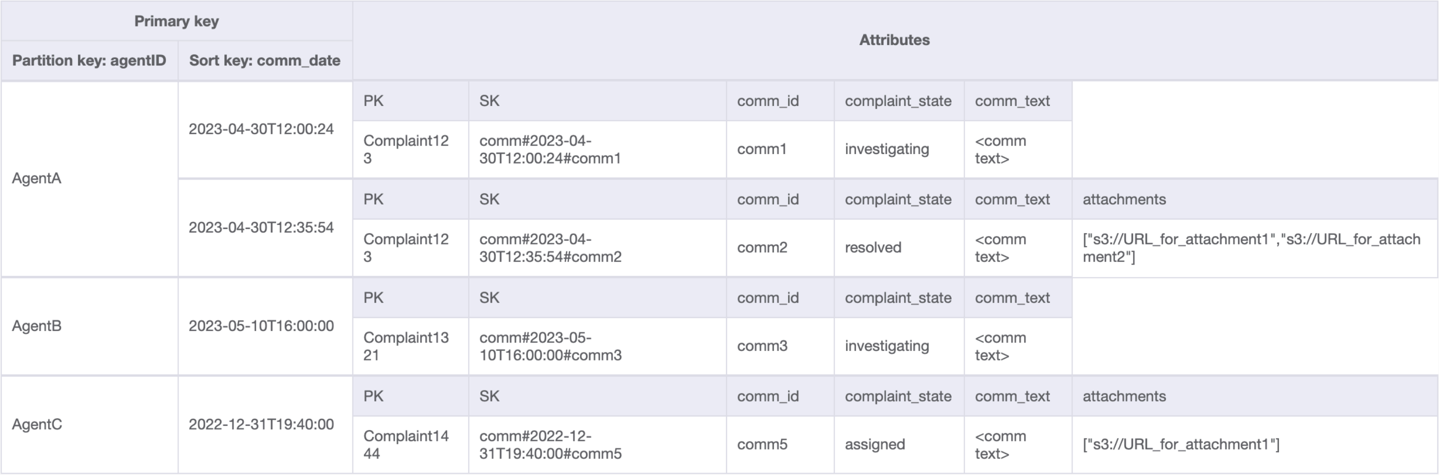


Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

アクセスパターン 11 (getAllEscalatedComplaints) ですべてのエスカレーションされた苦情を取得するには、この GSI を単にスキャンします。このスキャンは、GSI のサイズにより、パフォーマンスとコスト効率が向上することに注意してください。特定のエージェント (アクセスパターン 12 (getEscalatedComplaintsByAgentID)) でエスカレーションされた苦情を取得する場合は、パーティションキーを escalated_to=agentID とし、ScanIndexForward を False に設定して新しいものから古いものへの順に取得します。

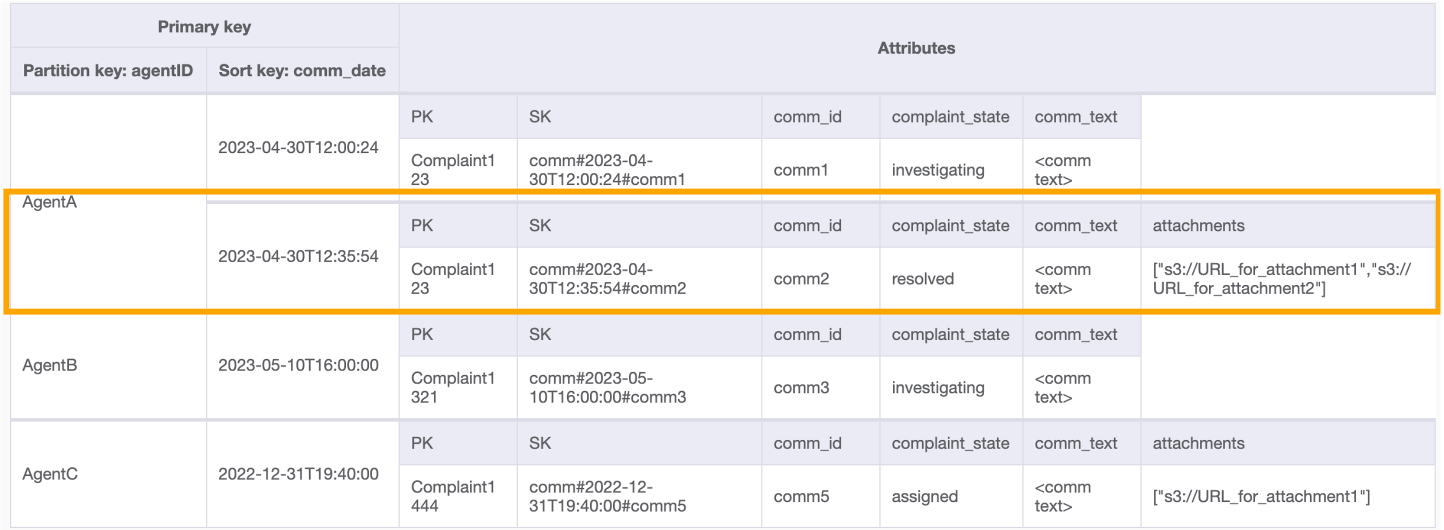
ステップ 7: アクセスパターン 13 (**getCommentsByAgentID**) に対処する

最後のアクセスパターンでは、新しいディメンション AgentID によるルックアップを実行する必要があります。また、2 つの日付間のコメントを読み取るには時間ベースの順序付けも必要であるため、パーティションキーを agent_id、ソートキーを comm_date として GSI を作成します。この GSI のデータは次のようになります。



Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

この GSI のクエリの例は `partition key agentID=AgentA` および `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)` となり、その結果は次のようになります。



Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

すべてのアクセスパターンと各アクセスパターンにスキーマ設計で対処する方法を次の表にまとめています。

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
createComplaint	ベーステーブル	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	ベーステーブル	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	ベーステーブル	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByComplaintID	ベーステーブル	GetItem	PK=complaint_id	SK=metadata	

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
addCommentByComplaintID	ベーステーブル	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#comm_date#comm_id	
getAllCommentsByComplaintID	ベーステーブル	Query	PK=complaint_id	SK begins_with "comm#"	
getLatestCommentByComplaintID	ベーステーブル	Query	PK=complaint_id	SK begins_with "comm#"	scan_index_forward=False, Limit 1
getAComplaintbyCustomerIDAndComplaintID	Customer_complaint_GSI	Query	customer_id=customer_id	complaint_id = complaint_id	
getAllComplaintsByCustomerID	Customer_complaint_GSI	Query	customer_id=customer_id	該当なし	
escalateComplaintByComplaintID	ベーステーブル	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedComplaints	Escalations_GSI	Scan	該当なし	該当なし	

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
getEscalatedComplaintsByAgentID (新しいものから古いものへの順)	Escalations_GSI	Query	escalated_to=agent_id	該当なし	scan_index_forward=False
getCommentsByAgentID (2 つの日付間)	Agents_Comments_GSI	Query	agent_id=agent_id	SK between (date1, date2)	

苦情管理システムの最終スキーマ

最終的なスキーマ設計は次のとおりです。このスキーマ設計を JSON ファイルとしてダウンロードするには、GitHub の DynamoDB の例を参照してください。

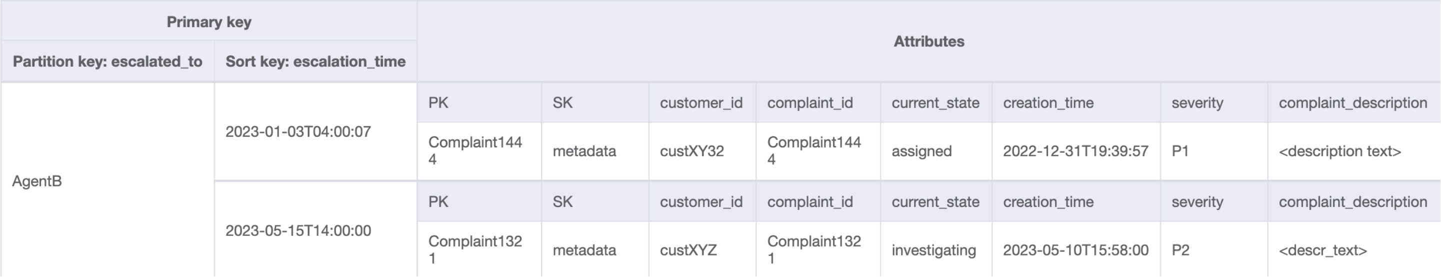
ベーステーブル

Primary key		Attributes							
Partition key: PK	Sort key: SK								
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA			
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>		
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB			
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>		

Customer_Complaint_GSI

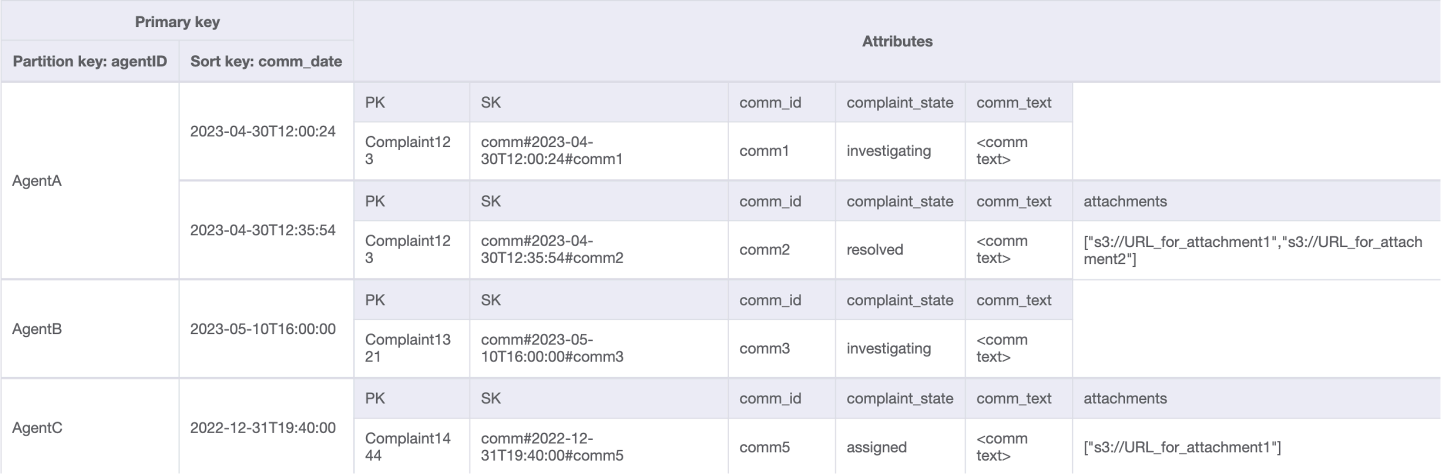
Primary key		Attributes							
Partition key: customer_id	Sort key: complaint_id								
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>		
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>		
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00

Escalations_GSI



Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Agents_Comments_GSI



Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

このスキーマ設計での NoSQL Workbench の使用

この最終スキーマを、DynamoDB のデータモデリング、データ視覚化、クエリ開発機能を提供するビジュアルツールである NoSQL Workbench にインポートして、新しいプロジェクトを詳しく調べたり編集したりできます。使用を開始するには、次の手順に従います。

1. NoSQL Workbench をダウンロードします。詳細については、「the section called “ダウンロード”」を参照してください。
2. 上記の JSON スキーマファイルをダウンロードします。このファイルは既に NoSQL Workbench モデル形式になっています。
3. JSON スキーマファイルを NoSQL Workbench にインポートします。詳細については、「the section called “既存のモデルのインポート”」を参照してください。
4. NOSQL Workbench にインポートしたら、データモデルを編集できます。詳細については、「the section called “既存モデルの編集”」を参照してください。
5. データモデルの視覚化、サンプルデータの追加、CSV ファイルからのサンプルデータのインポートを行うには、NoSQL Workbench のデータビジュアライザー機能を使用します。

DynamoDB での定期払いスキーマの設計

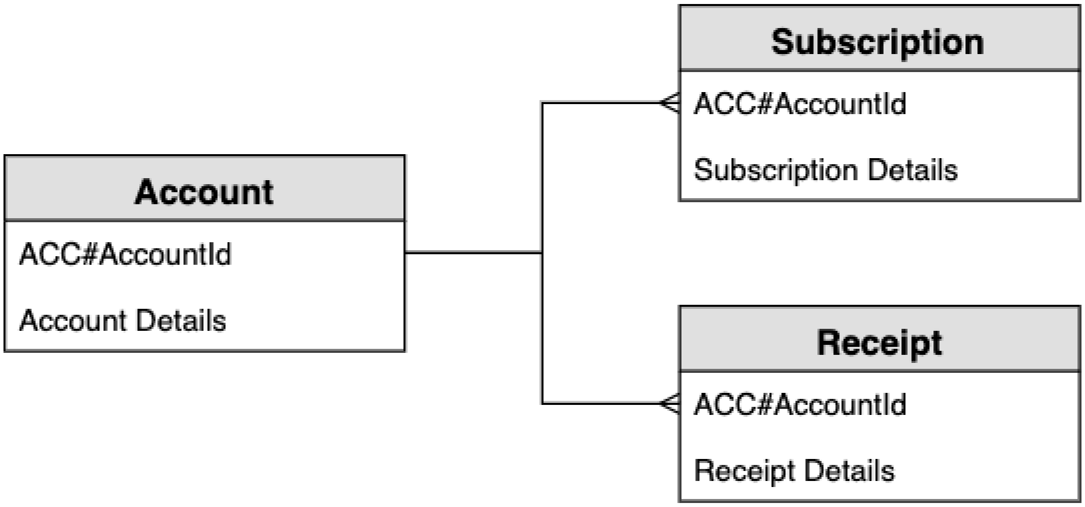
定期払いのビジネスユースケース

このユースケースでは、DynamoDB を使用して定期払いシステムを実装する方法について説明します。データモデルには、アカウント、サブスクリプション、領収書のエンティティがあります。このユースケースの詳細は次のとおりです。

- アカウントごとに複数のサブスクリプションを持つことができます。
- サブスクリプションは、次の支払いを処理する必要があるときは `NextPaymentDate`、E メールによるリマインダーをお客様に送信するときは `NextReminderDate` です。
- サブスクリプションには、支払いが処理されると、保存および更新される項目があります (平均の項目サイズは約 1 KB で、スループットはアカウントとサブスクリプションの数によって異なります)。
- 支払いプロセッサは、処理の一環として領収書も作成します。領収書は、テーブルに保存され、TTL 属性を使用して一定期間後に期限切れになるように設定されます。

定期払いエンティティ関係図

これは、定期払いシステムのスキーマ設計に使用するエンティティ関係図 (ERD) です。



定期払いシステムのアクセスパターン

これらは、定期払いシステムのスキーマ設計で検討するアクセスパターンです。

1. `createSubscription`

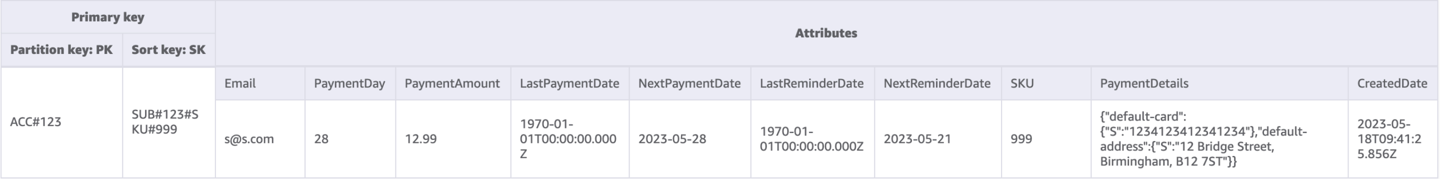
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

定期払いのスキーマ設計

一般的な名前である PK や SK は、アカウント、サブスクリプション、領収書などの異なる種類のエンティティを同じテーブルに保存するためのキー属性に使用します。ユーザーは最初にサブスクリプションを作成します。サブスクリプションにより、ユーザーは製品の代金として毎月同じ日に金額を支払うことに同意します。月内のどの日に支払いを処理するかは、ユーザーが選択できます。支払いを処理する前にリマインダーも送信されます。このアプリケーションは、毎日 2 つのバッチジョブを実行することで機能します。1 つのバッチジョブでは、当日を期限とするリマインダーを送信し、もう 1 つのバッチジョブでは、当日を支払期日とするすべての支払いを処理します。

ステップ 1: アクセスパターン 1 (**`createSubscription`**) に対処する

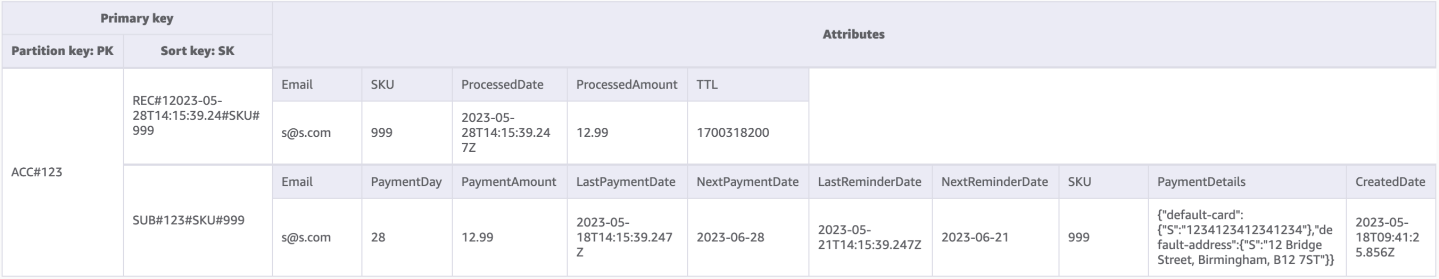
アクセスパターン 1 (`createSubscription`) を使用してサブスクリプションを最初に作成し、`SKU`、`NextPaymentDate`、`NextReminderDate`、`PaymentDetails` などの詳細を設定します。このステップでは、1 つのサブスクリプションで 1 つのアカウントに限り、テーブルの状態を表示します。項目コレクションには複数のサブスクリプションを含めることができるため、これは 1 対多リレーションシップになります。



Primary key		Attributes									
Partition key: PK	Sort key: SK										
		Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	1970-01-01T00:00:00.000Z	2023-05-28	1970-01-01T00:00:00.000Z	2023-05-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

ステップ 2: アクセスパターン 2 (**`createReceipt`**) と 3 (**`updateSubscription`**) に対処する

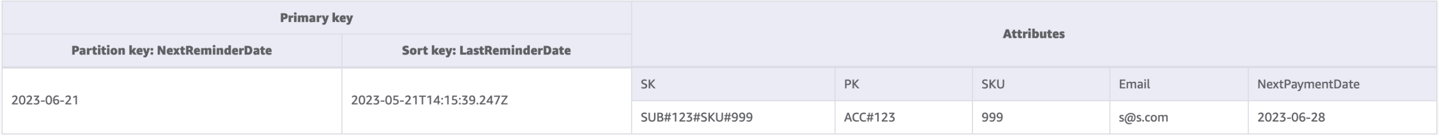
アクセスパターン 2 (`createReceipt`) を使用して領収書項目を作成します。毎月支払いを処理すると、支払いプロセッサは領収書をベーステーブルに書き戻します。項目コレクションには複数の領収書を含めることができるため、これは 1 対多リレーションシップです。支払いプロセッサは、サブスクリプション項目 (アクセスパターン 3 (`updateSubscription`)) も更新し、翌月の `NextReminderDate` または `NextPaymentDate` に変更します。



Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

ステップ 3: アクセスパターン 4 (**getDueRemindersByDate**) に対処する

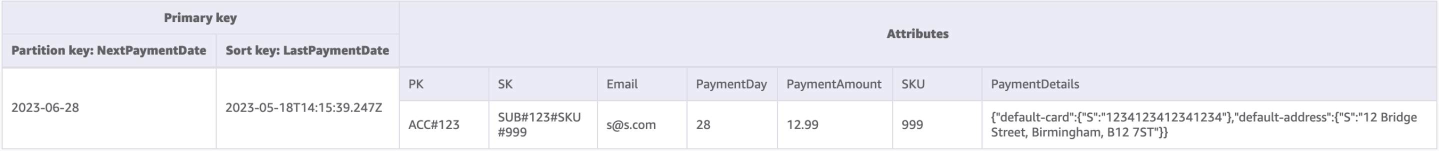
アプリケーションは、当日の支払いのリマインダーをバッチで処理します。そのため、アプリケーションは別のディメンション (アカウントではなく日付) でサブスクリプションにアクセスする必要があります。これは、グローバルセカンダリインデックス (GSI) に適したユースケースです このステップでは、インデックス GSI-1 を追加し、NextReminderDate を GSI パーティションキーとして使用します。すべての項目をレプリケートする必要はありません。この GSI はスパースインデックスであり、領収書項目はレプリケートされません。また、すべての属性を射影する必要はなく、属性の一部を含めるだけで済みます。次の図に示す GSI-1 のスキーマは、アプリケーションがリマインダー E メールを送信するために必要な情報を提供します。



Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate					
2023-06-21	2023-05-21T14:15:39.247Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

ステップ 4: アクセスパターン 5 (**getDuePaymentsByDate**) に対処する

アプリケーションは、リマインダーの場合と同じように、当日の支払いをバッチで処理します。このステップでは GSI-2 を追加し、NextPaymentDate を GSI パーティションキーとして使用します。すべての項目をレプリケートする必要はありません。GSI はスパースインデックスであり、領収書項目はレプリケートされません。次の図は GSI-2 のスキーマを示しています。

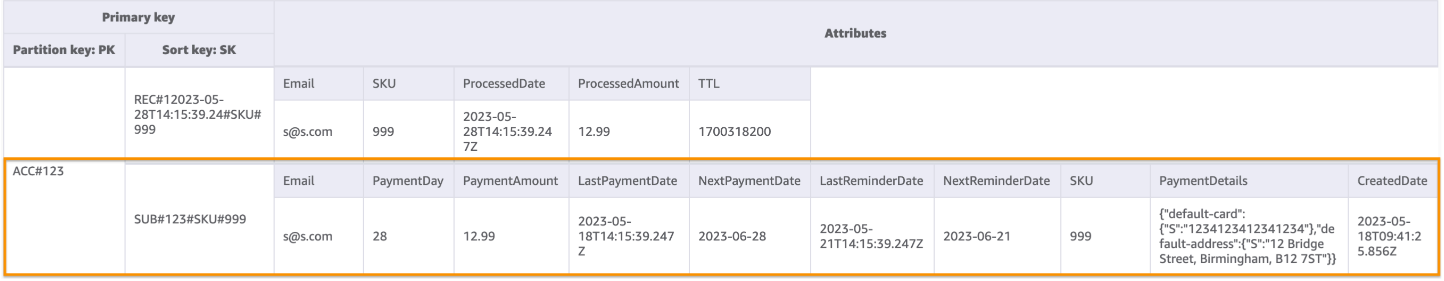


Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate							
2023-06-28	2023-05-18T14:15:39.247Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
		ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

ステップ 5: アクセスパターン 6 (**getSubscriptionsByAccount**) と 7 (**getReceiptsByAccount**) に対処する

アプリケーションは、ベーステーブルでアカウント識別子 (PK) をターゲットとするクエリを使用して、アカウントのすべてのサブスクリプションを取得できます。また、範囲演算子を使用して SK が「SUB#」で始まるすべての項目を取得できます。また、アプリケーションは、同じクエ

リ構造を使用してすべての領収書を取得し、範囲演算子を使用して SK が「REC#」で始まるすべての項目を取得できます。これにより、アクセスパターン 6 (`getSubscriptionsByAccount`) と 7 (`getReceiptsByAccount`) に対処できます。これらのアクセスパターンをアプリケーションで使用することで、ユーザーは現在のサブスクリプションと過去 6 か月間の領収書を確認できます。このステップでは、テーブルスキーマに変更はありません。アクセスパターン 6 (`getSubscriptionsByAccount`) でサブスクリプション項目だけをターゲットにする方法は次の表で確認できます。



Primary key		Attributes									
Partition key: PK	Sort key: SK										
	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
ACC#123	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

すべてのアクセスパターンと各アクセスパターンにスキーマ設計で対処する方法を次の表にまとめています。

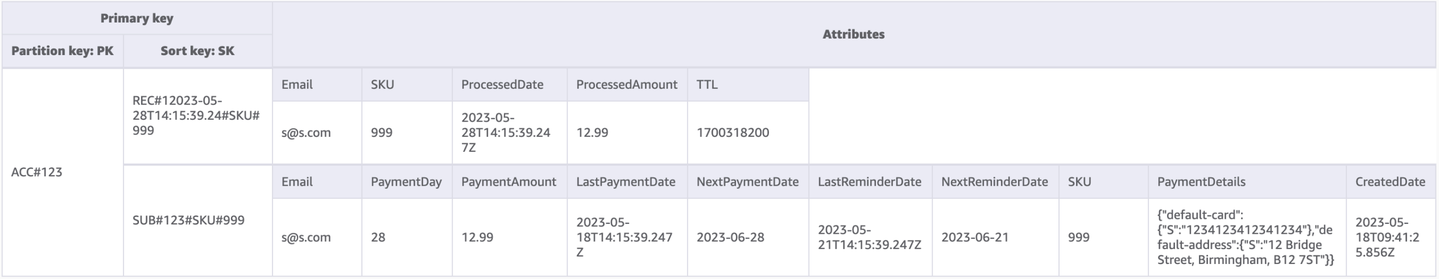
アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値
CreateSubscription	ベーステーブル	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	ベーステーブル	PutItem	ACC#account_id	REC#<RecieptDate>#SKU<SKUID>
updateSubscription	ベーステーブル	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
getDueRemindersByDate	GSI-1	Query	<NextReminderDate>	
getDuePaymentsByDate	GSI-2	Query	<NextPaymentDate>	

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値
getSubscriptionsByAccount	ベーステーブル	Query	ACC#account_id	SK begins_with “SUB#”
getReceiptsByAccount	ベーステーブル	Query	ACC#account_id	SK begins_with “REC#”

Recurring payments final schema

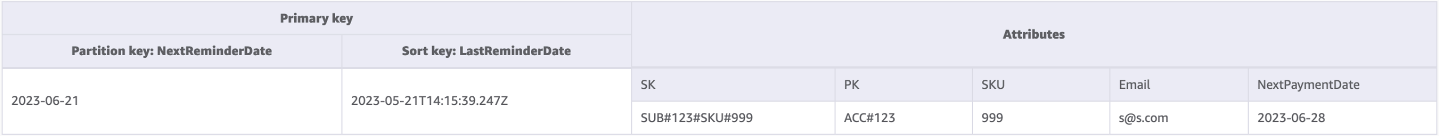
最終的なスキーマ設計は次のとおりです。このスキーマ設計を JSON ファイルとしてダウンロードするには、GitHub の DynamoDB の例を参照してください。

ベーステーブル



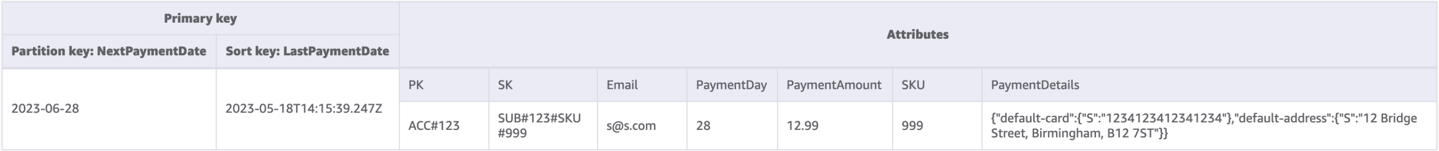
Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

GSI-1



Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate					
2023-06-21	2023-05-21T14:15:39.247Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

GSI-2



Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate							
2023-06-28	2023-05-18T14:15:39.247Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
		ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

このスキーマ設計での NoSQL Workbench の使用

この最終スキーマを、DynamoDB のデータモデリング、データ視覚化、クエリ開発機能を提供するビジュアルツールである NoSQL Workbench にインポートして、新しいプロジェクトを詳しく調べたり編集したりできます。使用を開始するには、次の手順に従います。

1. NoSQL Workbench をダウンロードします。詳細については、「the section called “ダウンロード”」を参照してください。
2. 上記の JSON スキーマファイルをダウンロードします。このファイルは既に NoSQL Workbench モデル形式になっています。
3. JSON スキーマファイルを NoSQL Workbench にインポートします。詳細については、「the section called “既存のモデルのインポート”」を参照してください。
4. NOSQL Workbench にインポートしたら、データモデルを編集できます。詳細については、「the section called “既存モデルの編集”」を参照してください。
5. データモデルの視覚化、サンプルデータの追加、CSV ファイルからのサンプルデータのインポートを行うには、NoSQL Workbench のデータビジュアライザー機能を使用します。

DynamoDB でのデバイスステータス更新のモニタリング

このユースケースでは、DynamoDB を使用して、DynamoDB 内でデバイスのステータスの更新 (またはデバイスのステータスの変更) をモニタリングする方法について説明しています。

ユースケース

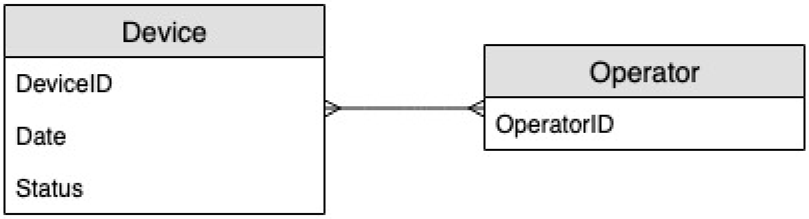
IoT のユースケース (スマートファクトリーなど) では、多くのデバイスをオペレーターがモニタリングする必要があり、オペレーターはステータスやログをモニタリングシステムに定期的に送信します。デバイスに問題が発生すると、デバイスのステータスが [正常] から [警告] に変わります。デバイスの異常な動作の重大度やタイプによって、さまざまなログレベルまたはステータスがあります。その後、システムはデバイスをチェックするオペレーターを割り当て、オペレーターは必要に応じて問題を上司にエスカレーションします。

このシステムの典型的なアクセスパターンには次のものがあります。

- デバイスのログエントリを作成する
- 特定のデバイスステータスのすべてのログを取得し、最新のログを最初に表示する
- 2 つの日付間の特定のオペレーターのすべてのログを取得する
- 特定のスーパーバイザーのエスカレーションされたログをすべて取得する
- 特定のスーパーバイザーの特定のデバイスステータスに関するすべてのエスカレーションログを取得
- 特定の日付における特定のスーパーバイザーの特定のデバイスステータスを含むすべてのエスカレーションされたログを取得する

エンティティ関係図

これは、デバイスステータスの更新をモニタリングするために使用するエンティティ関係図 (ERD) です。



アクセスパターン

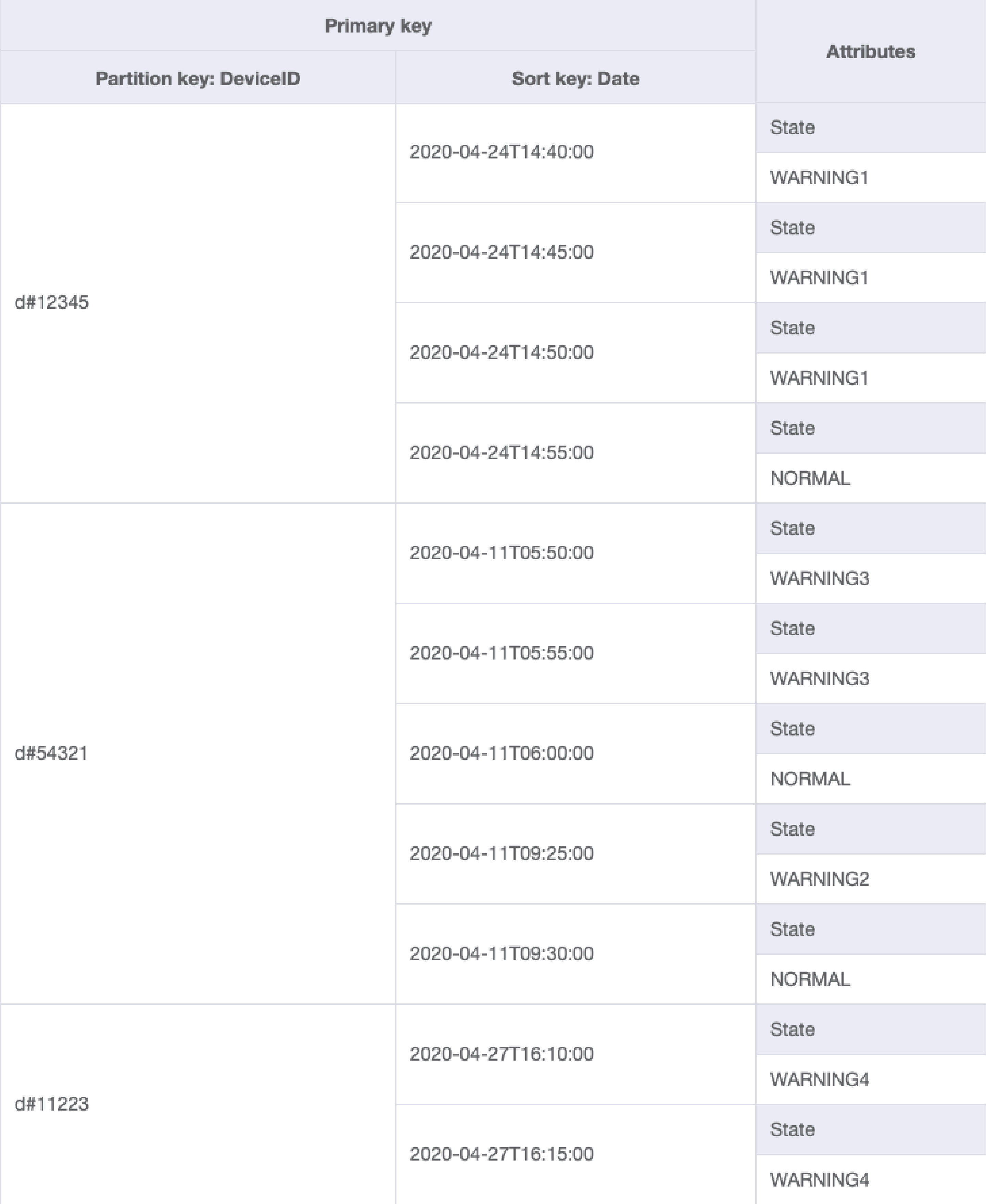
デバイスのステータス更新をモニタリングする方法について説明します。

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

スキーマ設計の進化

ステップ 1: アクセスパターン 1 (**`createLogEntryForSpecificDevice`**) と 2 (**`getLogsForSpecificDevice`**) に対処する

デバイス追跡システムのスケーリングの単位は、個々のデバイスです。このシステムでは、`deviceID` はデバイスを一意に識別します。これにより、`deviceID` がパーティションキーの適切な候補となります。各デバイスは定期的に (例えば 5 分おきに) 情報を追跡システムに送信します。この順序によって、日付が論理的なソート基準となり、したがってソートキーになります。この場合のサンプルデータは次のようになります。



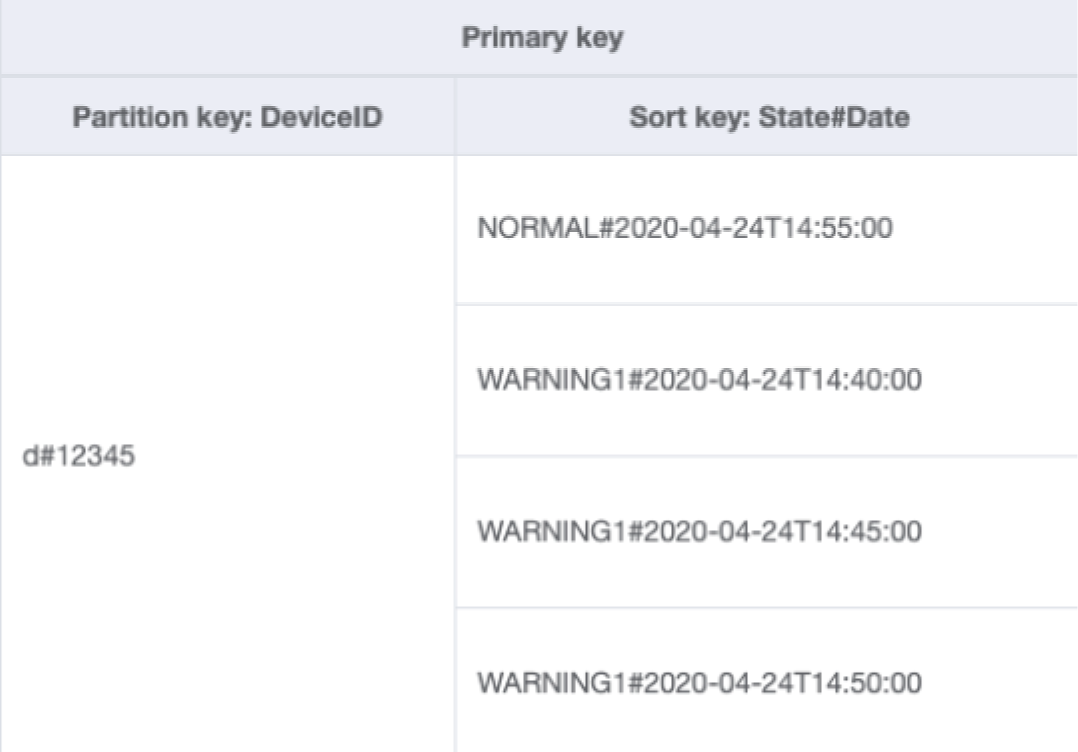
Primary key		Attributes
Partition key: DeviceID	Sort key: Date	
d#12345	2020-04-24T14:40:00	State WARNING1
	2020-04-24T14:45:00	State WARNING1
	2020-04-24T14:50:00	State WARNING1
	2020-04-24T14:55:00	State NORMAL
d#54321	2020-04-11T05:50:00	State WARNING3
	2020-04-11T05:55:00	State WARNING3
	2020-04-11T06:00:00	State NORMAL
	2020-04-11T09:25:00	State WARNING2
	2020-04-11T09:30:00	State NORMAL
d#11223	2020-04-27T16:10:00	State WARNING4
	2020-04-27T16:15:00	State WARNING4

特定のデバイスのログエントリを取得するには、パーティションキー DeviceID="d#12345" を使用してクエリオペレーションを実行します。

ステップ 2: アクセスパターン 3 (**getWarningLogsForSpecificDevice**) に対処する

State は非キー属性であるため、現在のスキーマでアクセスパターン 3 に対処するにはフィルター式が必要になります。DynamoDB では、フィルター式はキー条件式を使用してデータを読み取った後に適用されます。例えば、d#12345 の警告ログを取得する場合、パーティションキー DeviceID="d#12345" を使用したクエリオペレーションは、上記のテーブルから 4 つの項目を読み取り、警告ステータスを持つ 1 つの項目をフィルターで除外します。このアプローチは大規模な場合は効率的ではありません。フィルター式は、除外される項目の割合が低い場合や、クエリの実行頻度が低い場合に、クエリ対象の項目を除外するのに適した方法です。ただし、テーブルから多くの項目が取得され、その大半の項目がフィルターで除外される場合は、より効率的に実行されるようにテーブル設計を進化させ続けることができます。

複合ソートキーを使用して、このアクセスパターンを処理する方法を変更してみましょう。ソートキーを State#Date に変更したサンプルデータを DeviceStateLog_3.json からインポートできます。このソートキーは、属性 State、#、および Date から構成されます。この例では、# は区切り文字として使用されています。これで、データは次のようになります。

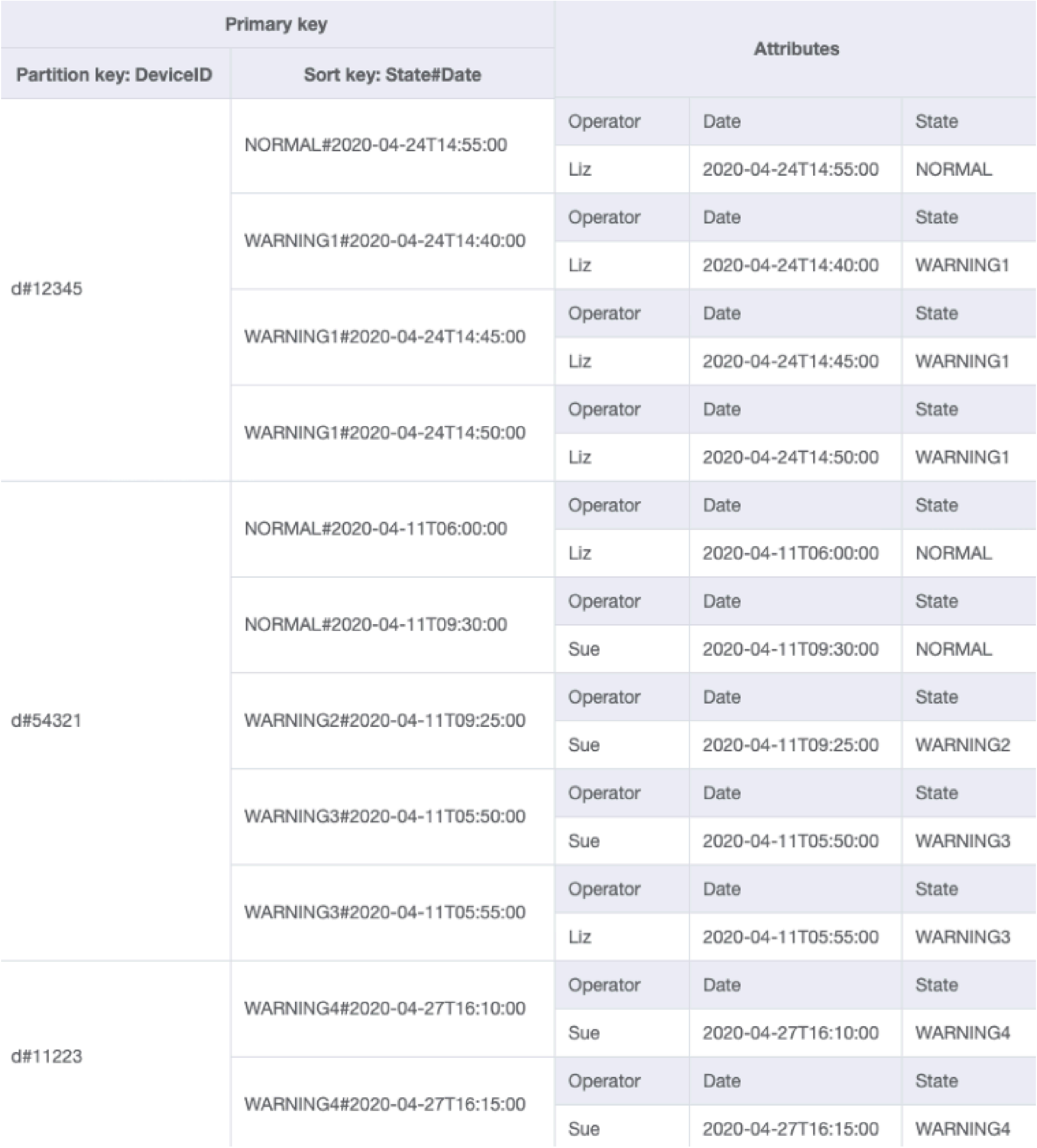


Primary key	
Partition key: DeviceID	Sort key: State#Date
d#12345	NORMAL#2020-04-24T14:55:00
	WARNING1#2020-04-24T14:40:00
	WARNING1#2020-04-24T14:45:00
	WARNING1#2020-04-24T14:50:00

デバイスの警告ログのみを取得する場合は、このスキーマの方がクエリの対象が絞り込まれます。クエリのキー条件にはパーティションキー DeviceID="d#12345" とソートキー State#Date begins_with "WARNING" が使用されます。このクエリは、警告ステータスの関連する 3 つの項目のみを読み取ります。

ステップ 3: アクセスパターン 4 (**getLogsForOperatorBetweenTwoDates**) に対処する

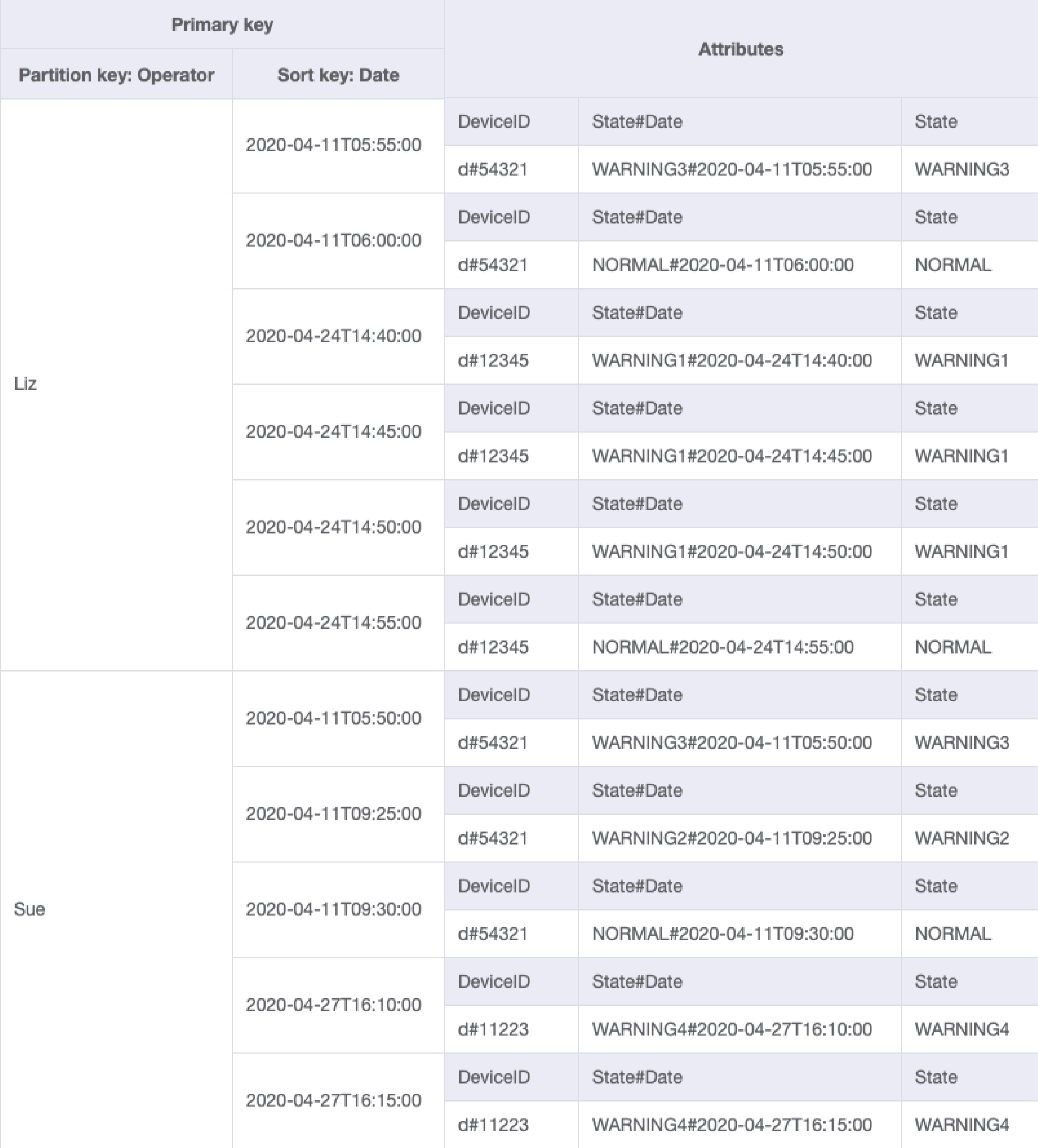
Operator 属性がサンプルデータとともに DeviceStateLog テーブルに追加された DeviceStateLog_4.jsonD をインポートできます。



Primary key		Attributes		
Partition key: DeviceID	Sort key: State#Date			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State
		Liz	2020-04-24T14:55:00	NORMAL
	WARNING1#2020-04-24T14:40:00	Operator	Date	State
		Liz	2020-04-24T14:40:00	WARNING1
	WARNING1#2020-04-24T14:45:00	Operator	Date	State
		Liz	2020-04-24T14:45:00	WARNING1
	WARNING1#2020-04-24T14:50:00	Operator	Date	State
		Liz	2020-04-24T14:50:00	WARNING1
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State
		Liz	2020-04-11T06:00:00	NORMAL
	NORMAL#2020-04-11T09:30:00	Operator	Date	State
		Sue	2020-04-11T09:30:00	NORMAL
	WARNING2#2020-04-11T09:25:00	Operator	Date	State
		Sue	2020-04-11T09:25:00	WARNING2
	WARNING3#2020-04-11T05:50:00	Operator	Date	State
		Sue	2020-04-11T05:50:00	WARNING3
	WARNING3#2020-04-11T05:55:00	Operator	Date	State
		Liz	2020-04-11T05:55:00	WARNING3
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State
		Sue	2020-04-27T16:10:00	WARNING4
	WARNING4#2020-04-27T16:15:00	Operator	Date	State
		Sue	2020-04-27T16:15:00	WARNING4

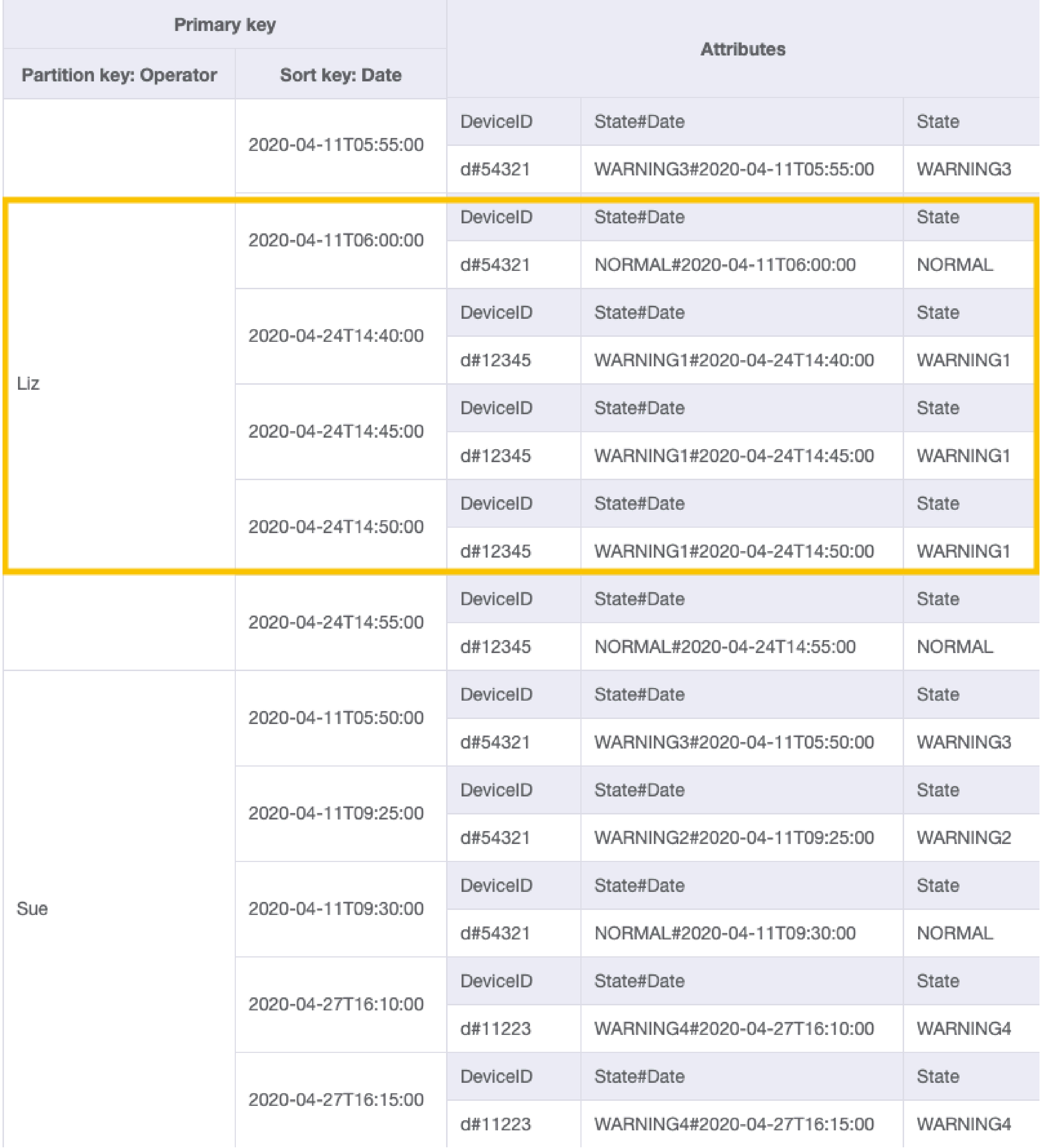
Operator は現在パーティションキーではないため、OperatorID に基づいてこのテーブルのキーと値を直接検索する方法はありません。OperatorID にグローバルセカンダリインデックスを使用

して新しい項目コレクションを作成する必要があります。アクセスパターンでは日付に基づく検索が必要なため、日付がグローバルセカンダリインデックス (GSI) のソートキー属性になります。GSI は現在次のようになっています。



Primary key		Attributes		
Partition key: Operator	Sort key: Date			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

アクセスパターン 4 (`getLogsForOperatorBetweenTwoDates`) の場合、`2020-04-11T05:58:00` と `2020-04-24T14:50:00` の間のパーティションキー `OperatorID=Liz` とソートキー `Date` を使用してこの GSI をクエリできます。



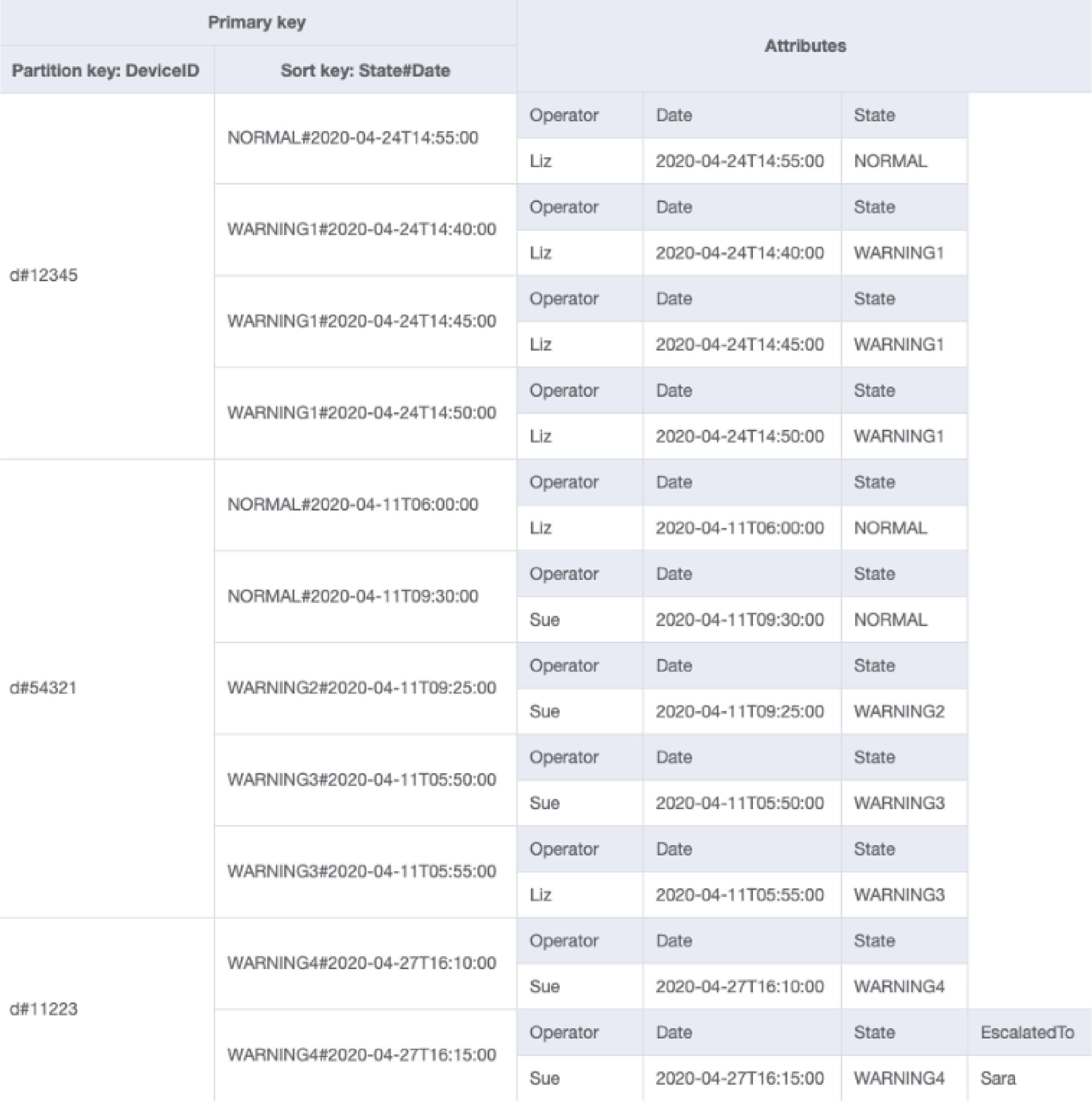
Primary key		Attributes		
Partition key: Operator	Sort key: Date			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

ステップ 4: アクセスパターン 5 (**getEscalatedLogsForSupervisor**)、6 (**getEscalatedLogsWithSpecificStatusForSupervisor**)、および 7 (**getEscalatedLogsWithSpecificStatusForSupervisorForDate**) に対処する

スパースインデックスを使用して、これらのアクセスパターンに対処します。

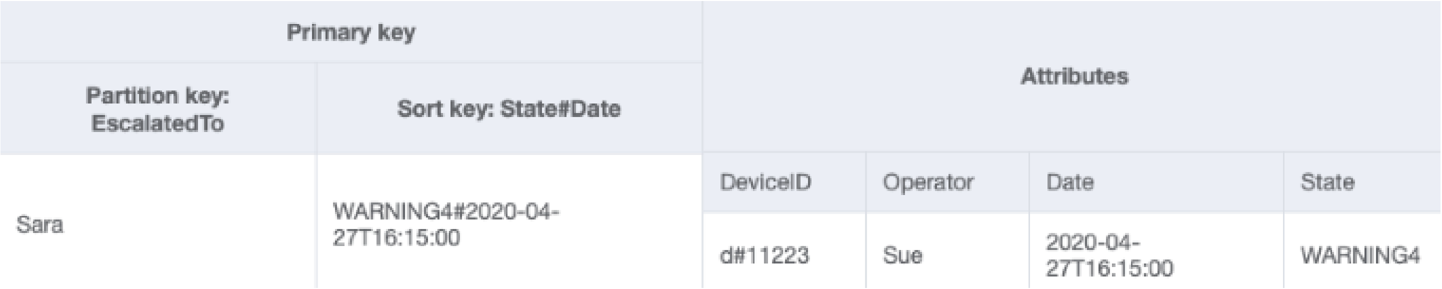
グローバルセカンダリインデックスはデフォルトではスパースであるため、インデックスのプライマリキー属性を含むベーステーブルの項目だけが実際にインデックスに表示されます。これは、モデル化するアクセスパターンに関係のない項目を除外するもう 1 つの方法です。

`EscalatedTo` 属性がサンプルデータとともに `DeviceStateLog` テーブルに追加された DeviceStateLog_6.json をインポートできます。前述のように、すべてのログがスーパーバイザーにエスカレーションされるわけではありません。



Primary key		Attributes			
Partition key: DeviceID	Sort key: State#Date				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

これで、EscalatedTo がパーティションキー、State#Date がソートキーである新しい GSI を作成できるようになりました。EscalatedTo 属性と State#Date 属性の両方を持つ項目のみがインデックスに表示されることに注意してください。



Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
		DeviceID	Operator	Date	State
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue	2020-04-27T16:15:00	WARNING4

その他のアクセスパターンは、以下のようにまとめられています。

すべてのアクセスパターンと各アクセスパターンにスキーマ設計で対処する方法を次の表にまとめています。

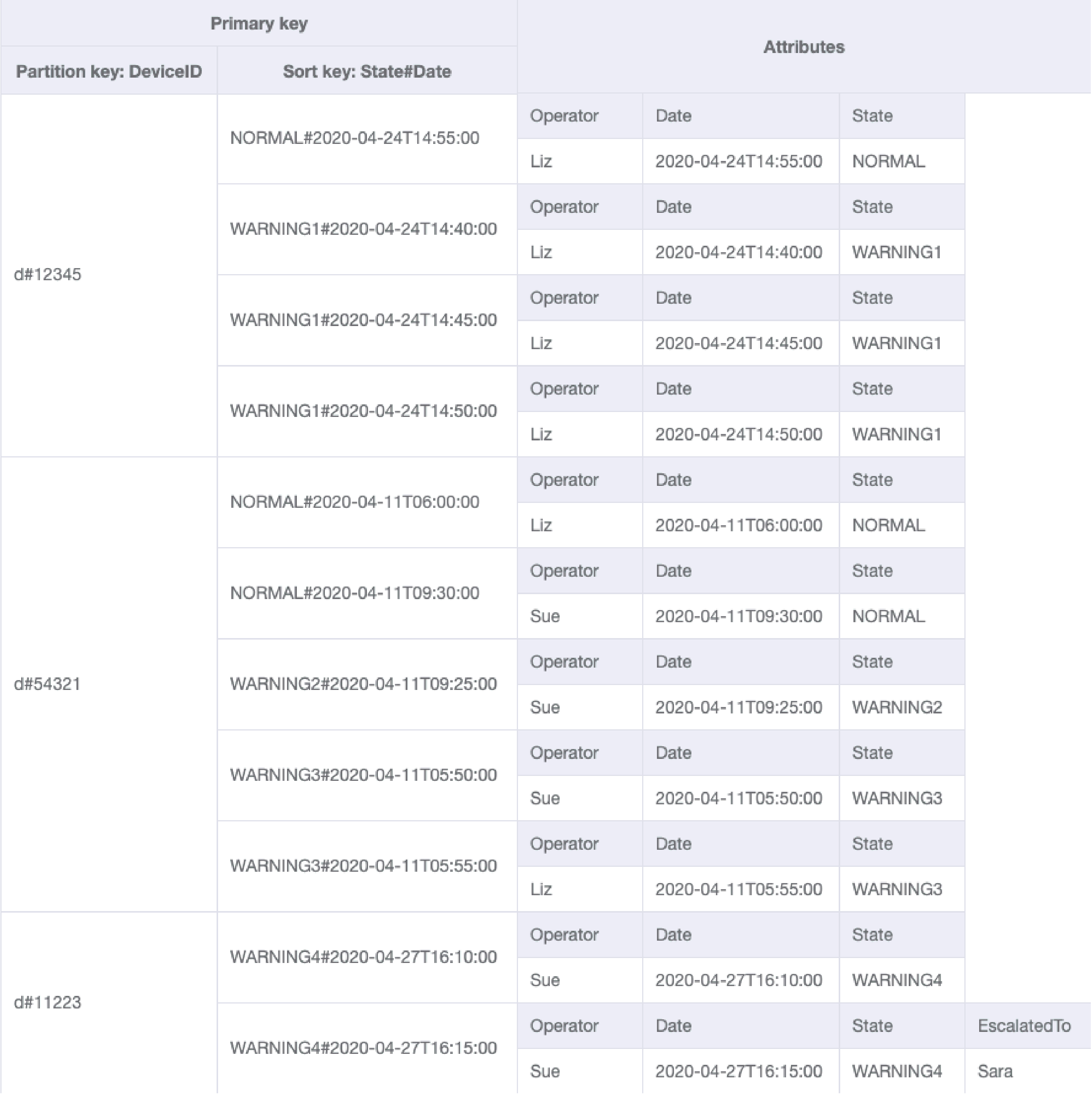
アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
createLogEntryForSpecificDevice	ベーステーブル	PutItem	DeviceID=deviceId	State#Date=state#date	
getLogsForSpecificDevice	ベーステーブル	Query	DeviceID=deviceId	State#Date begins_with "state1#"	ScanIndexForward = False
getWarningLogsForSpecificDevice	ベーステーブル	Query	DeviceID=deviceId	State#Date begins_with "WARNING"	
getLogsForOperatorBetweenTwoDates	GSI-1	Query	Operator=operatorName	Date between date1 and date2	
getEscalatedLogsForSupervisor	GSI-2	Query	EscalatedTo=supervisorName		

アクセスパターン	ベーステーブル/GSI/LSI	操作	パーティションキー値	ソートキー値	その他の条件/フィルター
getEscalatedLogsWithSpecificStatusForSupervisor	GSI-2	Query	EscalatedTo=supervisorName	State#Date begins_with "state1#"	
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Query	EscalatedTo=supervisorName	State#Date begins_with "state1#date1"	

最終スキーマ

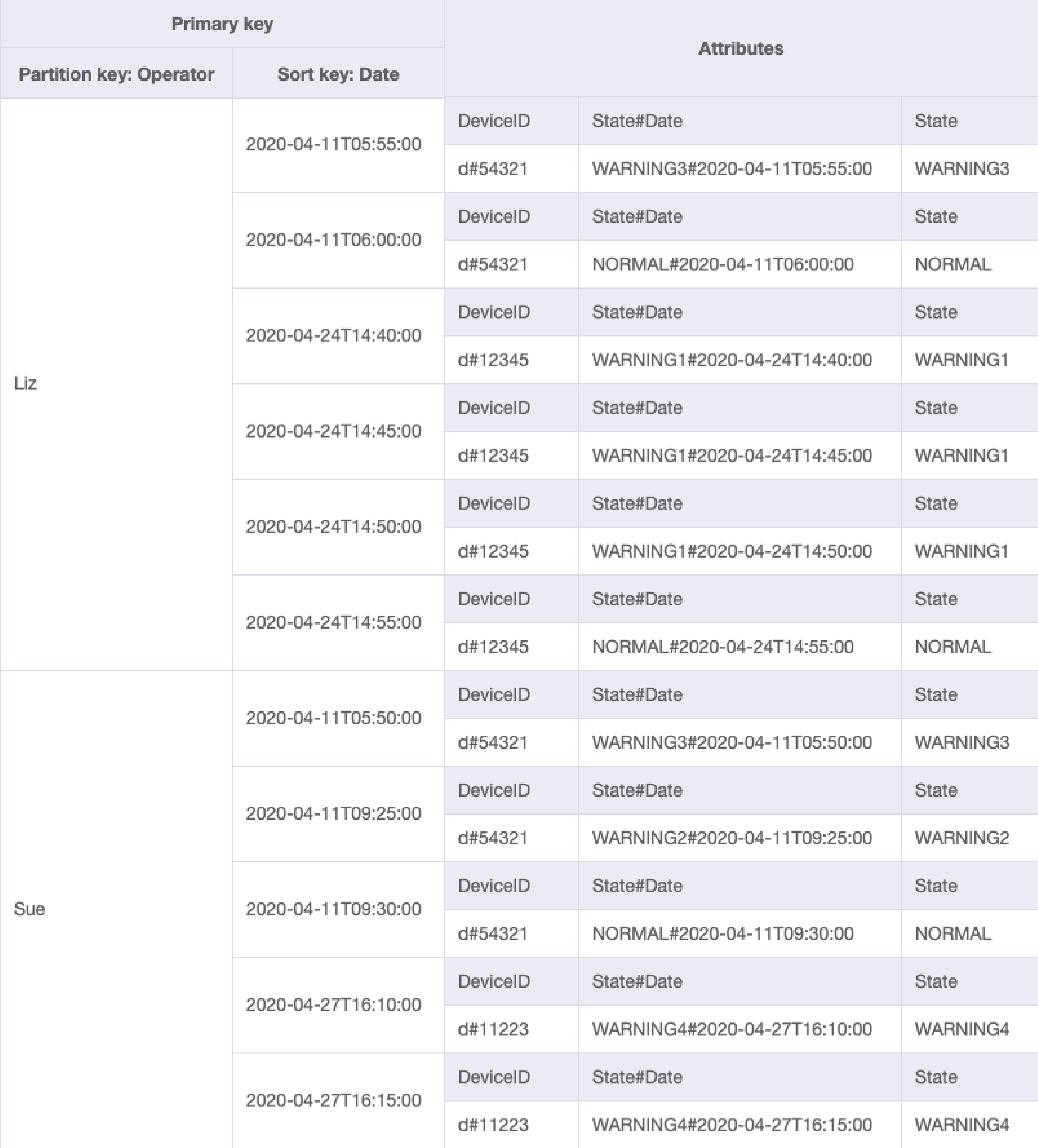
最終的なスキーマ設計は次のとおりです。このスキーマ設計を JSON ファイルとしてダウンロードするには、GitHub の DynamoDB の例を参照してください。

ベーステーブル



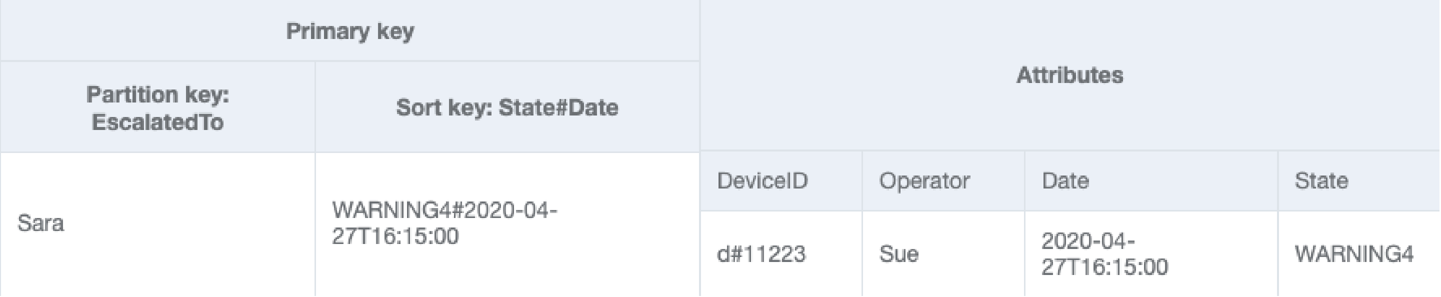
Primary key		Attributes			
Partition key: DeviceID	Sort key: State#Date				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

GSI-1



Primary key		Attributes		
Partition key: Operator	Sort key: Date			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

GSI-2



Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator	Date	State
		d#11223	Sue	2020-04-27T16:15:00	WARNING4

このスキーマ設計での NoSQL Workbench の使用

この最終スキーマを、DynamoDB のデータモデリング、データ視覚化、クエリ開発機能を提供するビジュアルツールである NoSQL Workbench にインポートして、新しいプロジェクトを詳しく調べたり編集したりできます。使用を開始するには、次の手順に従います。

1. NoSQL Workbench をダウンロードします。詳細については、「the section called “ダウンロード”」を参照してください。
2. 上記の JSON スキーマファイルをダウンロードします。このファイルは既に NoSQL Workbench モデル形式になっています。
3. JSON スキーマファイルを NoSQL Workbench にインポートします。詳細については、「the section called “既存のモデルのインポート”」を参照してください。
4. NOSQL Workbench にインポートしたら、データモデルを編集できます。詳細については、「the section called “既存モデルの編集”」を参照してください。
5. データモデルの視覚化、サンプルデータの追加、CSV ファイルからのサンプルデータのインポートを行うには、NoSQL Workbench のデータビジュアライザー機能を使用します。

DynamoDB をオンラインショップのデータストアとして使用する

このユースケースでは、オンラインショップ (または e ストア) のデータストアとして DynamoDB を使用する方法について説明しています。

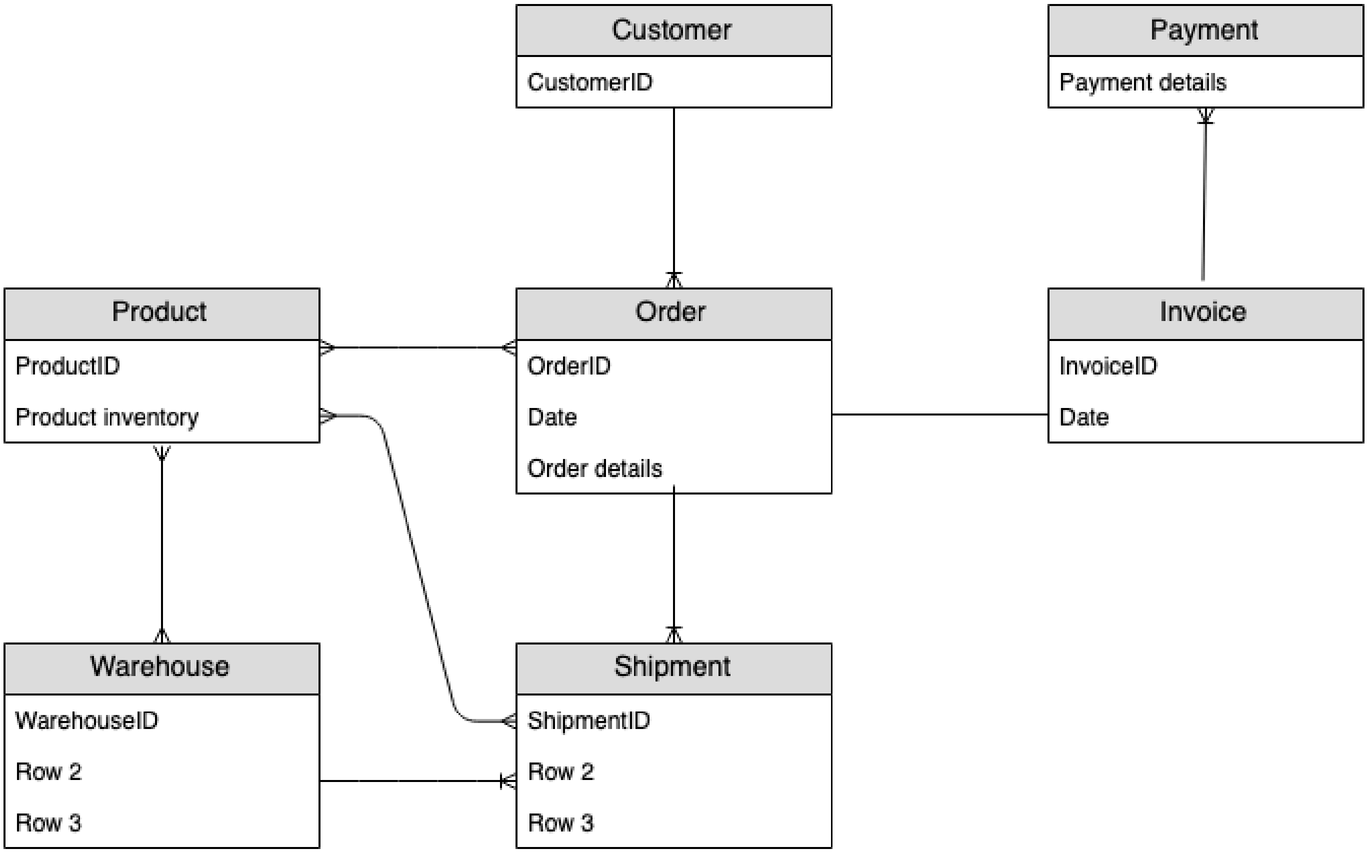
ユースケース

オンラインストアでは、ユーザーはさまざまな製品を閲覧し、最終的に購入することができます。生成された請求書に基づいて、お客様はディスカウントコードまたはギフトカードを使用して支払い、残りの金額をクレジットカードで支払うことができます。購入した商品は、複数の倉庫から集荷され、指定された住所に発送されます。オンラインストアの一般的なアクセスパターンは以下のとおりです。

- 指定された customerId のお客様を取得する
- 指定された productId の製品を取得する
- 指定された warehouseId の倉庫を取得する
- productId で全倉庫の商品在庫を取得する
- 指定された OrderId の注文を取得する
- 指定された OrderID のすべての商品を取得する
- 指定された OrderID の請求書を取得する
- 指定された OrderID のすべての出荷情報を取得する
- 指定された日付範囲における指定された productId のすべての注文を取得する
- 指定された invoiceId の請求書を取得する
- 指定された InvoiceID のすべての支払いを取得する
- 指定された shipmentId の出荷詳細を取得する
- 指定された warehouseId のすべての出荷情報を取得する
- 指定された warehouseId のすべての商品の在庫を取得する
- 指定された日付範囲における指定された customerId のすべての請求書を取得する
- 指定された日付範囲で指定された customerId で注文されたすべての製品を取得する

エンティティ関係図

これは、オンラインショップのデータストアとして DynamoDB をモデル化するために使用するエンティティ関係図 (ERD) です。



アクセスパターン

以上が、DynamoDB をオンラインショップのデータストアとして利用する場合のアクセスパターンです。

1. `getCustomerByCustomerId`
2. `getProductByProductId`
3. `getWarehouseByWarehouseId`
4. `getProductInventoryByProductId`
5. `getOrderDetailsByOrderId`
6. `getProductByOrderId`
7. `getInvoiceByOrderId`
8. `getShipmentByOrderId`
9. `getOrderByProductIdForDateRange`
10. `getInvoiceByInvoiceId`

11getPaymentByInvoiceId

12getShipmentDetailsByShipmentId

13getShipmentByWarehouseId

14getProductInventoryByWarehouseId

15getInvoiceByCustomerIdForDateRange

16getProductsByCustomerIdForDateRange

スキーマ設計の進化

DynamoDB 用の NoSQL Workbench を使用して、AnOnlineShop_1.json をインポートし、AnOnlineShop という新しいデータモデルと OnlineShop という新しいテーブルを作成します。パーティションキーとソートキーには一般的な名前 PK と SK を使用することに注意してください。これは、さまざまなタイプのエンティティを同じテーブルに保存するための方法です。

ステップ 1: アクセスパターン 1 (**getCustomerByCustomerId**) に対処する

AnOnlineShop_2.json をインポートして、アクセスパターン 1 (getCustomerByCustomerId) を処理します。エンティティの中には他のエンティティとリレーションシップを持たないものもあるため、それらには同じ値の PK と SK を使用します。データ例では、customerId を後で追加される他のエンティティから区別するために、キーに接頭辞 c# が使われていることに注意してください。この方法は他のエンティティでも繰り返されます。

このアクセスパターンに対処するために、GetItem オペレーションを PK=customerId と SK=customerId で使用できます。

ステップ 2: アクセスパターン 2 (**getProductByProductId**) に対処する

AnOnlineShop_3.json をインポートして、product エンティティのアクセスパターン 2 (getProductByProductId) に対処します。製品エンティティには p# というプレフィックスが付けられ、同じソートキー属性が customerID と productID の保存にも使われています。一般的な命名規則と垂直パーティショニングによって、このような項目コレクションを作成し、効果的な単一テーブル設計を実現できます。

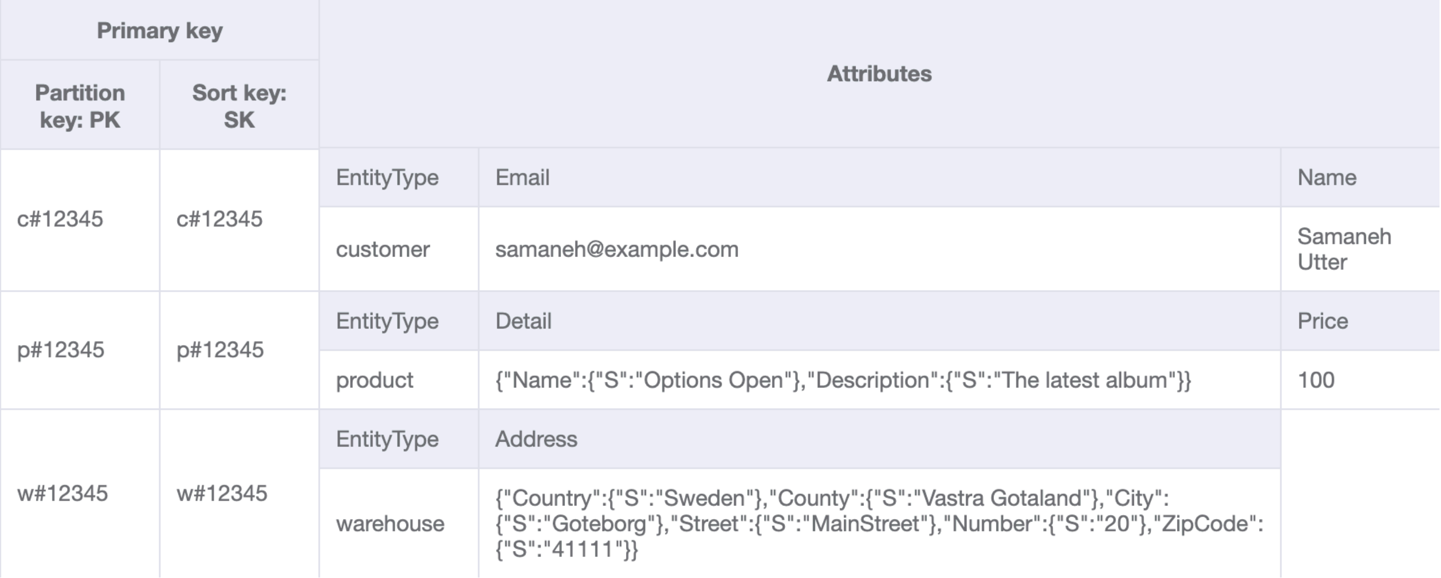
このアクセスパターンに対処するため、GetItem オペレーションは PK=productId および SK=productId で使用できます。

ステップ 3: アクセスパターン 3 (**getWarehouseByWarehouseId**) に対処する

AnOnlineShop_4.json をインポートして、warehouse エンティティのアクセスパターン 3 (getWarehouseByWarehouseId) に対処します。現在、customer、product、および warehouse エンティティが同じテーブルに追加されています。これらはプレフィックスと EntityType 属性を使用して区別されます。タイプ属性 (またはプレフィックスの命名) は読みやすさを向上させます。さまざまなエンティティの英数字 ID を同じ属性に保存するだけでは、読みやすさに影響が出ます。これらの識別子がないと、あるエンティティを別のエンティティと区別するのが難しくなります。

このアクセスパターンに対処するため、GetItem オペレーションは PK=warehouseId および SK=warehouseId で使用できます。

ベーステーブル:



Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
p#12345	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

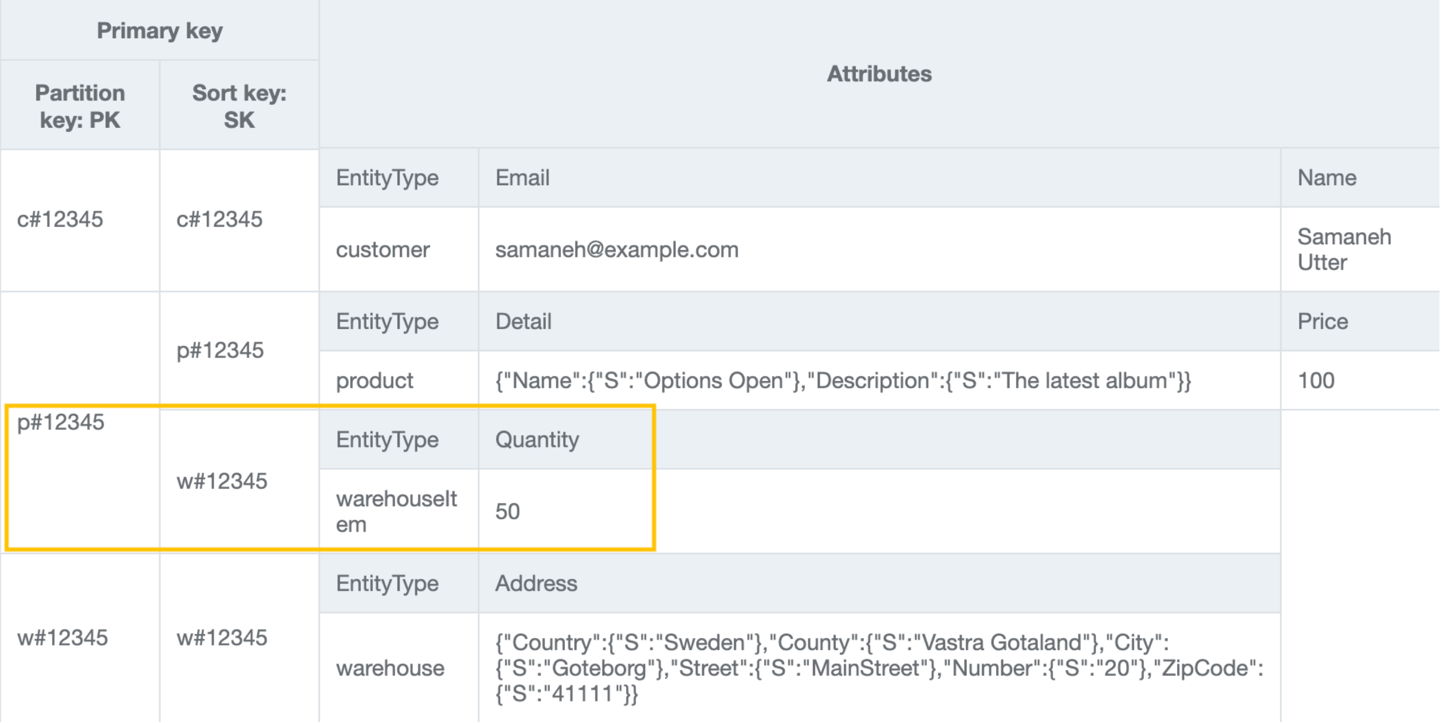
ステップ 4: アクセスパターン 4 (**getProductInventoryByProductId**) に対処する

AnOnlineShop_5.json をインポートしてアクセスパターン 4 (getProductInventoryByProductId) に対処します。warehouseItem エンティティは、各倉庫の商品数を追跡するために使用されます。この項目は通常、倉庫に商品が追加されたり、倉庫から商品が削除されたりすると更新されます。ERD にあるように、product と warehouse の間には多対多リレーションシップがあります。ここで、product から warehouse への 1 対多リレーションシップは warehouseItem のようにモデル化されます。その後、warehouse から product への 1 対多リレーションシップもモデル化されます。

アクセスパターン 4 は、PK=ProductId および SK begins_with “w#“ に対するクエリで対処できます。

ソートキーに適用できる begins_with() およびその他の式の詳細については、「キー条件式」を参照してください。

ベーステーブル:



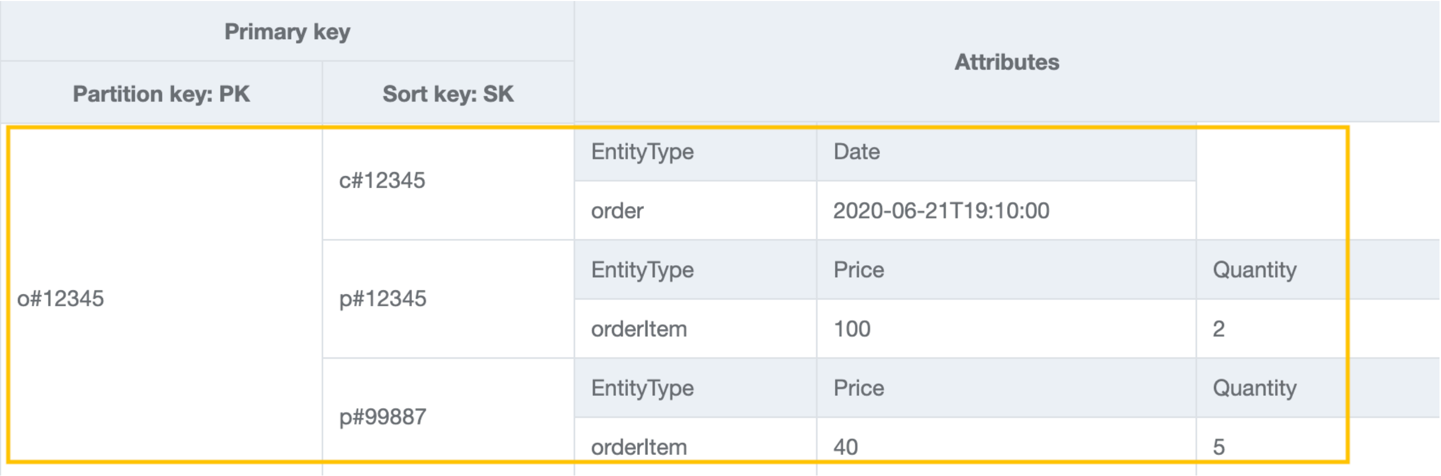
Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
p#12345	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
	w#12345	EntityType	Quantity	
		warehouseItem	50	
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

ステップ 5: アクセスパターン 5 (**getOrderDetailsByOrderId**) と 6 (**getProductByOrderId**) に対処する

AnOnlineShop_6.json をインポートして、さらに customer、product、および warehouse の項目をテーブルに追加します。次に、AnOnlineShop_7.json をインポートして、アクセスパターン 5 (getOrderDetailsByOrderId) および 6 (getProductByOrderId) に対処できる order 用項目コレクションを構築します。orderItem エンティティとしてモデル化された order と product の間の 1 対多リレーションシップを確認できます。

アクセスパターン 5 (getOrderDetailsByOrderId) に対処するには、PK=orderId を使用してテーブルをクエリします。これにより、customerId および注文された製品を含む注文に関するすべての情報が提供されます。

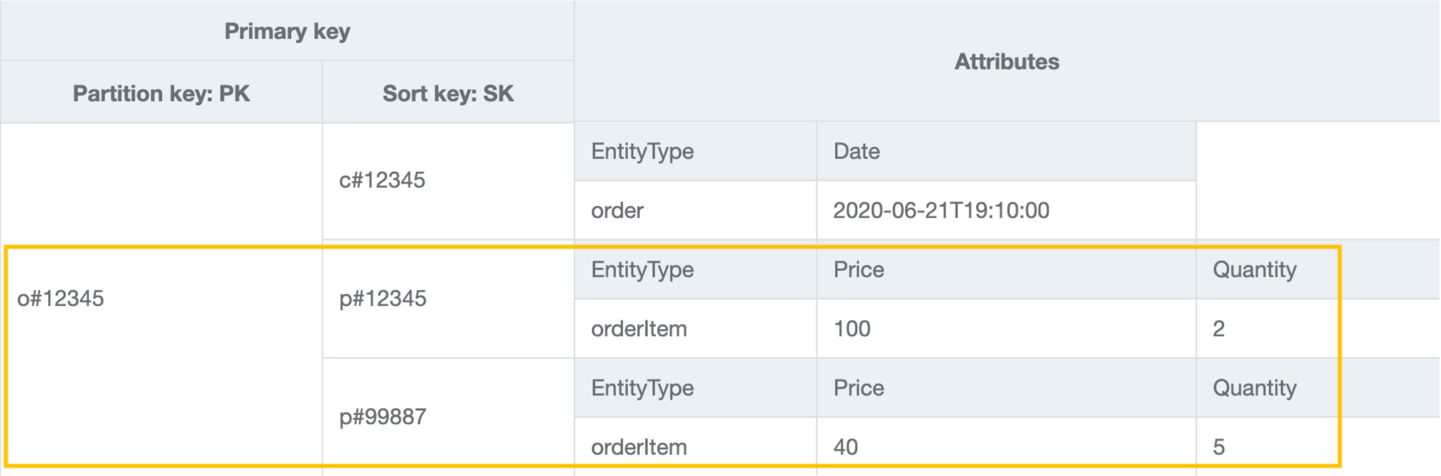
ベーステーブル:



Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

アクセスパターン 6 (`getProductByOrderId`) に対処するには、`order` 内の製品のみを読み取る必要があります。これを達成するために、`PK=orderId` および `SK begins_with “p#”` を使用してテーブルをクエリします。

ベーステーブル:

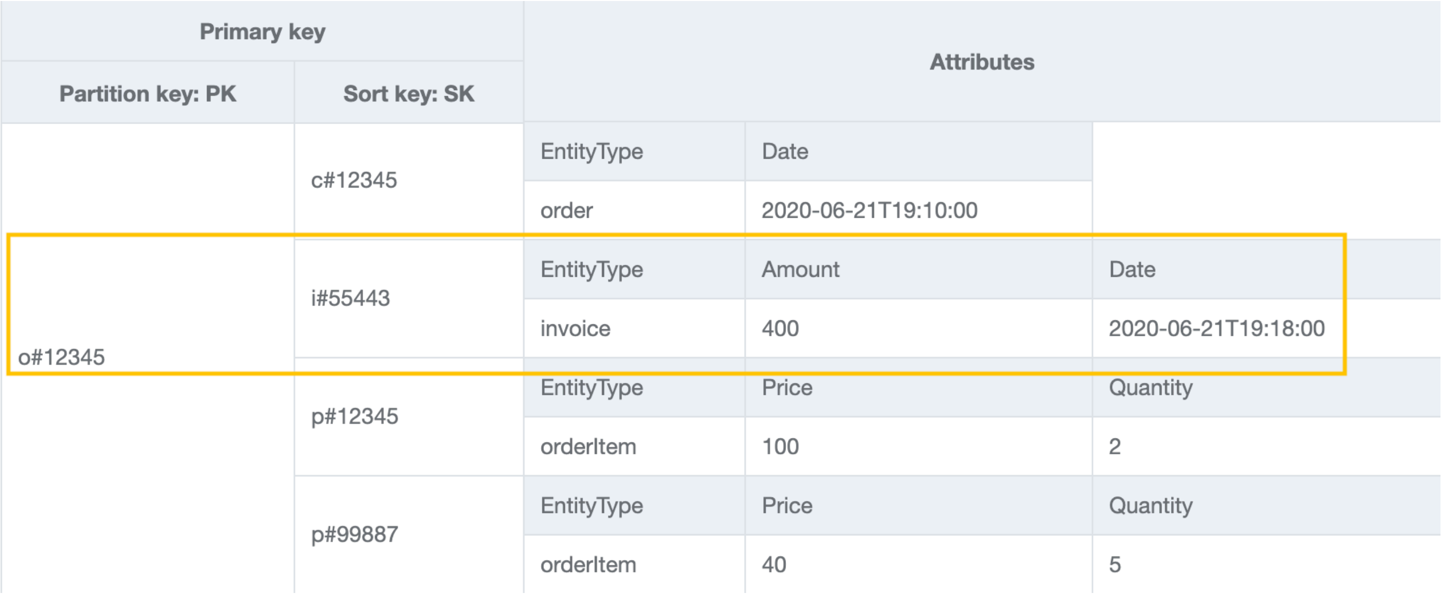


Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

ステップ 6: アクセスパターン 7 (**`getInvoiceByOrderId`**) に対処する

AnOnlineShop_8.json をインポートして、アクセスパターン 7 (`getInvoiceByOrderId`) を処理する注文項目コレクションに `invoice` エンティティを追加します。このアクセスパターンに対処するために、`PK=orderId` および `SK begins_with “i#”` でクエリオペレーションを使用できます。

ベーステーブル:



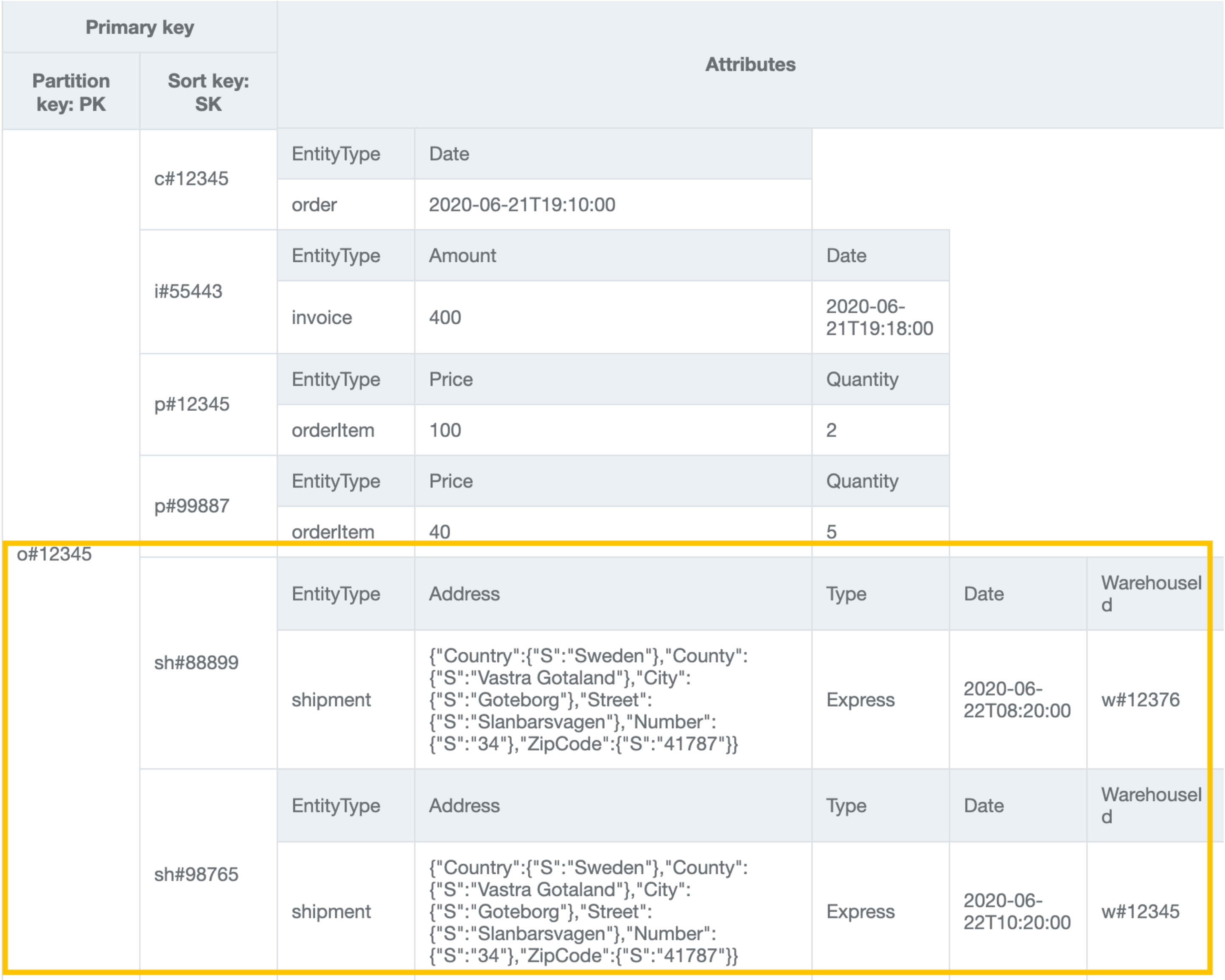
Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	i#55443	EntityType	Amount	Date
		invoice	400	2020-06-21T19:18:00
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

ステップ 7: アクセスパターン 8 (**`getShipmentByOrderId`**) に対処する

AnOnlineShop_9.json をインポートして、アクセスパターン 8 (`getShipmentByOrderId`) を処理する注文項目コレクションに `shipment` エンティティを追加します。単一テーブルの設計にさらに多くの種類のエンティティを追加することで、同じ垂直分割モデルを拡張しています。注文項目コレクションに、`order` エンティティが `shipment`、`orderItem`、および `invoice` エンティティと持つさまざまな関係がどのように含まれているかに注意してください。

`orderId` による出荷を取得するには、`PK=orderId` と `SK begins_with “sh#”` でクエリオペレーションを実行します。

ベーステーブル:



Primary key		Attributes				
Partition key: PK	Sort key: SK					
o#12345	c#12345	EntityType	Date			
		order	2020-06-21T19:10:00			
	i#55443	EntityType	Amount	Date		
		invoice	400	2020-06-21T19:18:00		
	p#12345	EntityType	Price	Quantity		
		orderItem	100	2		
	p#99887	EntityType	Price	Quantity		
		orderItem	40	5		
	sh#88899	EntityType	Address	Type	Date	WarehouseId
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00	w#12376
	sh#98765	EntityType	Address	Type	Date	WarehouseId
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00	w#12345

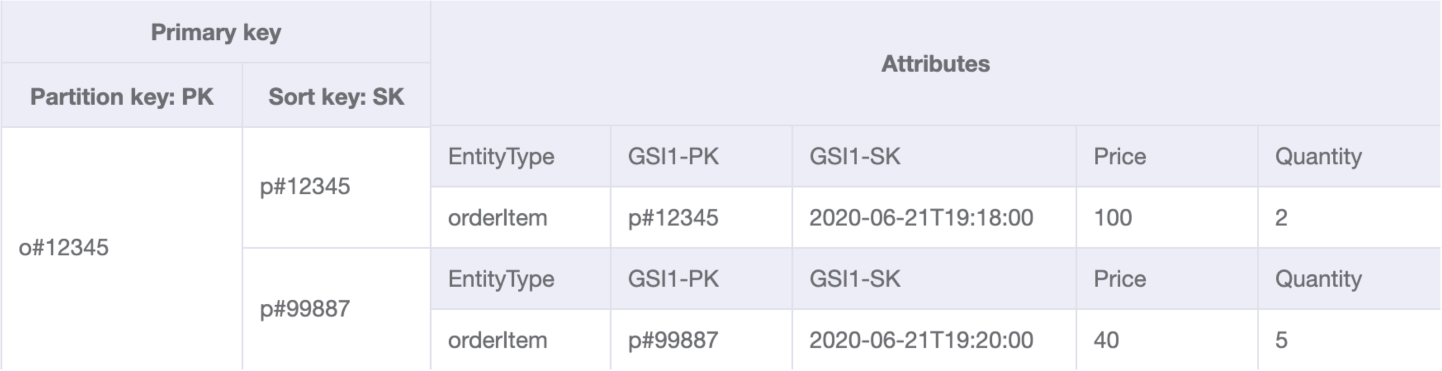
ステップ 8: アクセスパターン 9 (`getOrderByProductIdForDateRange`) に対処する

前のステップで、注文項目コレクションを作成しました。このアクセスパターンには新しいルックアップディメンション (`ProductID` および `Date`) があり、テーブル全体をスキャンし、関連するレコードをフィルタリングして、ターゲット項目を取得する必要があります。このアクセスパターンに対処するには、グローバルセカンダリインデックス (GSI) を作成する必要があります。AnOnlineShop_10.json をインポートし、複数の注文項目コレクションから `orderItem` データを取得できる GSI を使用して、新しい項目コレクションを作成します。データには `GSI1-PK` と `GSI1-SK` が含まれ、それぞれ `GSI1` のパーティションキーとソートキーになります。

DynamoDB は、GSI のキー属性を含む項目をテーブルから GSI に自動的に入力します。GSI に追加の挿入を手動で行う必要はありません。

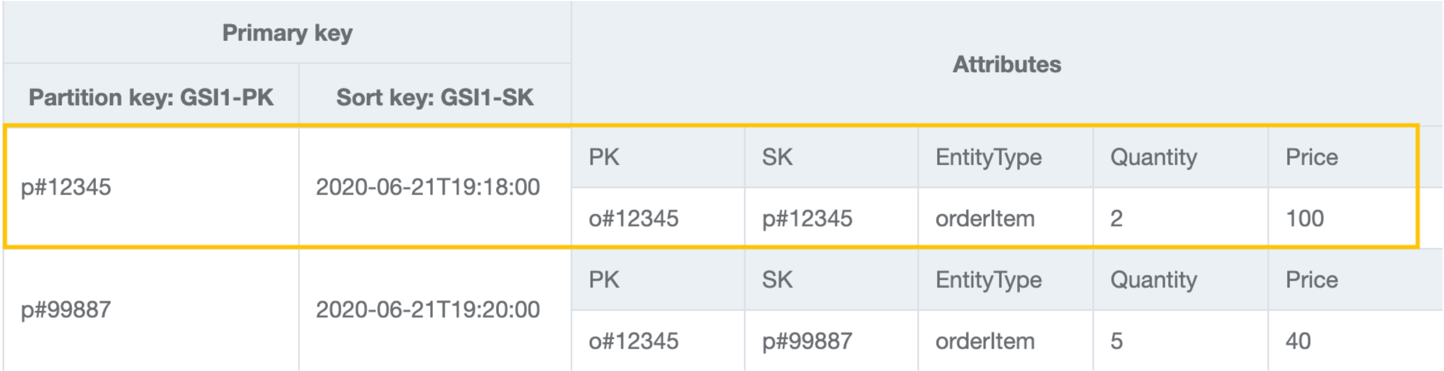
アクセスパターン 9 に対処するには、`GSI1-PK=productId` および `GSI1SK between (date1, date2)` を使用して `GSI1` でクエリを実行します。

ベーステーブル:



Primary key		Attributes				
Partition key: PK	Sort key: SK					
o#12345	p#12345	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#12345	2020-06-21T19:18:00	100	2
	p#99887	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#99887	2020-06-21T19:20:00	40	5

GSI1:



Primary key		Attributes				
Partition key: GSI1-PK	Sort key: GSI1-SK					
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#12345	orderItem	2	100
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#99887	orderItem	5	40

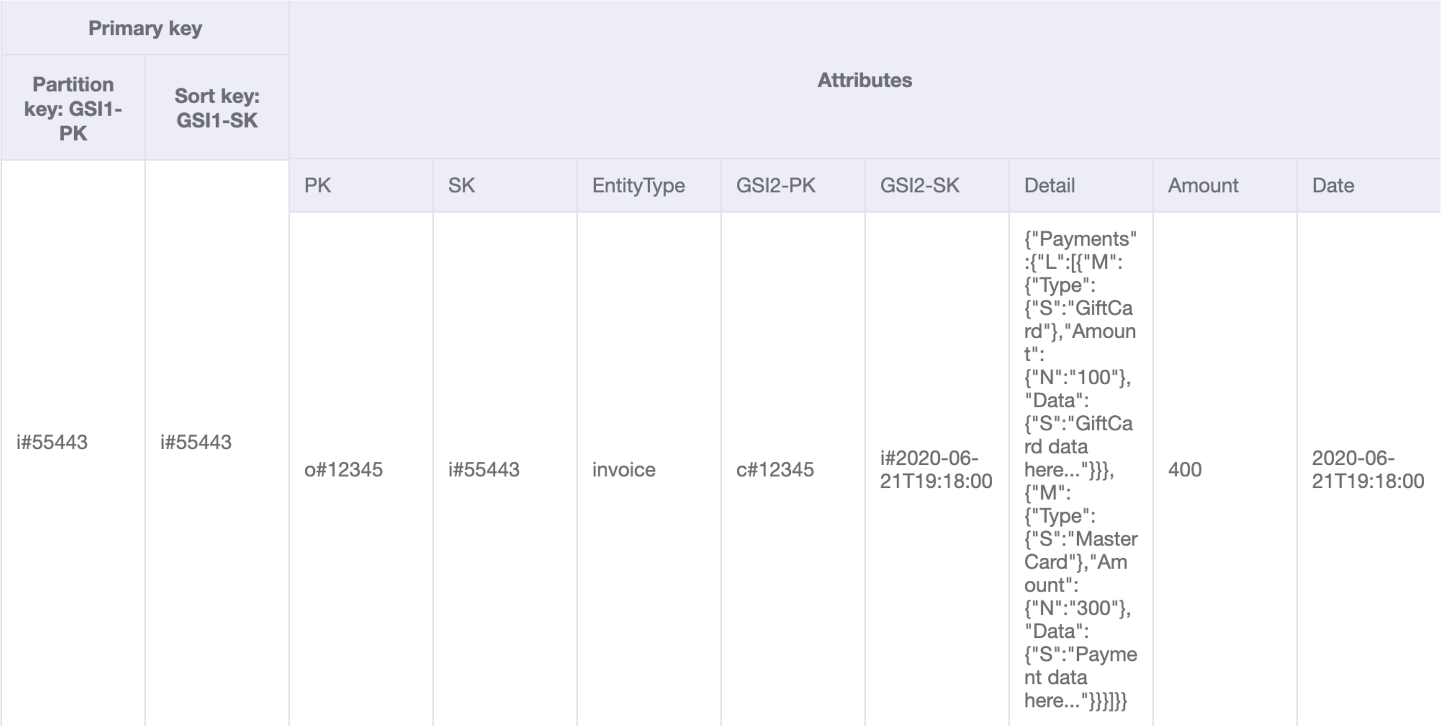
ステップ 9: アクセスパターン 10 (**getInvoiceByInvoiceId**) と 11 (**getPaymentByInvoiceId**) に対処する

AnOnlineShop_11.json をインポートして、アクセスパターン 10 (`getInvoiceByInvoiceId`) と 11 (`getPaymentByInvoiceId`) に対処します。どちらも `invoice` に関連しています。これらは 2 つの異なるアクセスパターンですが、同じキー条件を使用して実現されます。`Payments` は、`invoice` エンティティのマップデータタイプを持つ属性として定義されます。

> ⓘ Note
>
> `GSI1-PK` と `GSI1-SK` は、さまざまなエンティティに関する情報を保存するためにオーバーロードされているため、同じ GSI から複数のアクセスパターンに対応できます。GSI オーバーロードの詳細については、「グローバルセカンダリインデックスの多重定義」を参照してください。

アクセスパターン 10 と 11 に対処するには、`GSI1-PK=invoiceId` および `GSI1-SK=invoiceId` を使用して `GSI1` に対してクエリを実行します。

GSI1:



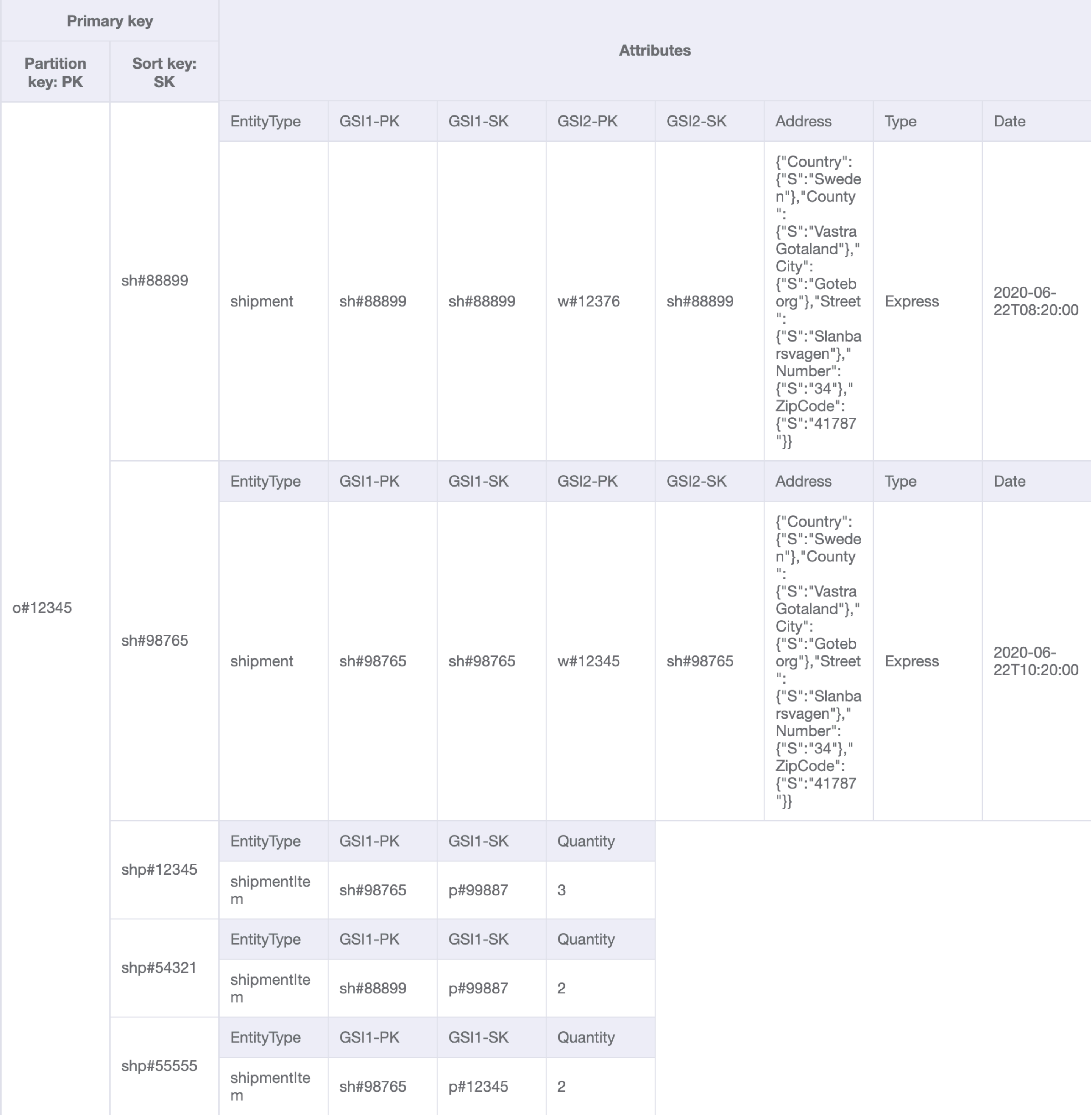
Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK								
		PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
i#55443	i#55443	o#12345	i#55443	invoice	c#12345	i#2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00

ステップ 10: アクセスパターン 12 (**`getShipmentDetailsByShipmentId`**) と 13 (**`getShipmentByWarehouseId`**) に対処する

AnOnlineShop_12.json をインポートして、アクセスパターン 12 (`getShipmentDetailsByShipmentId`) と 13 (`getShipmentByWarehouseId`) に対処します。

1 回のクエリオペレーションで注文に関するすべての詳細を取得できるように、`shipmentItem` エンティティがベーステーブルの注文項目コレクションに追加されることに注意してください。

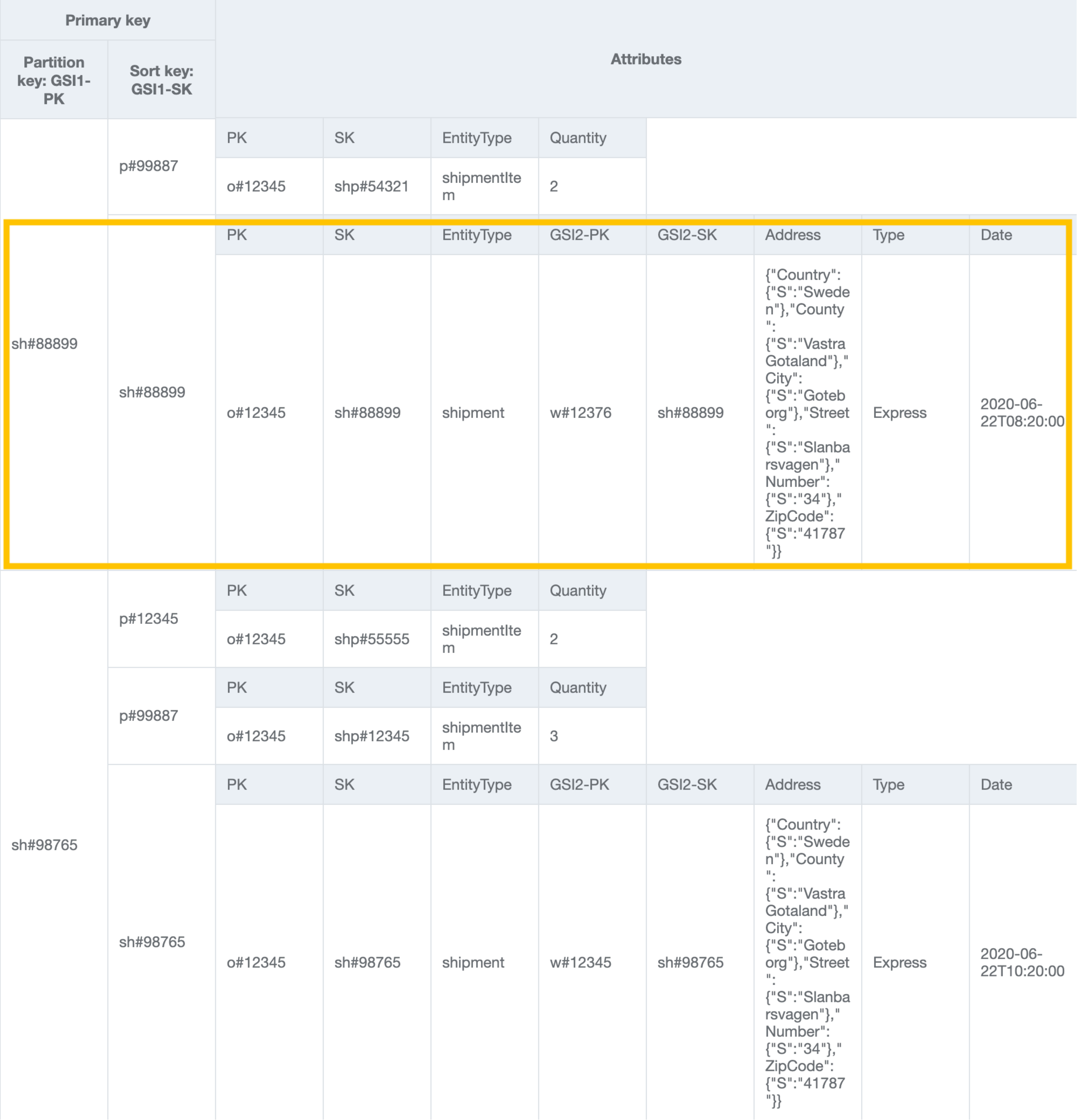
ベーステーブル:



Primary key		Attributes							
Partition key: PK	Sort key: SK								
o#12345	sh#88899	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#88899	sh#88899	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
	sh#98765	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#98765	sh#98765	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00
	shp#12345	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#99887	3				
	shp#54321	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#88899	p#99887	2				
	shp#55555	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#12345	2				

GSI1 パーティションとソートキーは、`shipment` と `shipmentItem` の間の 1 対多リレーションシップをモデル化するために既に使用されています。アクセスパターン 12 (`getShipmentDetailsByShipmentId`) に対処するには、`GSI1-PK=shipmentId` および `GSI1-SK=shipmentId` を使用して `GSI1` に対してクエリを実行します。

GSI1:

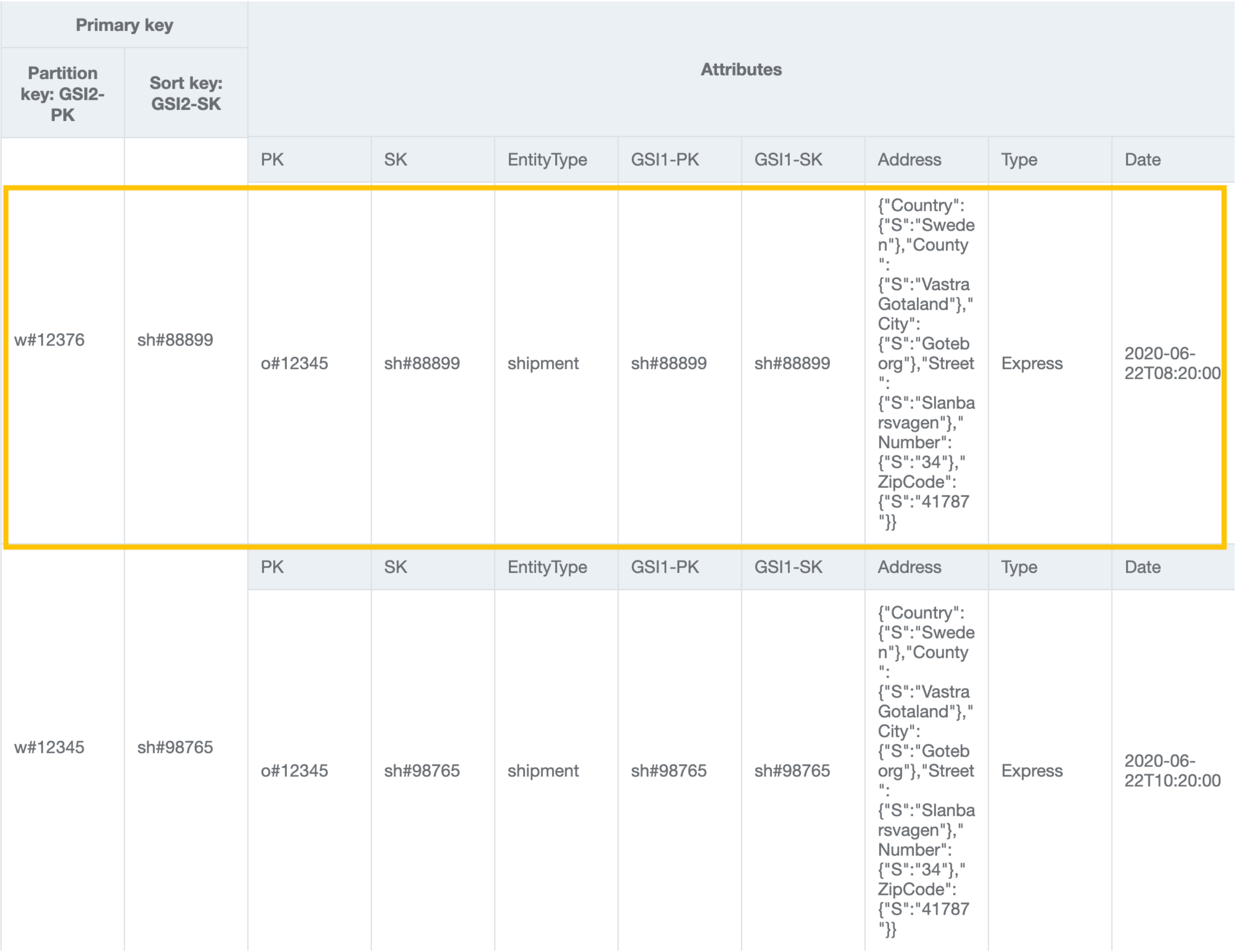


Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK								
sh#88899	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
sh#98765	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

アクセスパターン 13 (getShipmentByWarehouseId) の warehouse と shipment の間の新しい 1 対多リレーションシップをモデル化するには、別の GSI (GSI2) を作成する必要があります。アク

セスパターンに対処するには、GSI2-PK=warehouseId および GSI2-SK begins_with “sh#” を使用して GSI2 に対してクエリを実行します。

GSI2:

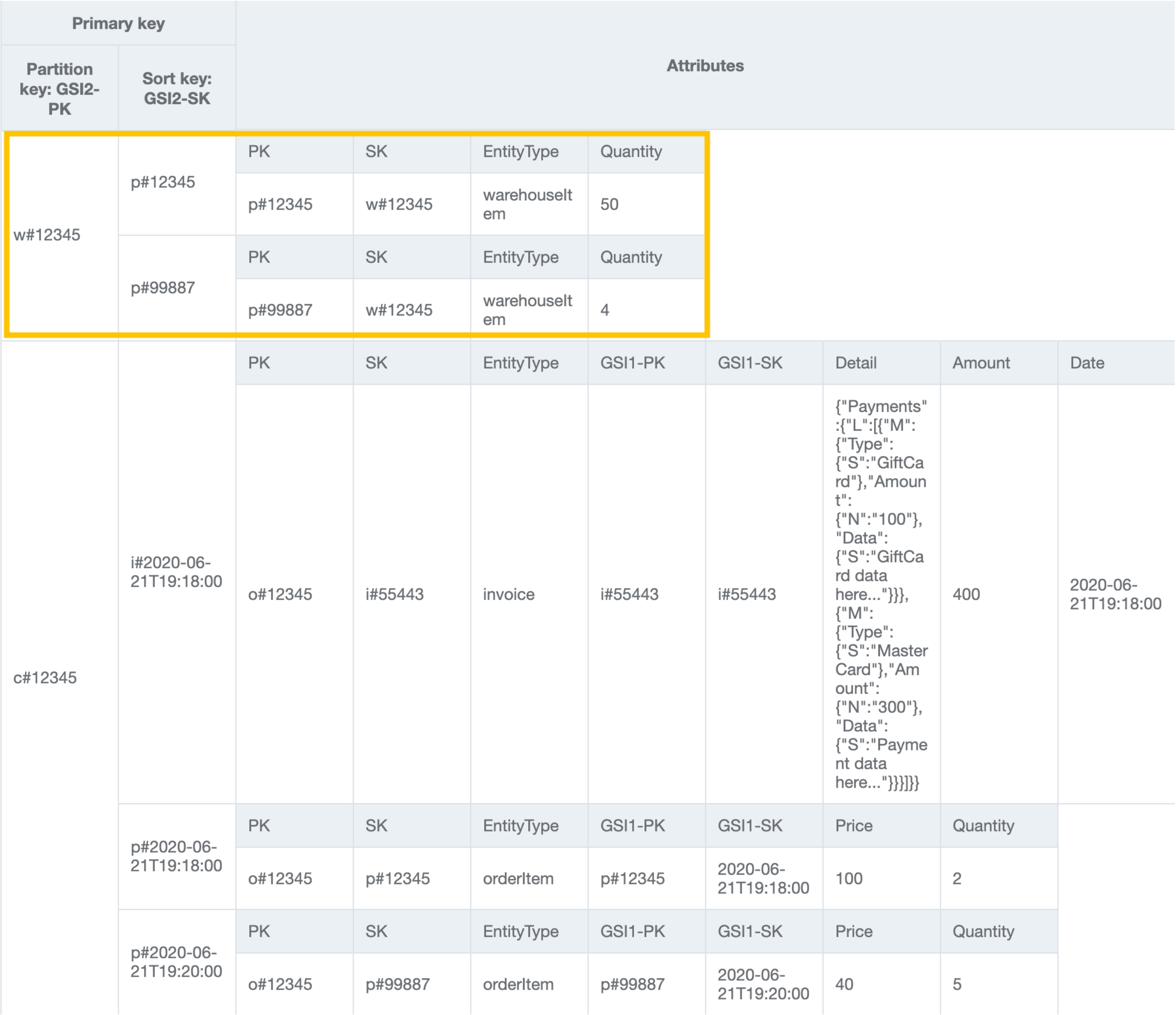


Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12376	sh#88899	o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12345	sh#98765	o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

ステップ 11: アクセスパターン 14 (**getProductInventoryByWarehouseId**)、15 (**getInvoiceByCustomerIdForDateRange**)、および 16 (**getProductsByCustomerIdForDateRange**) に対処する

AnOnlineShop_13.json をインポートして、次のアクセスパターンセットに関連するデータを追加します。アクセスパターン 14 (getProductInventoryByWarehouseId) に対処するには、GSI2-PK=warehouseId および GSI2-SK begins_with “p#” を使用して GSI2 に対してクエリを実行します。

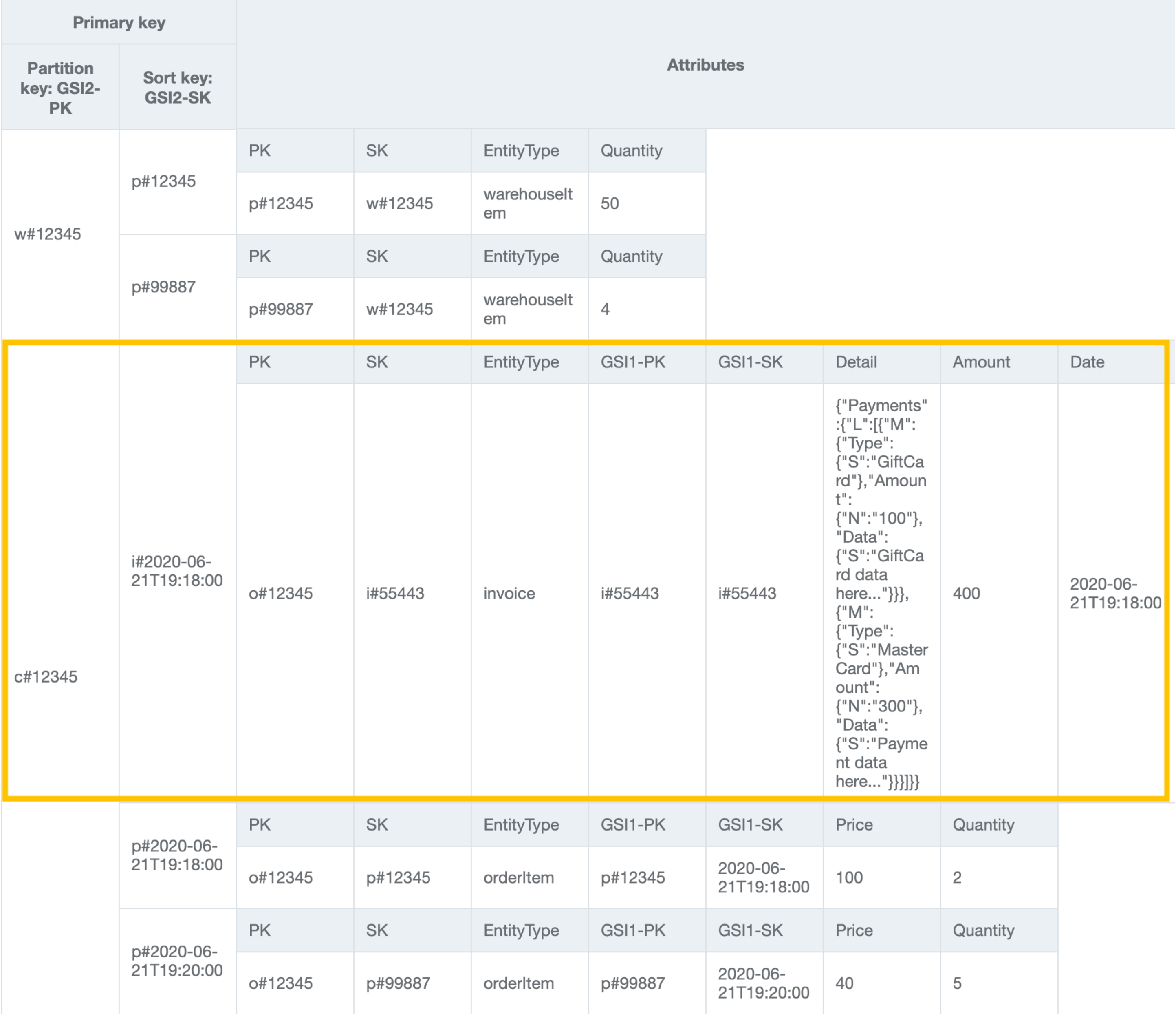
GSI2:



Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

アクセスパターン 15 (`getInvoiceByCustomerIdForDateRange`) に対処するには、`GSI2-PK=customerId` および `GSI2-SK between (i#date1, i#date2)` を使用して GSI2 に対してクエリを実行します。

GSI2:



Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

アクセスパターン 16 (`getProductsByCustomerIdForDateRange`) に対処するには、`GSI2-PK=customerId` および `GSI2-SK between (p#date1, p#date2)` を使用して `GSI2` に対してクエリを実行します。

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Note

NoSQL Workbench では、ファセットは DynamoDB に対するアプリケーションのさまざまなデータアクセスパターンを表します。ファセットを使用すると、ファセットの制約を満たさないレコードを表示することなく、テーブル内のデータのサブセットを表示できます。プロビジョンドキャパシティの引き上げがファセットはビジュアルデータモデリングツールと見なされ、アクセスパターンのモデリングを純粋に補助するものであるため、DynamoDB で使用できるコンストラクトとしては存在しません。

AnOnlineShop_facets.json をインポートして、このユースケースのファセットを確認します。

すべてのアクセスパターンと各アクセスパターンにスキーマ設計で対処する方法を次の表にまとめています。

アクセスパターン	ベーステーブル/GSI/LSI	オペレーション	パーティションキー値	ソートキー値
getCustomerByCustomerId	ベーステーブル	GetItem	PK=customerId	SK=customerId
getProductByProductId	ベーステーブル	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	ベーステーブル	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	ベーステーブル	Query	PK=productId	SK begins_with "w#"
getOrderDetailsByOrderId	ベーステーブル	Query	PK=orderId	
getProductByOrderId	ベーステーブル	Query	PK=orderId	SK begins_with "p#"
getInvoiceByOrderId	ベーステーブル	Query	PK=orderId	SK begins_with "i#"
getShipmentByOrderId	ベーステーブル	Query	PK=orderId	SK begins_with "sh#"
getOrderByProductIdForDateRange	GSI1	Query	PK=productId	SK between date1 and date2

アクセスパターン	ベーステーブル/GSI/LSI	オペレーション	パーティションキー値	ソートキー値
getInvoiceByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId
getPaymentByInvoiceId	GSI1	Query	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Query	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "sh#"
getProductInventoryByWarehouseId	GSI2	Query	PK=warehouseId	SK begins_with "p#"
getInvoiceByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK between i#date1 and i#date2
getProductsByCustomerIdForDateRange	GSI2	Query	PK=customerId	SK between p#date1 and p#date2

オンラインショップの最終スキーマ

最終的なスキーマ設計は次のとおりです。このスキーマ設計を JSON ファイルとしてダウンロードするには、GitHub の「DynamoDB 設計パターン」を参照してください。

ベーステーブル

Primary key		Attributes			
Partition key: PK	Sort key: SK				
c#12345	c#12345	EntityType	Email	Name	
		customer	samaneh@example.com	Samaneh	
c#23456	c#23456	EntityType	Email	Name	
		customer	kathleen@example.com	Kathleen	
c#54321	c#54321	EntityType	Email	Name	
		customer	henrik@example.com	Henrik	
p#12345	p#12345	EntityType	Detail	Price	
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#12345	50
p#99887	p#99887	EntityType	Detail	Price	
		product	{"Name":{"S":"The Book"},"Description":{"S":"The best book ever"}}	40	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#99887	4
	w#12376	EntityType	Quantity		
		warehouseItem	4		
w#12345	w#12345	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}		
		EntityType	Address		
			{"Country":{"S":"Sweden"},"County"		

GSI1

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK								
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#12345	orderItem	c#12345	2020-06-21T19:18:00	100	2	
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#99887	orderItem	c#12345	2020-06-21T19:20:00	40	5	
i#55443	i#55443	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	c#12345	2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
sh#88899	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
sh#98765	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"	Express	2020-06-22T10:20:00

GSI2

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
	sh#98765	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00
c#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	
w#12376	sh#88899	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00

このスキーマ設計での NoSQL Workbench の使用

この最終スキーマを、DynamoDB のデータモデリング、データ視覚化、クエリ開発機能を提供するビジュアルツールである NoSQL Workbench にインポートして、新しいプロジェクトを詳しく調べたり編集したりできます。使用を開始するには、次の手順に従います。

1. NoSQL Workbench をダウンロードします。詳細については、「the section called “ダウンロード”」を参照してください。
2. 上記の JSON スキーマファイルをダウンロードします。このファイルは既に NoSQL Workbench モデル形式になっています。
3. JSON スキーマファイルを NoSQL Workbench にインポートします。詳細については、「the section called “既存のモデルのインポート”」を参照してください。
4. NOSQL Workbench にインポートしたら、データモデルを編集できます。詳細については、「the section called “既存モデルの編集”」を参照してください。
5. データモデルの視覚化、サンプルデータの追加、CSV ファイルからのサンプルデータのインポートを行うには、NoSQL Workbench のデータビジュアライザー機能を使用します。

リレーショナルデータベースから DynamoDB への移行

リレーショナルデータベースを DynamoDB に移行する際には、確実に成功させるための綿密な計画が必要です。このガイドでは、このプロセスの仕組み、利用できるツール、考えられる移行戦略を評価して要件に合うものを選択する方法を説明します。

トピック

DynamoDB に移行する理由

Amazon DynamoDB への移行は、企業や組織に幅広い魅力的なメリットをもたらします。データベースの移行先に DynamoDB を選択することで得られる主なメリットは次のとおりです。

- スケーラビリティ: DynamoDB は、大量のワークロードを処理し、データ量とトラフィックの増加に合わせてシームレスにスケールできるように設計されています。DynamoDB を使用すると、需要に応じてデータベースを簡単にスケールアップまたはスケールダウンできるため、アプリケーションのパフォーマンスを損なうことなくトラフィックの急増に対応できます。
- パフォーマンス: DynamoDB では低レイテンシーでのデータアクセスが可能であるため、並外れた速度でデータをアプリケーションに取得して処理できます。分散型アーキテクチャにより、読み取りと書き込みの操作が複数のノードに分散され、高いリクエストレートでも 1 桁ミリ秒単位の応答時間を一貫して実現できます。
- フルマネージド: DynamoDB は、AWS が提供するフルマネージドサービスです。つまり、プロビジョニング、設定、パッチ適用、バックアップ、スケーリングなど、データベース管理の運用面は AWS が処理するということです。これにより、企業はデータベースの管理タスクよりもアプリケーションの開発に集中できます。

- サーバーレスアーキテクチャ: DynamoDB は、DynamoDB オンデマンドと呼ばれるサーバーレスモデルをサポートしています。このモデルでは、事前にキャパシティをプロビジョニングすることなく、アプリケーションによって行われる実際の読み取りおよび書き込みリクエストに対してのみ料金が発生します。このリクエストベースの課金モデルでは、使用したリソースに対してのみ支払いを行うことになり、キャパシティのプロビジョニングや監視の必要がないため、コスト効率が向上し、運用上のオーバーヘッドも最小限に抑えられます。
- NoSQL の柔軟性: 従来のリレーショナルデータベースとは異なり、DynamoDB は NoSQL データモデルを採用しているため、スキーマ設計に柔軟性があります。DynamoDB では、構造化データ、半構造化データ、非構造化データを保存でき、変化を続ける多様なデータ型の処理に最適です。この柔軟性により、開発サイクルが短縮され、変化するビジネス要件に容易に適応できます。
- 優れた可用性と耐久性: DynamoDB では、リージョン内の複数のアベイラビリティーゾーンにデータをレプリケーションして、優れた可用性とデータ耐久性を確保します。レプリケーション、フェイルオーバー、リカバリが自動的に処理されるため、データ損失やサービス中断のリスクが最小限に抑えられます。DynamoDB では、最大 99.999% の可用性 SLA が提供されています。
- セキュリティとコンプライアンス: DynamoDB と AWS Identity and Access Management の統合によって、きめ細かいアクセスコントロールが実現します。保管中も転送中もデータを暗号化してセキュリティを確保します。また、DynamoDB は HIPAA、PCI DSS、GDPR などのさまざまなコンプライアンス標準に準拠しているため、規制要件も満たすことができます。
- AWS エコシステムとの統合: DynamoDB は AWS エコシステムの一部として、AWS Lambda、AWS CloudFormation、AWS AppSync などの他の AWS サービスとシームレスに統合します。この統合により、サーバーレスアーキテクチャの構築、Infrastructure as Code の活用、リアルタイムのデータ主導型アプリケーションの作成が可能になります。

リレーショナルデータベースを DynamoDB に移行する際の考慮事項

リレーショナルデータベースシステムと NoSQL データベースにはそれぞれ異なる長所と短所があります。これらの相違点により、2 つのシステム間でデータベース設計が異なるものになります。

	リレーショナルデータベース	NoSQL データベース
データベースのクエリ	リレーショナルデータベースでは、データは柔軟にクエリできますが、クエリは比較的コストが高く、トラ	一方、DynamoDB のような NoSQL データベースでは、データは限られた数の方法で効率的にクエリできます

	リレーショナルデータベース	NoSQL データベース
	フィックが多い状況ではスケールがうまくいかない場合があります (「DynamoDB でリレーショナルデータをモデル化するための最初のステップ」を参照)。リレーショナルデータベースアプリケーションでは、ストアドプロシージャ、SQL サブクエリ、一括更新クエリ、集計クエリにビジネスロジックを実装する場合があります。	が、その範囲外では、クエリは高コストで低速になりがちです。DynamoDB への書き込みはシングルトンです。以前にストアドプロシージャで実行されていたアプリケーションのビジネスロジックは、Amazon EC2 や AWS Lambda などのホストで実行されるカスタムコードによって DynamoDB の外部で実行されるようにリファクタリングする必要があります。
データベースの設計	実装の詳細やパフォーマンスを気にせずに柔軟に設計できます。クエリの最適化は一般的にスキーマ設計には影響しませんが、正規化は重要です。	最も一般的で重要なクエリをできるだけ速く、安価にするために、具体的にスキーマを設計します。データ構造は、ビジネスユースケースの特定の要件に合わせて調整されています。

NoSQL データベースの設計には、リレーショナルデータベース管理システム (RDBMS) の設計とは異なる考え方が必要です。RDBMS の場合は、アクセスパターンを考慮せずに正規化されたデータモデルを作成できます。その後、新しい課題とクエリの要件が発生したら、そのデータモデルを拡張することができます。各タイプのデータを独自のテーブルに整理できます。

NoSQL の設計では、答えを必要とする質問が分かるまで、DynamoDB のスキーマの設計を開始すべきではありません。ビジネス上の問題とアプリケーションの読み取り/書き込みパターンを理解することが不可欠です。さらに、DynamoDB アプリケーションで維持するテーブルをできるだけ少なくする必要もあります。テーブルの数が少なくなると、スケーラビリティが向上し、必要なアクセス権限の管理が少なくなり、DynamoDB アプリケーションのオーバーヘッドが削減されます。また、バックアップコストを全体的に低く抑えるのにも役立ちます。

DynamoDB のリレーショナルデータをモデル化し、フロントエンドアプリケーションの新しいバージョンを構築するタスクについては、別のトピックで説明します。このガイドは、DynamoDB を使用するために構築された新しいバージョンのアプリケーションがあることを前提としていますが、カットオーバー中に履歴データを移行して同期する最適な方法を決定する必要はあります。

サイズに関する考慮事項

DynamoDB テーブルに格納する各項目 (行) の最大サイズは 400 KB です。詳細については、「クォータと制限」を参照してください。項目のサイズは、項目のすべての属性名と属性値の合計サイズによって決まります。詳細については、「the section called “項目サイズと形式”」を参照してください。

アプリケーションで DynamoDB のサイズ制限よりも多くのデータを格納する必要がある場合は、項目を項目コレクションに分割するか、項目データを圧縮するか、項目をオブジェクトとして Amazon Simple Storage Service (Amazon S3) に格納して Amazon S3 オブジェクト識別子を DynamoDB 項目に格納します。「the section called “大きな項目”」を参照してください。項目の更新にかかるコストは、項目のフルサイズに基づいて決まります。既存の項目を頻繁に更新する必要があるワークロードでは、1 ~ 2 KB の小さい項目の方が、大きい項目よりも更新にかかるコストが低くなります。項目コレクションの詳細については、「the section called “項目コレクションの操作”」を参照してください。

パーティションとソートキーの属性、その他のテーブル設定、項目のサイズと構造、およびセカンダリインデックスを作成するかどうかを選択するときは、必ず DynamoDB のモデリングに関するドキュメントと「the section called “コスト最適化”」のガイドを確認してください。DynamoDB の機能と制限の範囲内でコスト効率よく DynamoDB ソリューションを運用できるように、移行計画を必ずテストしてください。

DynamoDB への移行の仕組みを理解する

利用可能な移行ツールを確認する前に、DynamoDB による書き込みの処理方法を検討してください。

> Note
>
> DynamoDB はデータを自動的にシャーディングして複数の共有サーバーとストレージに分散するため、大きなデータセットを本番環境サーバーに直接一括インポートする方法はありません。

デフォルトの最も一般的な書き込み操作は、単一の PutItem API オペレーションです。ループ内で PutItem オペレーションを実行してデータセットを処理できます。DynamoDB は実質的に無制限の同時接続をサポートしているため、MapReduce や Spark などの大規模なマルチスレッドのロードルーチンを設定して実行できると仮定すると、書き込み速度はターゲットテーブルのキャパシティ (これも通常は無制限) によってのみ制限されます。

DynamoDB にデータをロードするときは、ローダーの書き込み速度を把握することが重要です。ロードする項目 (行) のサイズが 1 KB 以下の場合、この速度は単に 1 秒あたりの項目数となります。これで、この速度を処理するのに十分な WCU (書き込みキャパシティユニット) をターゲットテーブルにプロビジョニングできます。ローダーがプロビジョニングされたキャパシティを 1 秒でも超えた場合、追加のリクエストはスロットリングされるか、完全に拒否される可能性があります。スロットリングは、DynamoDB コンソールの [モニタリング] タブにある CloudWatch チャートで確認できます。

2 つ目のオペレーションは、BatchWriteItem という関連 API を使用して実行できます。BatchWriteItem では、最大 25 件の書き込みリクエストを 1 つの API コールにまとめることができます。これらはサービスによって受信され、テーブルへの個別の PutItem リクエストとして処理されます。BatchWriteItem を選択すると、PutItem によるシングルトン呼び出し時に、AWS SDK に備わっている自動再試行のメリットを享受できません。そのため、エラー (スロットリング例外など) が発生した場合は、BatchWriteItem への応答呼び出しで失敗した書き込みのリストを探す必要があります。CloudWatch スロットリングチャートで警告が検出された場合のスロットリング警告の処理については、「the section called “スロットリング”」を参照してください。

3 つ目のタイプのデータインポートには、DynamoDB の S3 からのインポート機能を使います。この機能により、Amazon S3 で大規模なデータセットをステージングし、そのデータを新しいテーブルに自動的にインポートするよう DynamoDB に指示できます。インポートは即時では行われず、データセットのサイズに比例した時間がかかりますが、ETL プラットフォームやカスタム DynamoDB コードの記述が必要ないため便利です。このインポート機能には制限があり、ダウンタイムが許容できる場合の移行に適しています。S3 からのデータは、インポートによって作成された新しいテーブルにロードされます。既存のテーブルにデータをロードすることはできません。データの変換は行われないため、データを最終的な形式で準備して S3 バケットに保存する上流プロセスが必要になります。

DynamoDB への移行に役立つツール

データを DynamoDB に移行するために使用できる一般的な移行ツールと ETL ツールがいくつかあります。

多くのお客様は、移行プロセス用のカスタムデータ変換を構築するために、独自の移行スクリプトとジョブを作成することを選択しています。大量の書き込みトラフィックや定期的かつ大規模な一括ロードジョブを伴う大容量の DynamoDB テーブルを運用する予定がある場合は、書き込みトラフィックが多くて DynamoDB の信頼できる動作を確保できるよう、移行ツールを独自に構築することができます。スロットリング処理や効率的なテーブルプロビジョニングなどのシナリオは、プロジェクトの早い段階で実際に移行を行う際に経験できます。

Amazon は、AWS Database Migration Service (DMS)、AWS Glue、Amazon EMR、Amazon Managed Streaming for Apache Kafka など、活用できるデータツールを多数提供しています。これらのツールはすべてダウンタイムありの移行に使用できますが、リレーショナルデータベースの変更データキャプチャ (CDC) 機能を活用できる特定のツールは、オンライン移行にも対応できます。最適なツールを選別するには、それぞれの機能、パフォーマンス、コストとともに、各ツールに関する組織のスキルセットと経験を考慮するとよいでしょう。

DynamoDB に移行するための適切な戦略の選択

大規模なリレーショナルデータベースアプリケーションは、100 以上のテーブルを使用し、複数の異なるアプリケーション機能をサポートしている場合があります。大規模な移行を行う場合は、アプリケーションを小さなコンポーネントまたはマイクロサービスに分割して、テーブルの小さなセットを一度に移行することを検討してください。その後、追加のコンポーネントを段階的に DynamoDB に移行できます。

移行戦略を選択する際、特定のパラメーターがあればソリューションを決定しやすくなります。以下のオプションをデシジョンツリーで表すことで、組織の要件とリソースに応じて利用可能なオプションを簡単に示すことができます。ここでは、概念について簡単に説明します (詳細はこのガイドの後半で説明します)。

- オフライン移行: 移行時にある程度のダウンタイムがアプリケーションで許容される場合は、移行プロセスが大幅に簡素化されます。
- ハイブリッド移行: この方法では、読み取りは許可するが書き込みは許可しない、読み取りと挿入は許可するが更新と削除は許可しないなど、移行時に部分的な稼働時間を確保できます。
- オンライン移行: 移行時にダウンタイムを一切許容しないアプリケーションの移行は容易ではなく、綿密な計画とカスタムプロセスの開発が必要になる場合があります。この場合に重要なのは、カスタム移行プロセスを構築するコストと、カットオーバー時のダウンタイムによってビジネスに生じる損失を見積もり、比較検討することです。

If	And		THEN
データ移行を実行するために、メンテナンスの時間帯にアプリケーションをしばらく停止してもかまわない場合。これはオフライン移行です。			AWS DMS でフルロードタスクを使用してオフライン移行を実行してください。必要に応じて、SQL の `VIEW` でソースデータを事前に形成します。
移行時に、アプリケーションを読み取り専用モードで実行できればよいという場合。これはハイブリッド移行です。			アプリケーションまたはソースデータベース内での書き込みを無効にします。AWS DMS でフルロードタスクを使用してオフライン移行を実行してください。
移行時に、アプリケーションで更新や削除ができなくても、読み取りや新規レコードの挿入ができ	アプリケーション開発スキルがあり、すべての新しいレコードについて DynamoDB への書き込みを含む二重書き込みを実行するために、既存のリレーショナルアプリを更新できる場合。		AWS DMS でフルロードタスクを使用してオフライン移行を実行してください。同時に、既存のアプリを読み取りと二重書き込みが可能なバージョンにしてデプロイします。

If	And		THEN
ればよいという場合。これはハイブリッド移行です。			
ダウンタイムを最小限に抑えて移行する必要がある場合。これはオンライン移行です。	スキーマを大幅に変更せずに、ソーステーブルを 1 対 1 で DynamoDB に移行する場合。		AWS DMS でオンラインデータ移行を実行してください。一括ロードタスクを実行し、続いて CDC 同期タスクを実行します。
	単一テーブルの理念に従って、ソーステーブルをより少ない数の DynamoDB テーブルに結合する場合。	バックエンドデータベース開発のスキルがあり、SQL ホストに予備のキャパシティがある場合。	SQL データベース内に NoSQL 対応テーブルを作成してください。JOIN、UNION、VIEW、トリガー、ストアドプロシージャを使用してデータの入力と同期を行います。
		バックエンドデータベース開発のスキルがなく、SQL ホストに予備のキャパシティもない場合。	ハイブリッドまたはオフラインの移行アプローチを検討してください。

If	And	THEN
	過去のトランザクションデータの移行は省略しても問題ない、または移行の代わりに Amazon S3 にアーカイブしてもよい場合。いくつかの小さな静的テーブルを移行するだけでよい場合。	スクリプトを記述するか、任意の ETL ツールを使用してテーブルを移行してください。必要に応じて、SQL の `VIEW` でソースデータを事前に形成します。

DynamoDB へのオフライン移行を実行する

オフライン移行は、移行時にダウンタイムを許容できる場合に適しています。リレーショナルデータベースは通常、メンテナンスとパッチ適用のために毎月少なくともある程度のダウンタイムを必要とし、ハードウェアのアップグレードやメジャーリリースのアップグレードではダウンタイムがさらに長くなることもあります。

Amazon S3 は移行時のステージングエリアとして使用できます。CSV (カンマ区切り値) または DynamoDB JSON 形式で保存されたデータは、DynamoDB の S3 からのインポート機能を使用して新しい DynamoDB テーブルに自動的にインポートできます。

計画

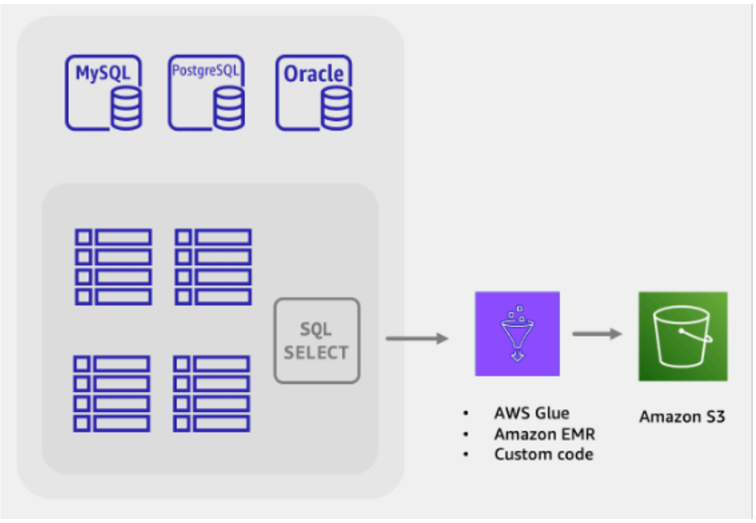
Amazon S3 を使用してオフライン移行を実行する

ツール

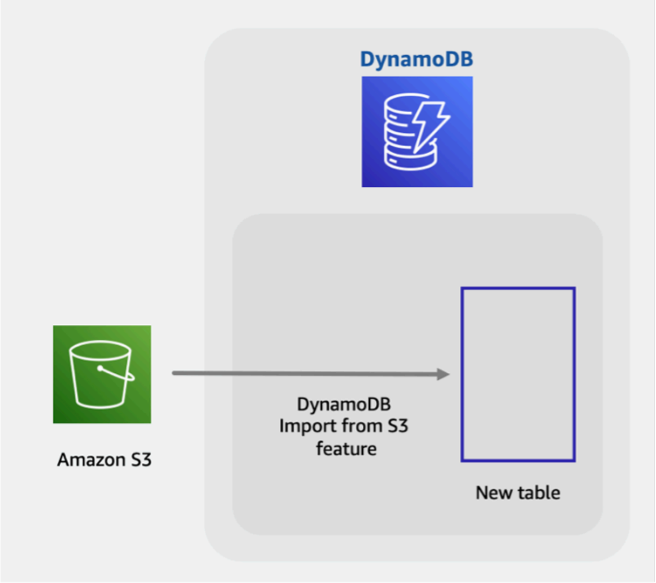
- SQL データを抽出して変換し、そのデータを S3 バケットに保存する次のような ETL ジョブ
  - AWS Glue
  - Amazon EMR
  - 独自のカスタムコード
- DynamoDB の S3 からのインポート機能

オフライン移行ステップ

1. SQL データベースにクエリを実行し、テーブルデータを DynamoDB JSON または CSV 形式に変換して、S3 バケットに保存する ETL ジョブを構築します。



2. DynamoDB の S3 からのインポート機能が呼び出され、新しいテーブルを作成して、S3 バケットからデータを自動的にロードします。



完全なオフライン移行はシンプルで簡単ですが、アプリケーションの所有者やユーザーにはおそらくあまり好まれません。移行時にアプリケーションのサービスがまったく提供されないのではなく、提供するサービスのレベルが下がるだけであれば、ユーザーも助かるでしょう。

オフライン移行時に書き込みだけを無効にして、読み取りは通常どおり継続できるようにする機能を追加できます。こうすることで、アプリケーションのユーザーは、リレーショナルデータの移行中でも既存のデータを安全に参照してクエリを実行できます。この方法が適している場合は、引き続きハイブリッド移行についてお読みください。

DynamoDB へのハイブリッド移行を実行する

読み取りおよび書き込み操作はどのデータベースアプリケーションでも実行されますが、ハイブリッド移行またはオンライン移行を計画する際には、実行される書き込み操作の種類を考慮する必要があります。データベースへの書き込みは、挿入、更新、削除の 3 つのバケットに分類できます。アプリケーションによっては、既存のレコードを更新しないものもあります。また、削除をすぐに処理する必要がなく、たとえば月末の一括クリーンアッププロセスまで処理を延期できるアプリケーションもあります。こうした種類のアプリケーションは、部分的な稼働時間を確保しながら、より簡単に移行できます。

計画

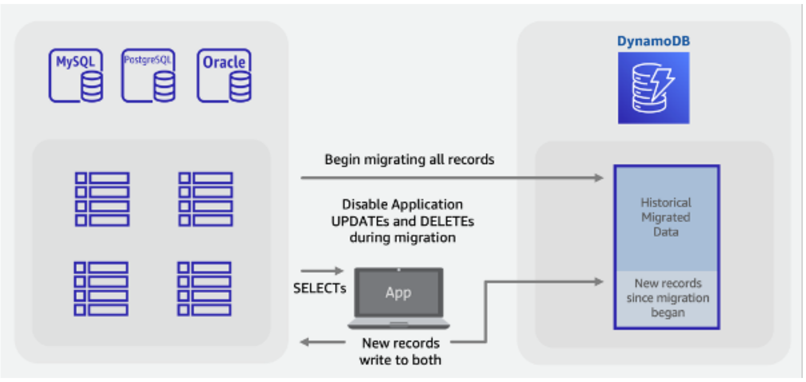
アプリケーションの二重書き込みによるオンライン/オフラインのハイブリッド移行を実行する

ツール

- SQL データを抽出して変換し、そのデータを S3 バケットに保存する次のような ETL ジョブ
  - AWS Glue
  - Amazon EMR
  - 独自のカスタムコード

ハイブリッド移行ステップ

1. ターゲットの DynamoDB テーブルを作成します。このテーブルには、過去のバルクデータと新しいライブデータの両方が格納されます。
2. レガシーアプリケーションで削除と更新を無効にし、すべての挿入が SQL データベースと DynamoDB の両方への二重書き込みとして実行されるようにして、新しいバージョンのアプリケーションを作成します。
3. ETL ジョブを開始して、既存のデータを移行すると同時に、新しいバージョンのアプリケーションをデプロイします。
4. ETL ジョブが完了すると、DynamoDB には既存のレコードと新しいレコードがすべて揃い、アプリケーションのカットオーバーの準備が整います。


MySQL
PostgreSQL
Oracle
DynamoDB
Begin migrating all records
Disable Application UPDATEs and DELETEs during migration
SELECTs
App
New records write to both
Historical Migrated Data
New records since migration began

Note

ETL ジョブでは SQL から DynamoDB に直接書き込みが行われます。インポート全体が完了するまでターゲットテーブルはパブリックにならず、他の書き込みにも使用できないため、オフライン移行の例のように S3 インポート機能を使用することはできません。

各テーブルを 1 対 1 で移行して DynamoDB へのオンライン移行を実行する

多くのリレーショナルデータベースには、変更データキャプチャ (CDC) と呼ばれる機能があり、ユーザーは特定の時点より前または後にテーブルに加えられた変更のリストをリクエストできます。CDC は内部ログを使用してこの機能を有効にするため、テーブルにタイムスタンプ列がなくても動作します。

SQL テーブルのスキーマを NoSQL データベースに移行する際、データを結合して再形成し、テーブルの数を減らすことができます。そうすることで、データを 1 か所に集めることができ、複数ステップの読み取り操作において関連データを手動で結合する必要がなくなります。ただし、単一テーブルのデータ形成は必ずしも必要ではなく、テーブルを 1 対 1 で DynamoDB に移行することもあります。こうした 1 対 1 のテーブル移行は、この種の移行をサポートする一般的な ETL ツールを使ってソースデータベースの CDC 機能を活用すれば、それほど複雑ではありません。各行のデータは新しい形式に変換することも可能ですが、各テーブルの範囲は変わりません。

SQL テーブルを 1 対 1 で DynamoDB に移行することを検討する場合は、サーバー側の結合はないという点に注意してください。

計画

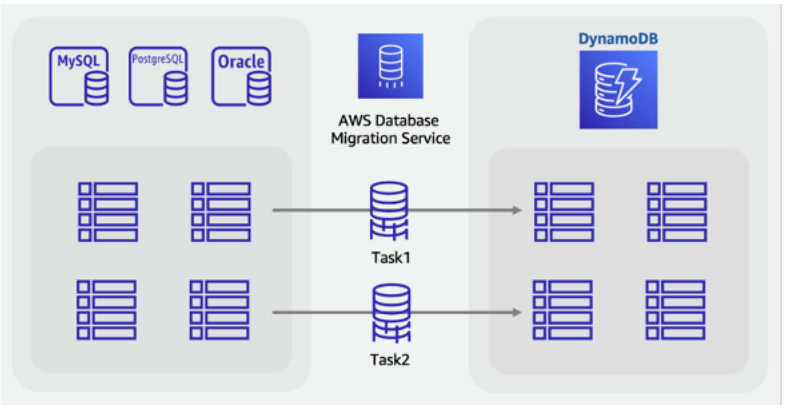
AWS DMS を使用して各テーブルの DynamoDB へのオンライン移行を実行する

ツール

- AWS Database Migration Service (DMS): 履歴データの一括ロードに加え、CDC を利用してソーステーブルとターゲットテーブルの同期も可能な ETL ツール

オンライン移行ステップ

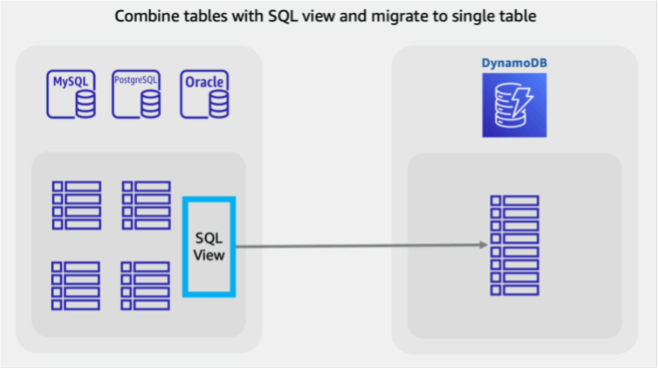
1. ソーススキーマ内の移行するテーブルを特定します。
2. DynamoDB に同様のキー構造で同じ数のテーブルを作成します。
3. AWS DMS にレプリケーションサーバーを作成し、ソースとターゲットのエンドポイントを設定します。
4. 行ごとの必要な変換 (連結列、日付の ISO-8601 文字列形式への変換など) を定義します。
5. [フルロードと変更データキャプチャ (CDC)] で、テーブルごとに移行タスクを作成します。
6. レプリケーションフェーズの処理が始まるまで、これらのタスクを監視します。
7. この時点で、検証監査をすべて実行すれば、DynamoDB へ読み取りと書き込みを行うアプリケーションにユーザーを切り替えることができます。



カスタムステージングテーブルを使用して DynamoDB へのオンライン移行を実行する

独自の NoSQL アクセスパターンを活用するためにテーブルを組み合わせることが望ましい場合があります (たとえば、4 つのレガシーテーブルを 1 つの DynamoDB テーブルに変換するなど)。通常、単一のキーと値のドキュメントのリクエスト、または事前にグループ化された項目コレクションのクエリは、複数テーブルの結合を実行する SQL データベースよりもレイテンシーが短くなります。ただし、これによって移行タスクは困難になります。SQL の `VIEW` を使ってソースデータベース内

で処理を行い、4 つのテーブルすべてを 1 つにまとめた単一のデータセットを作成することもできます。



このビューでは、テーブルを非正規化された 1 つの形式に JOIN することも、SQL の UNION を使用して正規化されたエンティティのままでテーブルをスタックすることもできます。リレーショナルデータの再形成に関する重要な決定事項については、こちらのビデオで説明しています。オフライン移行では、ビューを使用してテーブルを結合することが、DynamoDB の単一テーブルスキーマのデータを形成するのに最適な方法です。

ただし、変化するライブデータを含むオンライン移行では、CDC 機能は活用できません。CDC 機能は単一テーブルクエリでのみサポートされ、VIEW 内ではサポートされないためです。テーブルに最終更新日時のタイムスタンプ列があり、その列が VIEW に組み込まれている場合は、その列を使用して同期による一括ロードを実行するカスタム ETL ジョブを構築できます。

この課題に対する新しいアプローチとして、ビュー、ストアドプロシージャ、トリガーといった標準の SQL 機能を使用して、最終的に必要な DynamoDB NoSQL 形式で新しい SQL テーブルを作成することもできます。

データベースサーバーに追加のストレージ容量を割り当てることができる場合は、移行を開始する前に、この単一のステージングテーブルを作成することが可能です。そのためには、既存のテーブルからの読み取り、必要なデータ変換、新しいステージングテーブルへの書き込みを行うストアドプロシージャを作成します。一連のトリガーを追加して、テーブルの変更をステージングテーブルでリアルタイムに複製することもできます。会社のポリシーでトリガーが許可されていない場合は、ストアドプロシージャを変更しても同様の結果が得られる可能性があります。データを書き込むプロシージャに数行のコードを加え、同じ変更をステージングテーブルに追加で書き込みます。

このステージングテーブルをレガシーアプリケーションテーブルと完全に同期された場所に置いておくと、ライブ移行の出発点として最適です。このテーブルでは、AWS DMS など、データベース CDC を使ってライブ移行を実行するツールが使用できるようになっています。この方法のメリットは、リレーショナルデータベースエンジンで利用可能なよく知られた SQL スキルと機能を使用できることです。

計画

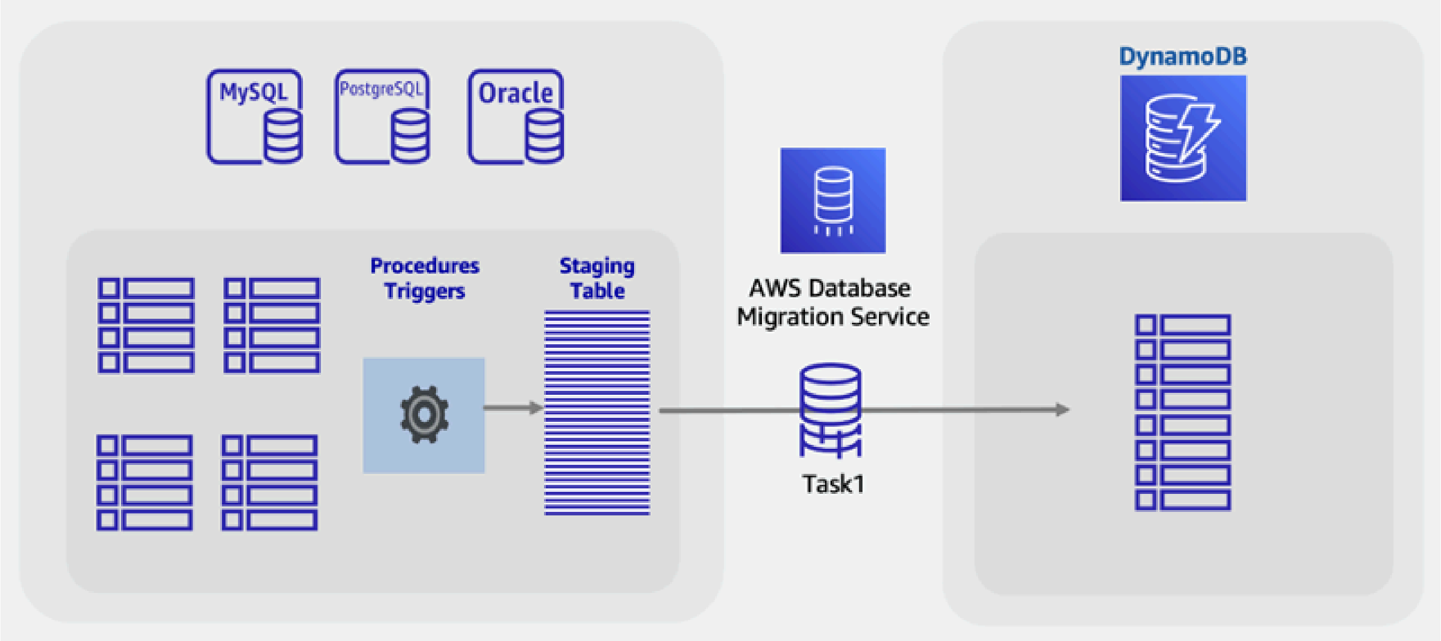
AWS DMS と SQL ステージングテーブルを使用してオンライン移行を実行する

ツール

- カスタム SQL ストアドプロシージャまたはトリガー
- AWS Database Migration Service (DMS): ライブステージングテーブルを DynamoDB に移行できる ETL ツール

オンライン移行ステップ

1. ソースのリレーショナルデータベースエンジン内に、予備のディスク容量と処理容量がある程度あることを確認します。
2. タイムスタンプまたは CDC 機能を有効にして、SQL データベースに新しいステージングテーブルを作成します。
3. ストアドプロシージャを記述して実行し、既存のリレーショナルテーブルデータをステージングテーブルにコピーします。
4. トリガーをデプロイするか、既存のプロシージャを変更して、既存のテーブルに通常の書き込みを実行しながら新しいステージングテーブルに二重書き込みが行われるようにします。
5. AWS DMS を実行して、このソーステーブルをターゲットの DynamoDB テーブルに移行して同期します。



このガイドでは、ダウンタイムを最小限に抑えることと、一般的なデータベースツールや手法を使用することに重点を置いて、リレーショナルデータベースのデータを DynamoDB に移行する際の考慮事項とアプローチをいくつか紹介しました。詳細については、次を参照してください:

- AWS DMS ユーザーガイド
- AWS Glue ユーザーガイド
- RDBMS から DynamoDB への移行のベストプラクティス

DynamoDB 用の NoSQL Workbench

Amazon DynamoDB 用の NoSQL Workbench は、最新のデータベース開発および運用向けのクロスプラットフォームのクライアント側 GUI アプリケーションです。Windows、macOS、Linux で使用できます。NoSQL Workbench は、DynamoDB テーブルの設計、作成、クエリ、管理に役立つデータモデリング、データ可視化、クエリ開発といった特徴を提供する視覚的開発ツールです。NoSQL Workbench に、インストールプロセスのオプションとして DynamoDB local が含まれるようになったため、DynamoDB local でデータを簡単にモデル化できます。DynamoDB local とその要件の詳細については、「 DynamoDB local (ダウンロード可能バージョン) のセットアップ 」を参照してください。

データモデリング

DynamoDB 用の NoSQL Workbench を使用すると、アプリケーションのデータアクセスパターンを満たす既存のデータモデルから新しいデータモデルを構築したり、既存のデータモデルに基づいてモデルを設計したりできます。プロセスの最後に、設計されたデータモデルをインポートおよびエクスポートすることもできます。詳細については、「NoSQL Workbench を使用したデータモデルの構築」を参照してください。

データの可視化

データモデルビジュアライザーは、コードを記述せずにクエリをマップし、アプリケーションのアクセスパターン (ファセット) を可視化できるキャンバスを提供します。すべてのファセットは、DynamoDB の異なるアクセスパターンに対応しています。データ型で使用するサンプルデータを自動生成できます。詳細については、「データアクセスパターンの可視化」を参照してください。

オペレーション構築

NoSQL Workbench は、クエリを開発およびテストするための豊富なグラフィカルユーザーインターフェイスを提供します。オペレーションビルダーを使用して、データセットを表示、探索、およびクエリできます。構造化オペレーションビルダーは、射影式、条件式をサポートし、複数の言語でサンプルコードを生成します。ある Amazon DynamoDB アカウントのテーブルのクローンを別のリージョンの別のアカウントで直接作成できます。また、DynamoDB Local と Amazon DynamoDB アカウントの間でテーブルを直接クローンして、開発環境間におけるテーブルのキースキーマ (およびオプションで GSI スキーマと項目) のコピーを高速化することもできます。詳細については、「NoSQL Workbench を使用したデータセットの探索とオペレーションの構築」を参照してください。

次の動画では、NoSQL Workbench を使用したデータモデリングの概念について詳しく説明します。

トピック

DynamoDB 用の NoSQL Workbench のダウンロード

Amazon DynamoDB 用の NoSQL Workbench および DynamoDB local* をダウンロードするには、以下の手順に従います。

前提条件

Ubuntu のインストールには、libfuse2 と curl という 2 つの前提条件となるソフトウェアが必要です。

libfuse2

> Ubuntu 22.04 では、libfuse2 はデフォルトでインストールされなくなりました。これを解決するには、`sudo add-apt-repository universe && sudo apt install libfuse2` を実行して最新の Ubuntu バージョンを実行してインストールしてください。

curl

> Ubuntu をアップデートし、`sudo apt update && sudo apt upgrade` を実行します。
>
> 次に、cURL をインストールし、`sudo apt install curl` を実行します。

NoSQL ワークベンチおよび DynamoDB local をダウンロードするには

1. オペレーティングシステムに適したバージョンの NoSQL Workbench をダウンロードします。

オペレーティングシステム	ダウンロードリンク
macOS (Intel)**	macOS (Intel) 用のダウンロード
macOS (Apple シリコン)	macOS (Apple シリコン) 用のダウンロード
Windows	Windows 用のダウンロード
Linux**	Linux 用のダウンロード

* NoSQL Workbench には、インストールプロセスのオプションとして DynamoDB local が含まれています。

** NoSQL Workbench を開こうとしたときに、アプリが身元確認された開発者によって Apple に登録されていないという警告メッセージが表示される場合は、次の操作を行います。

1. アプリを探して開きます。
2. Control キーを押しながらアプリケーションアイコンをクリックし、ショートカットメニューから [開く] を選択します。

   これにより、アプリケーションがセキュリティ設定の例外として保存されます。登録済みのアプリを開くのと同じように、アプリケーションをダブルクリックして開きます。

*** NoSQL Workbench は Ubuntu 12.04、Fedora 21、Debian 8、またはこれらの Linux ディストリビューションの新しいバージョンをサポートしています。

2. ダウンロードしたアプリケーションを起動し、「NoSQL Workbench のインストール」のステップに従います。

> Note
>
> DynamoDB Local を実行するには、Java ランタイム環境 (JRE) バージョン 8.x 以降が必要です。

DynamoDB 用の NoSQL Workbench のインストール

サポートされているプラットフォームに NoSQL Workbench と DynamoDB local をインストールするには、以下の手順に従ってください。

Windows

Windows で NoSQL Workbench をインストールするには

1. NoSQL Workbench インストーラーアプリケーションを実行し、セットアップ言語を選択します。次に、[OK] を選択してセットアップを開始します。NoSQL Workbench のダウンロードの詳細については、「DynamoDB 用の NoSQL Workbench のダウンロード」を参照してください。
2. [Next] (次へ) を選択してセットアップを続行し、次の画面で [Next] (次へ) を選択します。
3. デフォルトでは、[DynamoDB local のインストール] チェックボックスがオンになっており、インストールの一部として DynamoDB local が含まれます。このオプションを選択したままにしておくと、DynamoDB local がインストールされ、インストール先のパスが NoSQL Workbench のインストールパスと同じになります。このオプションのチェックボックスをオフにすると、DynamoDB local のインストールがスキップされ、インストールパスは NoSQL Workbench のみになります。

   ソフトウェアのインストール先を選択し、[Next] (次へ) を選択します。

   > Note
   >
   > セットアップの一部として DynamoDB Local を含めない場合は、[DynamoDB Local をインストールする] チェックボックスをオフにし、[次へ] をクリックしてステップ 6 に進みます。DynamoDB local は、後でスタンドアロンインストールとして個別にダウンロードできます。詳細については、「 DynamoDB local (ダウンロード可能バージョン) のセットアップ 」を参照してください。

   このステップでインストールパスを設定します。
4. DynamoDB local が使用するポート番号を選択します。デフォルトのポート番号は 8000 です。ポート番号を入力したら、[Next] (次へ) を選択します。
5. [Next] (次へ) を選択してセットアップを開始します。
6. セットアップが完了したら、[Finish] (完了) を選択してセットアップ画面を閉じます。

7. /programs/DynamoDBWorkbench/ などのインストールパスにあるアプリケーションを開きます。

macOS

macOS で NoSQL Workbench をインストールするには

1. NoSQL Workbench インストーラーアプリケーションを実行し、セットアップ言語を選択します。次に、[OK] を選択してセットアップを開始します。NoSQL Workbench のダウンロードの詳細については、「DynamoDB 用の NoSQL Workbench のダウンロード」を参照してください。
2. [Next] (次へ) を選択してセットアップを続行し、次の画面で [Next] (次へ) を選択します。
3. デフォルトでは、[DynamoDB local のインストール] チェックボックスがオンになっており、インストールの一部として DynamoDB local が含まれます。このオプションを選択したままにしておくと、DynamoDB local がインストールされ、インストール先のパスが NoSQL Workbench のインストールパスと同じになります。このオプションをオフにすると、DynamoDB local のインストールがスキップされ、インストールパスは NoSQL Workbench のみになります。

   ソフトウェアのインストール先を選択し、[Next] (次へ) を選択します。

   > **Note**
   >
   > セットアップの一部として DynamoDB Local を含めない場合は、[DynamoDB Local をインストールする] チェックボックスをオフにし、[次へ] をクリックしてステップ 6 に進みます。DynamoDB local は、後でスタンドアロンインストールとして個別にダウンロードできます。詳細については、「 DynamoDB local (ダウンロード可能バージョン) のセットアップ」を参照してください。

   このステップでインストールパスを設定します。
4. DynamoDB local が使用するポート番号を選択します。デフォルトのポート番号は 8000 です。ポート番号を入力したら、[Next] (次へ) を選択します。
5. [Next] (次へ) を選択してセットアップを開始します。
6. セットアップが完了したら、[Finish] (完了) を選択してセットアップ画面を閉じます。

7. /Applications/DynamoDBWorkbench/ などのインストールパスにあるアプリケーションを開きます。

> **Note**
>
> macOS 用 NoSQL ワークベンチは自動更新を実行します。更新に関する通知を受け取るには、[System Preferences] (システム環境設定) > [Notifications] (通知) で NoSQL Workbench への通知アクセスを有効にします。

Linux

Linux で NoSQL Workbench をインストールするには

1. NoSQL Workbench インストーラーアプリケーションを実行し、セットアップ言語を選択します。次に、[OK] を選択してセットアップを開始します。NoSQL Workbench のダウンロードの詳細については、「DynamoDB 用の NoSQL Workbench のダウンロード」を参照してください。
2. [Forward] (進む) を選択してセットアップを続行し、次の画面で [Forward] (進む) を選択します。
3. デフォルトでは、[DynamoDB local のインストール] チェックボックスがオンになっており、インストールの一部として DynamoDB local が含まれます。このオプションを選択したままにしておくと、DynamoDB local がインストールされ、インストール先のパスが NoSQL Workbench のインストールパスと同じになります。このオプションをオフにすると、DynamoDB local のインストールがスキップされ、インストールパスは NoSQL Workbench のみになります。

   ソフトウェアのインストール先を選択し、[Forward] (進む) を選択します。

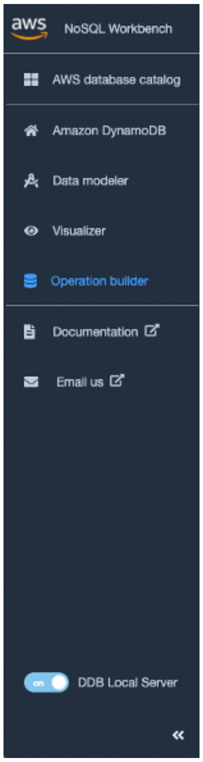
   > **Note**
   >
   > セットアップの一部として DynamoDB Local を含めない場合は、[DynamoDB Local をインストールする] チェックボックスをオフにし、[進む] をクリックしてステップ 6 に進みます。DynamoDB local は、後でスタンドアロンインストールとして個別にダウンロードできます。詳細については、「DynamoDB local (ダウンロード可能バージョン) のセットアップ」を参照してください。

このステップでインストールパスを設定します。

4. DynamoDB local が使用するポート番号を選択します。デフォルトのポート番号は 8000 です。ポート番号を入力したら、[Forward] (進む) を選択します。
5. [Forward] (進む) を選択してセットアップを開始します。
6. セットアップが完了したら、[Finish] (完了) を選択してセットアップ画面を閉じます。
7. /usr/local/programs/DynamoDBWorkbench/ などのインストールパスにあるアプリケーションを開きます。

> ⓘ Note
>
> NoSQL Workbench のインストールの一部として DynamoDB local をインストールすることを選択した場合、DynamoDB local はデフォルトオプションで事前設定されます。デフォルトオプションを編集するには、/resources/DDBLocal_Scripts/ ディレクトリにある DDBLocalStart スクリプトを変更します。これはインストール時に指定したパスにあります。DynamoDB local オプションの詳細については、「DynamoDB local の使用に関する注意事項」を参照してください。

NoSQL Workbench のインストールの一部として DynamoDB local をインストールすることを選択した場合、次の図に示すように、DynamoDB local を有効または無効に切り替えることができます。



NoSQL Workbench を使用したデータモデルの構築

Amazon DynamoDB 用 NoSQL Workbench のデータモデラーツールを使用して、新しいデータモデルを構築したり、アプリケーションのデータアクセスパターンを満たす既存のデータモデルに基づいてモデルを設計したりできます。データモデラーには、使用開始に役立つサンプルデータモデルがいくつか含まれています。

トピック

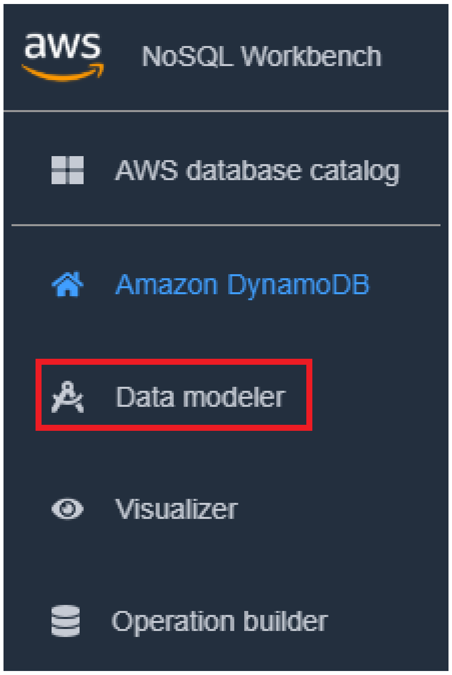
- 新しいデータモデルの作成
- 既存のデータモデルのインポート
- データモデルのエクスポート
- 既存のデータモデルの編集

新しいデータモデルの作成

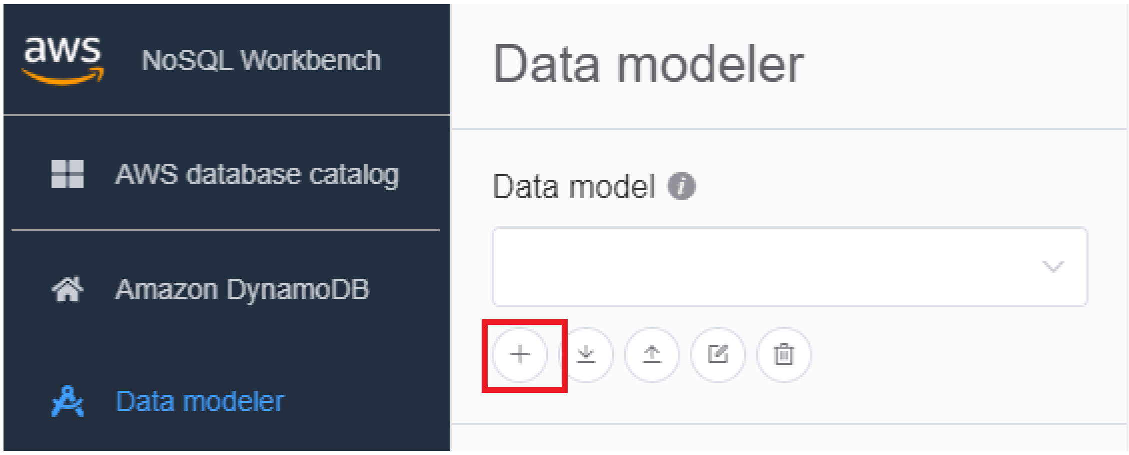
以下のステップに従って、NoSQL Workbench を使用して Amazon DynamoDB に新しいデータモデルを作成します。

新しいデータモデルを作成するには

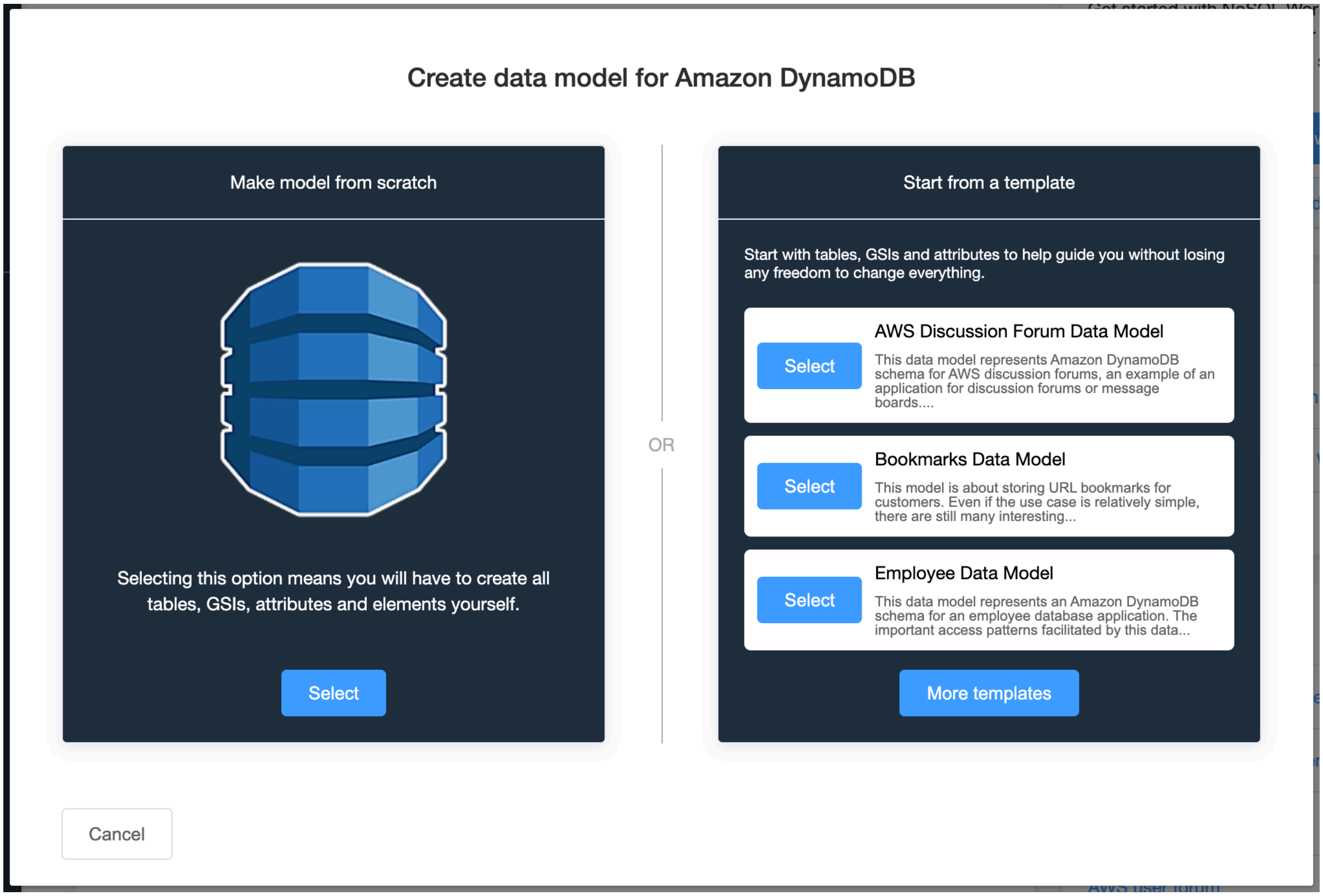
1. NoSQL Workbench を開き、左側のナビゲーションペインで [Data modeler (データモデラー)] アイコンを選択します。



2. [Create data model (データモデルの作成)] を選択します。



[Create data model] (データモデルの作成) には、[Make model from scratch] (モデルをゼロから作成) と [Start from a template] (テンプレートから開始) の 2 つの選択肢があります。



Create data model for Amazon DynamoDB

Make model from scratch

Selecting this option means you will have to create all tables, GSIs, attributes and elements yourself.

Start from a template

Start with tables, GSIs and attributes to help guide you without losing any freedom to change everything.

AWS Discussion Forum Data Model

This data model represents Amazon DynamoDB schema for AWS discussion forums, an example of an application for discussion forums or message boards....

Bookmarks Data Model

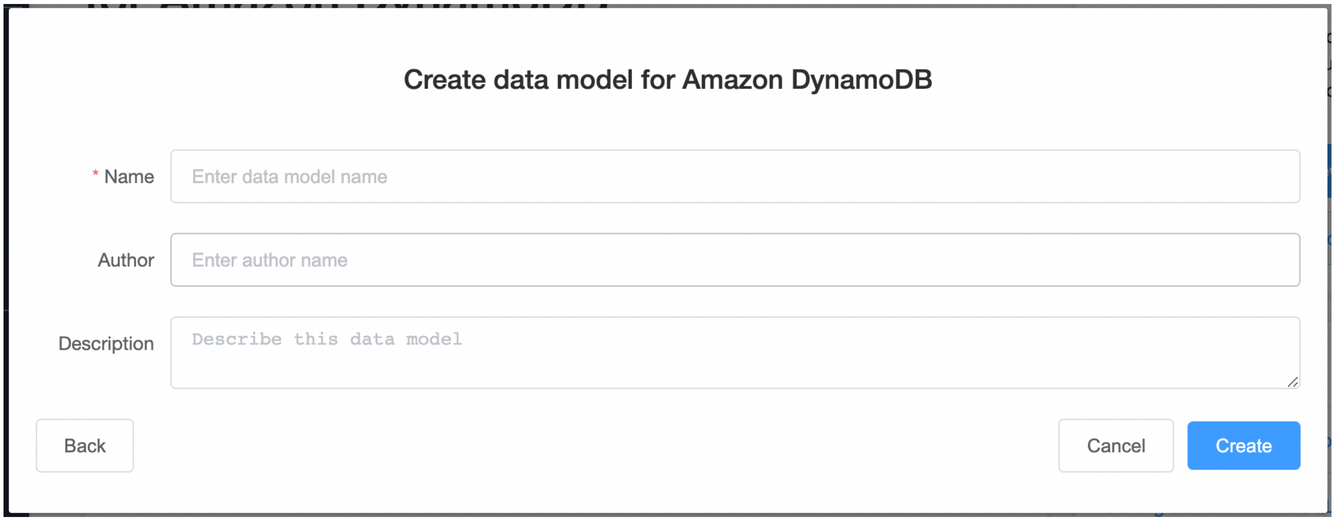
This model is about storing URL bookmarks for customers. Even if the use case is relatively simple, there are still many interesting...

Employee Data Model

This data model represents an Amazon DynamoDB schema for an employee database application. The important access patterns facilitated by this data...

Make model from scratch

モデルをゼロから作成するには、データモデルの名前、作成者、説明を入力します。終了したら、[Create] (作成) を選択します。



Create data model for Amazon DynamoDB

* Name: Enter data model name

Author: Enter author name

Description: Describe this data model

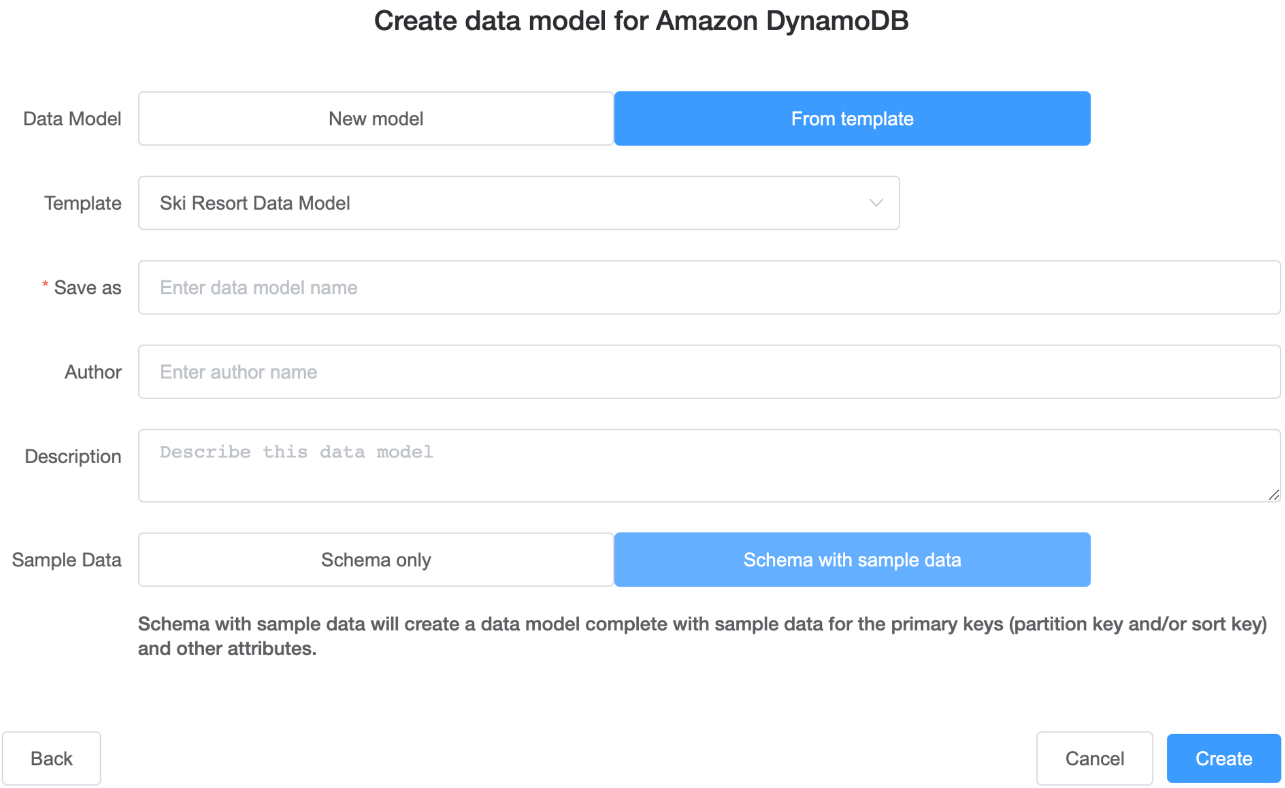
Start from a template

テンプレートから開始すると、開始するサンプルモデルを選択できます。その他のテンプレートオプションを表示するには、[More templates] (その他のテンプレート) を選択します。使用するテンプレートの[Select] (選択) を選びます。

選択したテンプレートのデータモデル名、作成者、および説明を入力します。[Schema only] (スキーマのみ) または [Schema with sample data] (サンプルデータを含むスキーマ) を選択できます。

- [Schema only] (スキーマのみ) は、プライマリキー (パーティションとソートキー) とその他の属性を持つ空のデータモデルを作成します。
- [Schema with sample data] (サンプルデータを含むスキーマ) は、プライマリキー (パーティションとソートキー) とその他の属性のサンプルデータを含むデータモデルを作成します。

この情報が入力できたら、[Create] (作成) を選択してモデルを作成します。



Create data model for Amazon DynamoDB

Data Model: New model / From template

Template: Ski Resort Data Model

* Save as: Enter data model name

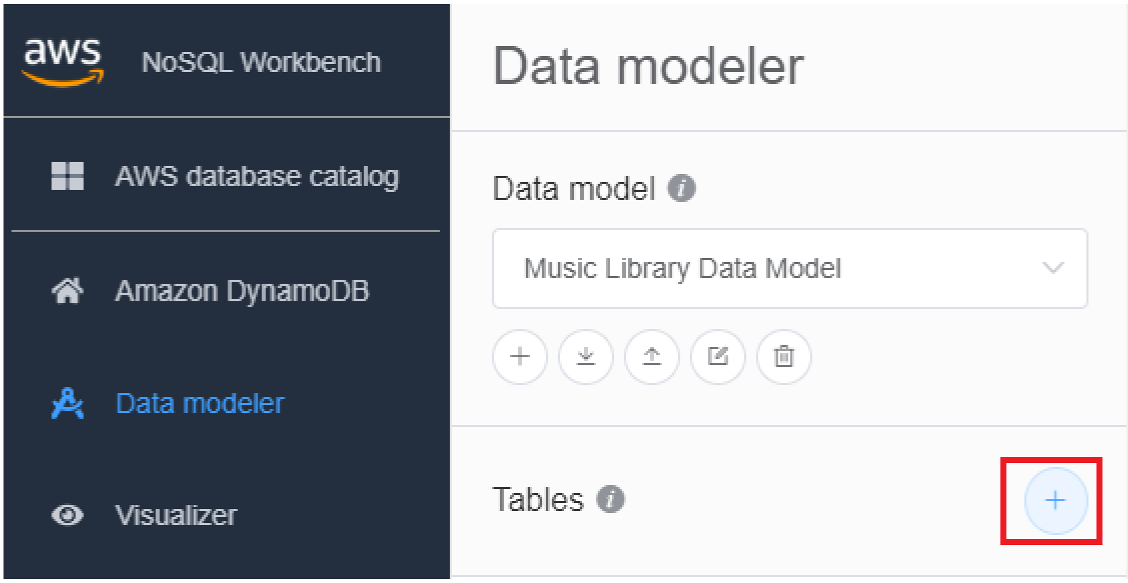
Author: Enter author name

Description: Describe this data model

Sample Data: Schema only / Schema with sample data

Schema with sample data will create a data model complete with sample data for the primary keys (partition key and/or sort key) and other attributes.

3. モデルを作成したら、[Add table] (テーブルを追加) を選択します。



テーブルの詳細については、「DynamoDB でのテーブルの使用」を参照してください。

4. 次を指定します:

   - [Table name] (テーブル名) – テーブルの一意な名前を入力します。
   - パーティションキー – パーティションキー名を入力し、そのタイプを指定します。オプションで、サンプルデータを生成するためのより詳細なデータ型形式を選択することもできます。
   - ソートキーを追加する場合:
     1. [Add sort key (ソートキーの追加)] を選択します。
     2. ソートキー名とそのタイプを指定します オプションで、サンプルデータを生成するためのより詳細なデータ型形式を選択することができます。

> Note
>
> プライマリキーの設計、パーティションキーの効果的な設計と使用、およびソートキーの使用の詳細については、以下を参照してください。
>
> - プライマリキー
> - パーティションキーを効率的に設計し、使用するためのベストプラクティス
> - ソートキーを使用してデータを整理するためのベストプラクティス

5. 他の属性を追加するには、各属性に対して以下を実行します。

   1. [属性の追加] を選択します。

2. 属性名とタイプを指定します。オプションで、サンプルデータを生成するためのより詳細なデータ型形式を選択することができます。

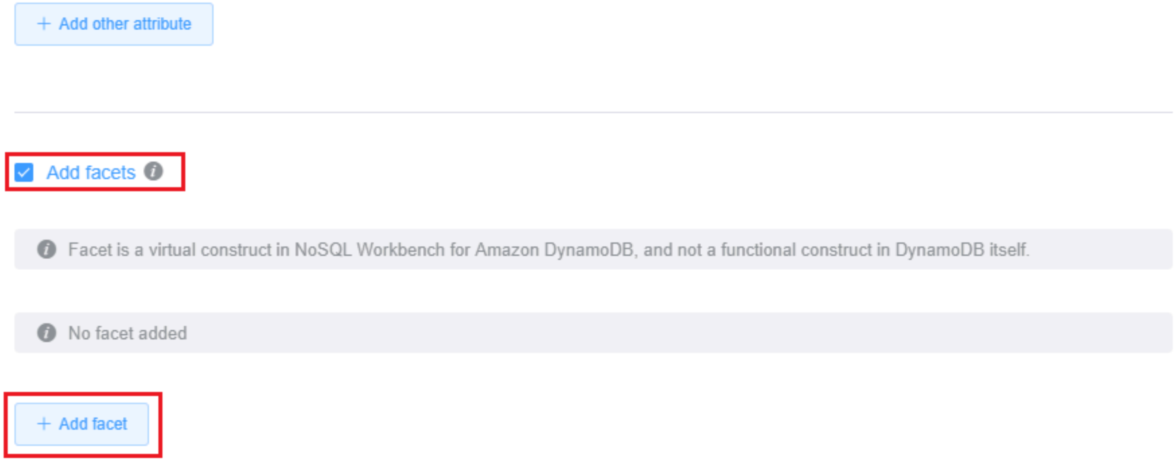
6. ファセットを追加する:

   必要に応じてファセットを追加できます。ファセットは NoSQL Workbench の仮想コンストラクトです。これは DynamoDB 自体の機能的コンストラクトではありません。

   > **Note**
   >
   > NoSQL Workbench のファセットは、テーブル内のデータのサブセットのみを使用して、Amazon DynamoDB に対するアプリケーションのさまざまなデータアクセスパターンを視覚化するのに役立ちます。ファセットの詳細については、「データアクセスパターンの表示」を参照してください。

   ファセットを追加するには、

   - [Add facets (ファセットの追加)] を選択します。
   - [Add facet (ファセットの追加)] を選択します。

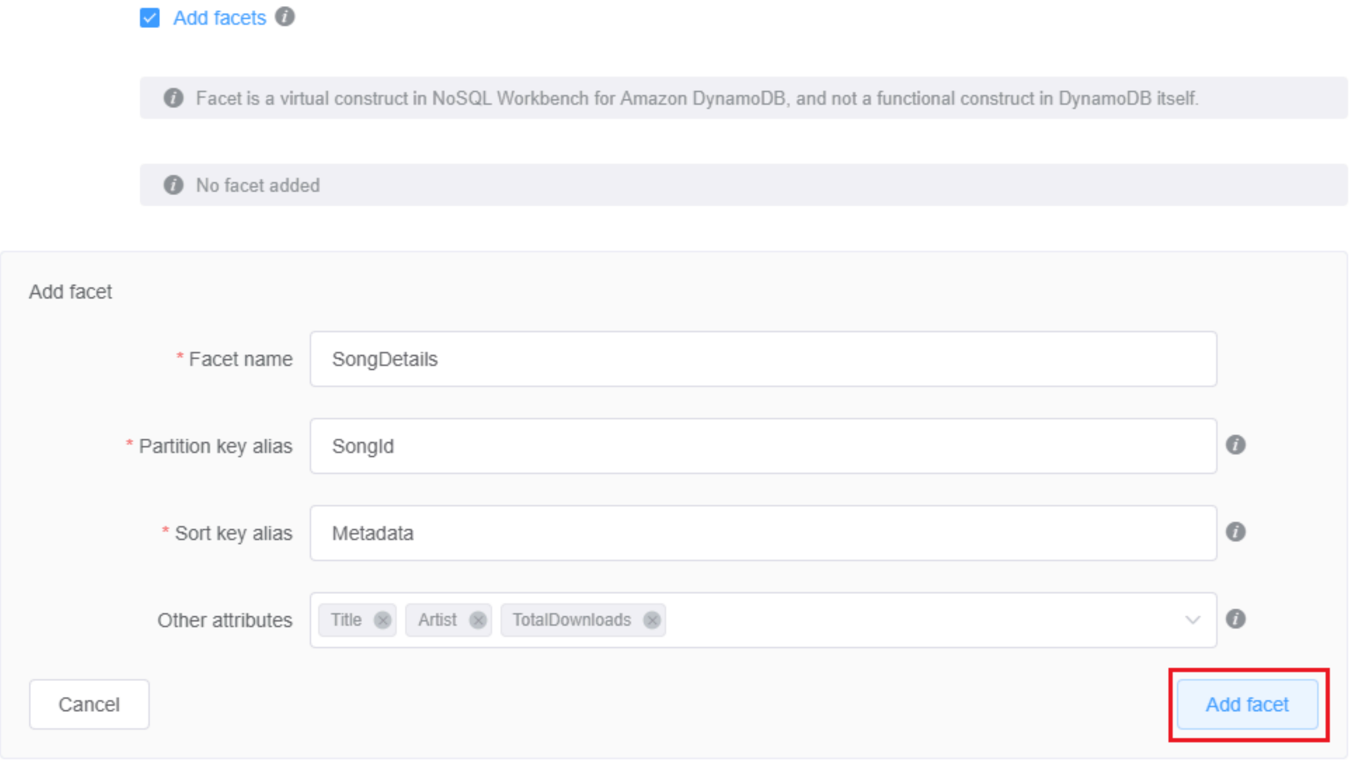


Facet is a virtual construct in NoSQL Workbench for Amazon DynamoDB, and not a functional construct in DynamoDB itself.

No facet added

   - 次を指定します:
     - Facet name (ファセット名)。
     - このファセットビューを区別するのに役立つパーティションキーエイリアス。
     - Sort key alias (ソートキーのエイリアス)。
     - このファセットの一部である [Other attributes (その他の属性)] を選択します。

[Add facet (ファセットの追加)] を選択します。



☑ Add facets

Facet is a virtual construct in NoSQL Workbench for Amazon DynamoDB, and not a functional construct in DynamoDB itself.

No facet added

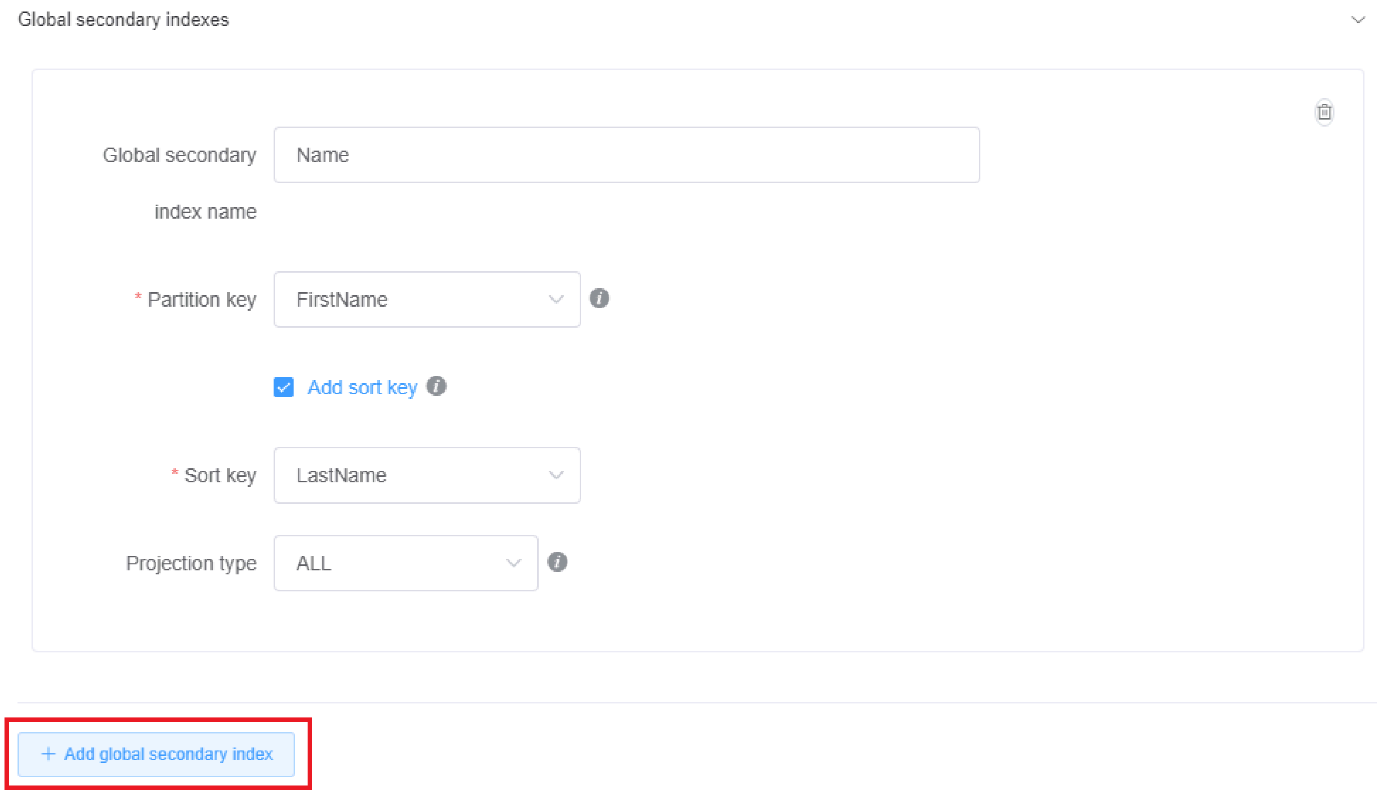
Add facet

* Facet name	SongDetails
* Partition key alias	SongId
* Sort key alias	Metadata
Other attributes	Title ⊗ Artist ⊗ TotalDownloads ⊗

さらにファセットを追加する場合は、このステップを繰り返します。

7. グローバルセカンダリインデックスを追加する場合は、[Add global secondary index (グローバルセカンダリインデックスの追加)] を選択します。

   [Global secondary index name (グローバルセカンダリインデックス名)]、[Partition key (パーティションキー)] 属性、および [Projection type (プロジェクションタイプ)] を指定します。



Global secondary indexes

Global secondary index name: Name

* Partition key: FirstName

☑ Add sort key

* Sort key: LastName

Projection type: ALL

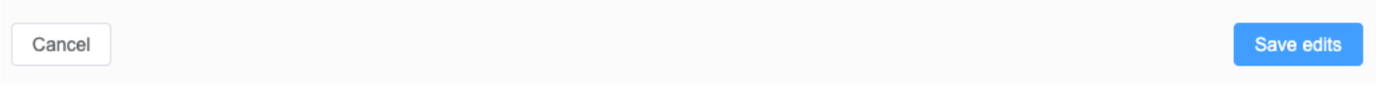
DynamoDB でグローバルセカンダリインデックスを使用する詳細方法については、「グローバルセカンダリインデックス」を参照してください。

8. デフォルトでは、テーブルは、読み込み容量と書き込み容量の両方で Auto Scaling を有効にしたプロビジョンされた容量モードを使用します。これらの設定を変更する場合は、[キャパシティ設定] で [デフォルト設定] のチェックを外してください。

   必要な容量モード、読み込みおよび書き込み容量、Auto Scaling IAM ロール (該当する場合) を選択します。

   DynamoDB 容量設定についての詳細は、DynamoDB のスループットキャパシティ を参照してください。

9. 編集内容をテーブル設定に保存します。



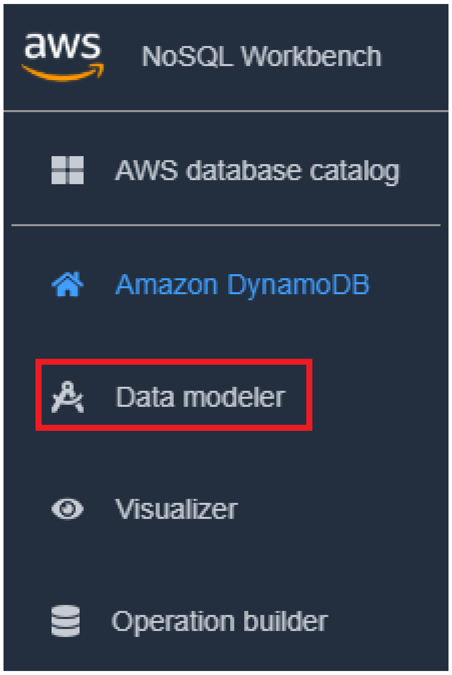
CreateTable API オペレーションの詳細については、Amazon DynamoDB API リファレンスの CreateTable を参照してください。

既存のデータモデルのインポート

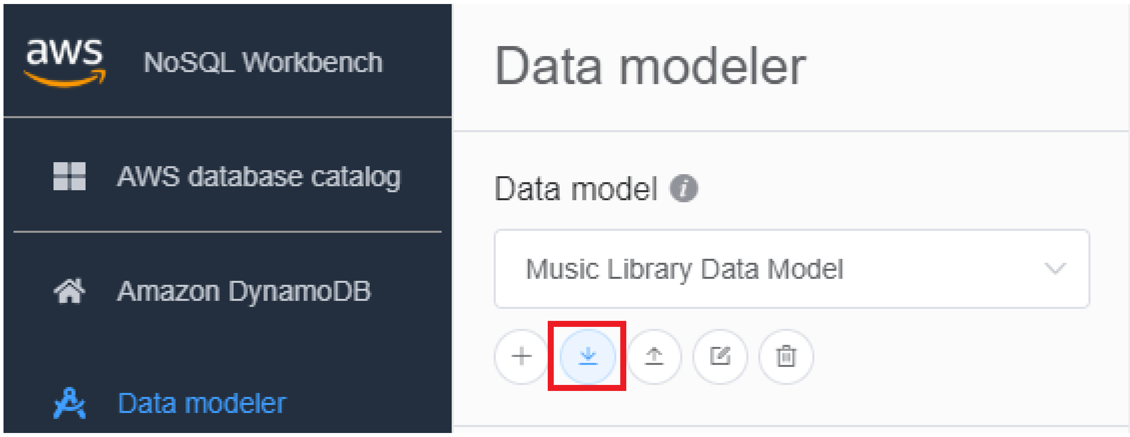
Amazon DynamoDB 用 NoSQL Workbench を使用して、既存のモデルをインポートおよび変更することでデータモデルを構築できます。データモデルは、NoSQL Workbench モデル形式または AWS CloudFormation JSON テンプレート形式のいずれかでインポートできます。

データモデルをインポートするには

1. NoSQL Workbench の左側のナビゲーションペインで [Data modeler (データモデラー)] アイコンを選択します。

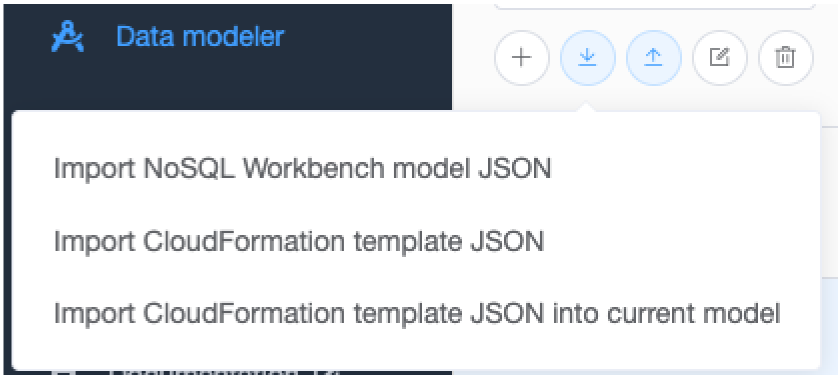


2. [Import data model] (データモデルのインポート) にポインタを合わせます。

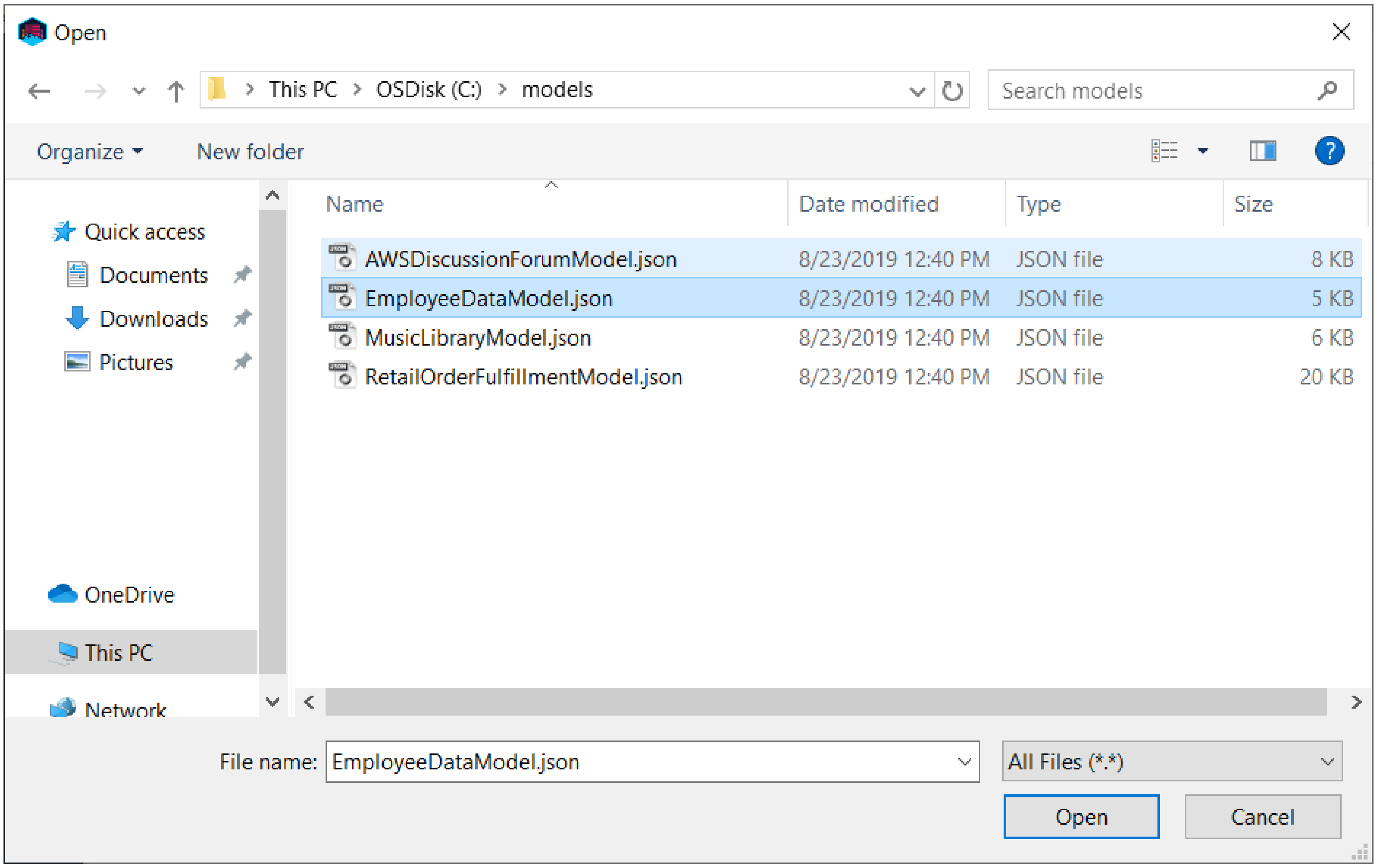


ドロップダウンリストで、インポートするモデルが NoSQL Workbench モデル形式か CloudFormation JSON テンプレート形式かを選択します。NoSQL Workbench で既存のテーブ

ルデータモデルを開いている場合は、CloudFormation テンプレートを現在のモデルにインポートするオプションがあります。


Data modeler
Import NoSQL Workbench model JSON
Import CloudFormation template JSON
Import CloudFormation template JSON into current model

3. インポートするモデルを選択します。

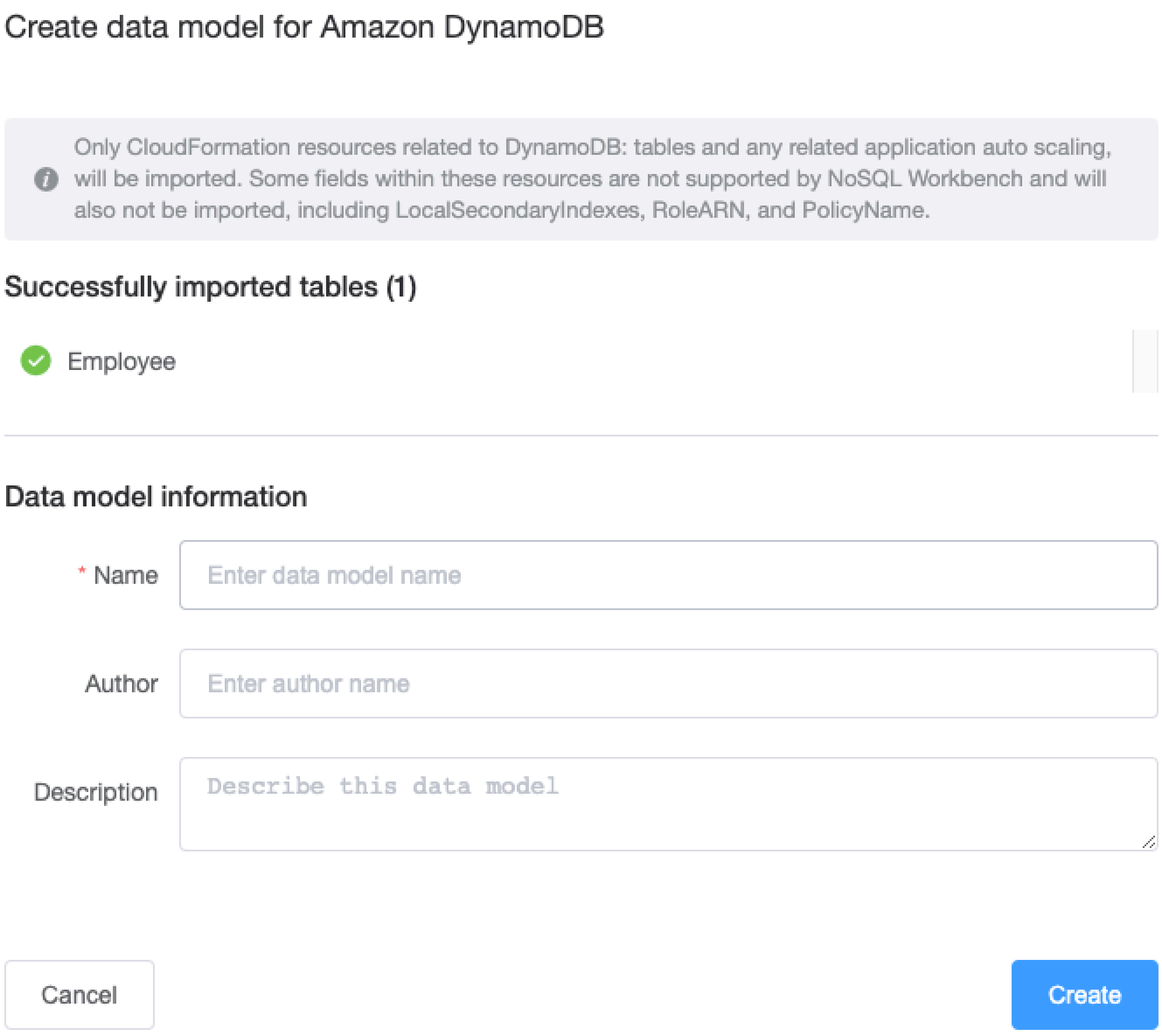


Name	Date modified	Type	Size
AWSDiscussionForumModel.json	8/23/2019 12:40 PM	JSON file	8 KB
EmployeeDataModel.json	8/23/2019 12:40 PM	JSON file	5 KB
MusicLibraryModel.json	8/23/2019 12:40 PM	JSON file	6 KB
RetailOrderFulfillmentModel.json	8/23/2019 12:40 PM	JSON file	20 KB

File name: EmployeeDataModel.json

All Files (*.*)

4. インポートするモデルが CloudFormation テンプレート形式の場合、インポートするテーブルのリストが表示され、データモデル名、作成者、説明を指定するオポチュニティがあります。



Create data model for Amazon DynamoDB

Only CloudFormation resources related to DynamoDB: tables and any related application auto scaling, will be imported. Some fields within these resources are not supported by NoSQL Workbench and will also not be imported, including LocalSecondaryIndexes, RoleARN, and PolicyName.

Successfully imported tables (1)

✓ Employee

Data model information

* Name: Enter data model name

Author: Enter author name

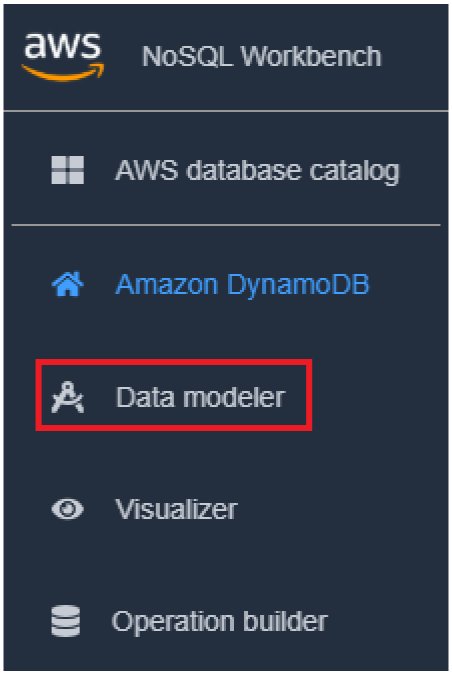
Description: Describe this data model

データモデルのエクスポート

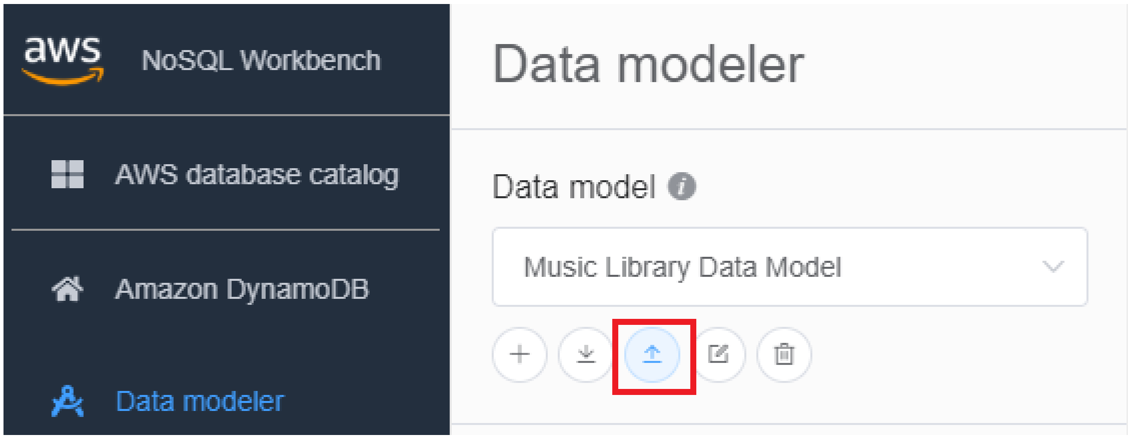
Amazon DynamoDB 用 NoSQL Workbench を使用してデータモデルを作成した後、モデルを NoSQL Workbench モデル形式または AWS CloudFormation JSON テンプレート形式で保存およびエクスポートできます。

データモデルをエクスポートするには

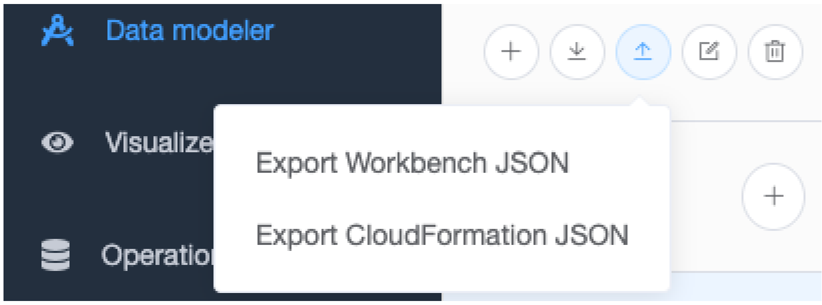
1. NoSQL Workbench の左側のナビゲーションペインで [Data modeler (データモデラー)] アイコンを選択します。



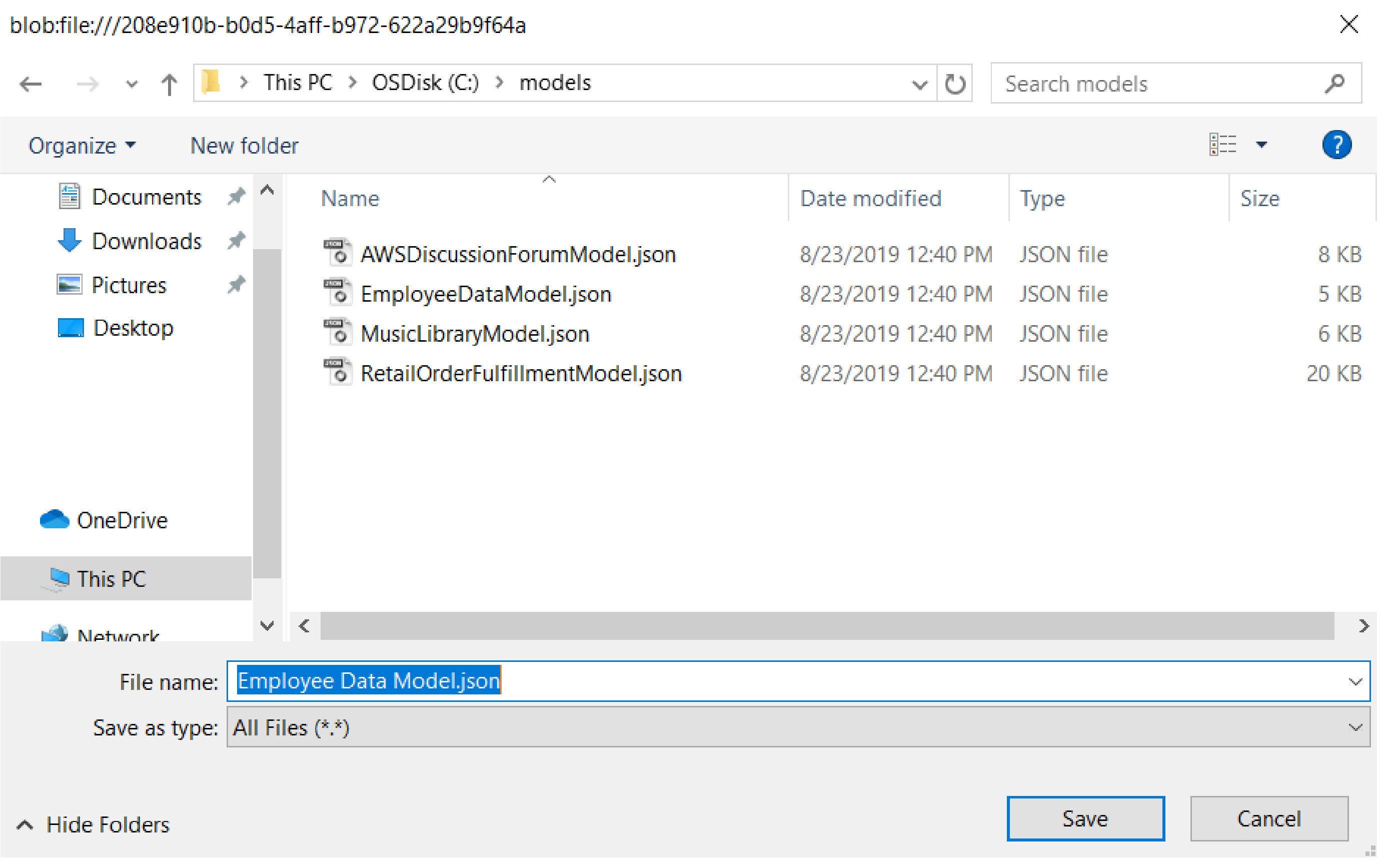
2. [Export data model] (データモデルのエクスポート) にポインタを合わせます。



ドロップダウンリストで、データモデルを NoSQL Workbench モデル形式または CloudFormation JSON テンプレート形式でエクスポートするかどうかを選択します。



3. モデルを保存する場所を選択します。



Name	Date modified	Type	Size
AWSDiscussionForumModel.json	8/23/2019 12:40 PM	JSON file	8 KB
EmployeeDataModel.json	8/23/2019 12:40 PM	JSON file	5 KB
MusicLibraryModel.json	8/23/2019 12:40 PM	JSON file	6 KB
RetailOrderFulfillmentModel.json	8/23/2019 12:40 PM	JSON file	20 KB

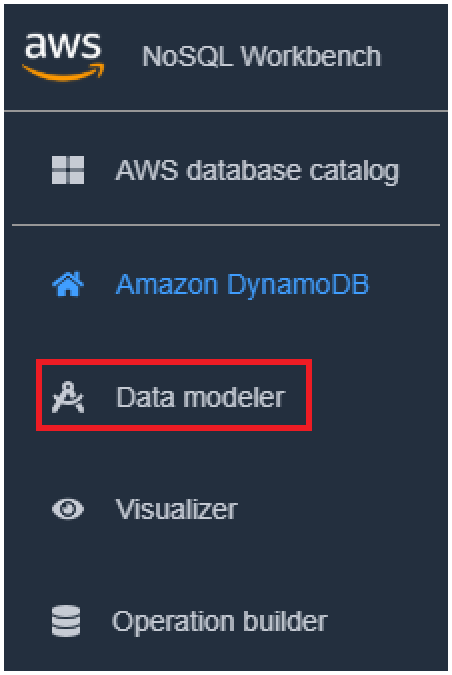
File name: Employee Data Model.json

Save as type: All Files (*.*)

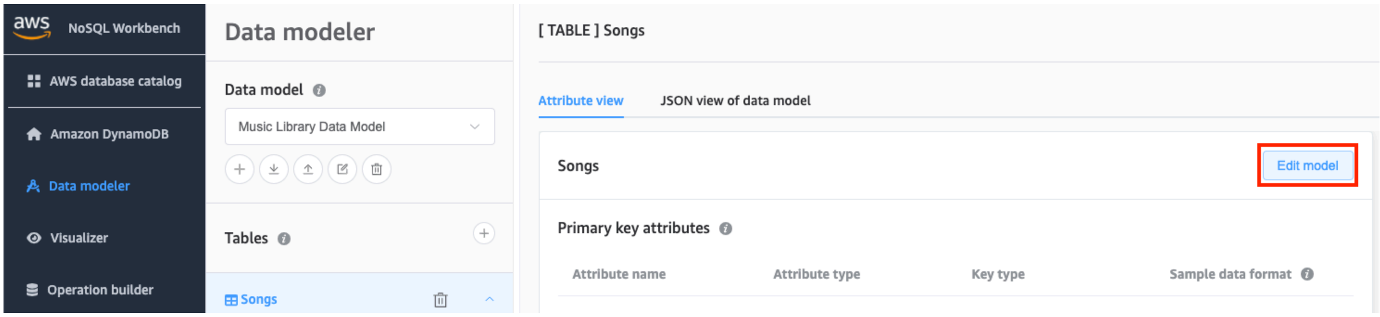
既存のデータモデルの編集

既存のモデルを編集するには

1. NoSQL Workbench の左側のナビゲーションペインで [Data modeler (データモデラー)] ボタンを選択します。



2. データモデルを選択し、編集するテーブルを選択します。[モデルの編集] を選択します。



3. 必要な編集を行い、[Save edits (編集を保存)] を選択します。

既存のモデルを手動で編集してファセットを追加するには

1. モデルをエクスポートします。詳細については、「」を参照してくださいデータモデルのエクスポート
2. エクスポートしたファイルをエディタで開きます。
3. ファセットを作成するテーブルの DataModel オブジェクトを見つけます。

   テーブルのすべてのファセットを表す TableFacets 配列を追加します。

   各ファセットについて、TableFacets 配列にオブジェクトを追加します。各配列要素には以下のプロパティがあります。

- `FacetName` – ファセットの名前。この値はモデル全体で一意である必要があります。
- `PartitionKeyAlias` – テーブルのパーティションキーのフレンドリ名。このエイリアスは、NoSQL Workbench でファセットを表示するときに表示されます。
- `SortKeyAlias` – テーブルのソートキーのフレンドリ名。このエイリアスは、NoSQL Workbench でファセットを表示するときに表示されます。テーブルにソートキーが定義されていない場合、このプロパティは必要ありません。
- `NonKeyAttributes` – アクセスパターンに必要な属性名の配列。これらの名前は、テーブルに定義されている属性名にマップする必要があります。

```
{
  "ModelName": "Music Library Data Model",
  "DataModel": [
    {
      "TableName": "Songs",
      "KeyAttributes": {
        "PartitionKey": {
          "AttributeName": "Id",
          "AttributeType": "S"
        },
        "SortKey": {
          "AttributeName": "Metadata",
          "AttributeType": "S"
        }
      },
      "NonKeyAttributes": [
        {
          "AttributeName": "DownloadMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloadsInMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Title",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Artist",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloads",
          "AttributeType": "S"
        },
        {
          "AttributeName": "DownloadTimestamp",
          "AttributeType": "S"
        }
      ],
      "TableFacets": [
        {
          "FacetName": "SongDetails",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "Title",
            "Artist",
            "TotalDownloads"
          ]
        },
        {
          "FacetName": "Downloads",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "DownloadTimestamp"
          ]
        }
      ]
    }
  ]
}
```

4. これで、変更したモデルを NoSQL Workbench にインポートできます 詳細については、「」を参照してください既存のデータモデルのインポート

データアクセスパターンの可視化

Amazon DynamoDB 用の NoSQL Workbench のビジュアライザーツールを使用して、クエリをマッピングし、アプリケーションのさまざまなアクセスパターン (ファセットと呼ばれる) を可視化できます。すべてのファセットは、DynamoDB の異なるアクセスパターンに対応しています。また、データモデルにデータを手動で追加したり、MySQL からデータをインポートしたりできます。

トピック

データモデルへのサンプルデータの追加

モデルにサンプルデータを追加すると、モデルとそのさまざまなデータアクセスパターン (ファセット) を可視化してデータを表示できます。

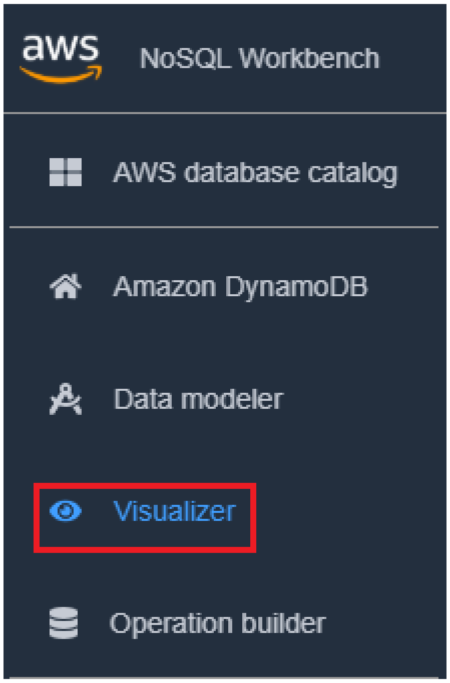
サンプルデータを追加するには 2 つの方法があります。1 つは、サンプルデータ自動生成ツールを使用する方法です。もう 1 つは、データを 1 つずつ追加する方法です。

Amazon DynamoDB 用の NoSQL Workbench を使用して、データモデルにサンプルデータを追加するには、以下のステップに従います。

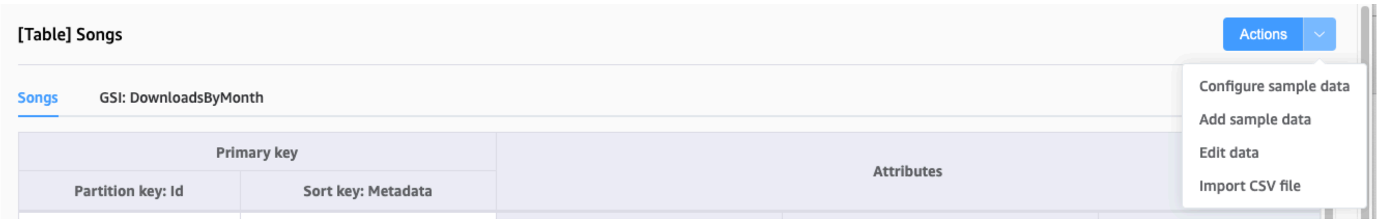
サンプルデータを自動生成するには

サンプルデータを自動生成すると、1 ~ 5000 行のデータを生成して、それをすぐに使用できます。きめ細かいサンプルデータ型を指定して、設計やテストのニーズに基づいた現実的なデータを作成できます。現実的なデータを生成する機能を利用するには、データモデラーで属性のサンプルデータ型フォーマットを指定する必要があります。サンプルデータ型フォーマットの指定については、「新しいデータモデルの作成」を参照してください。

1. 左側にあるナビゲーションペインで、[visualizer] (ビジュアライザー) アイコンを選択します。



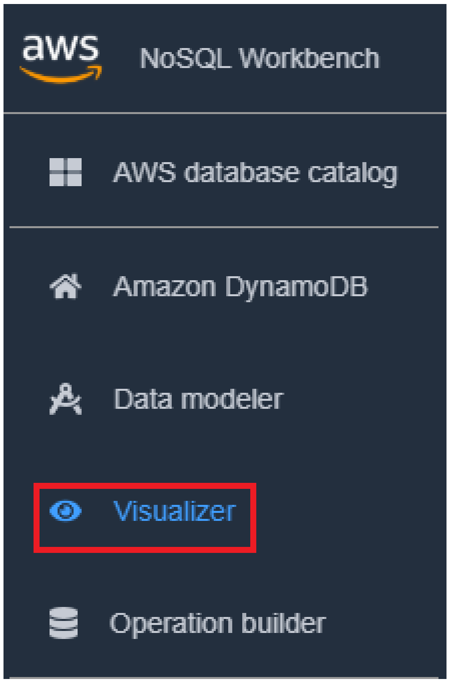
2. ビジュアライザーで、データ型を選択し、テーブルを選択します。
3. [アクション] ドロップダウンを選択し、[サンプルデータの追加] を選択します。



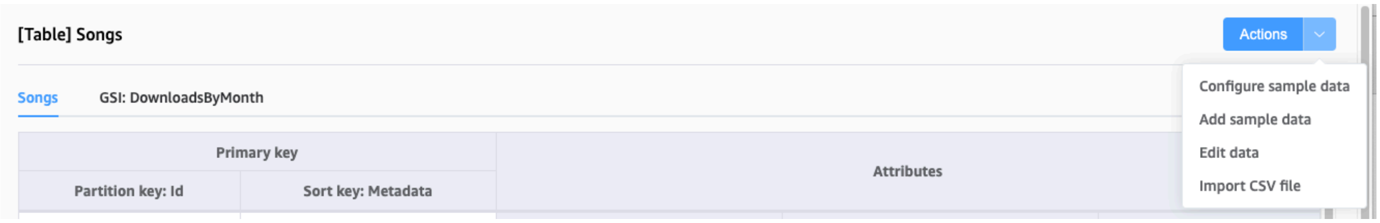
4. 生成するサンプルデータの数または項目を入力し、[確認] を選択します。

サンプルデータを 1 つずつ追加するには

1. 左側にあるナビゲーションペインで、[visualizer] (ビジュアライザー) アイコンを選択します。



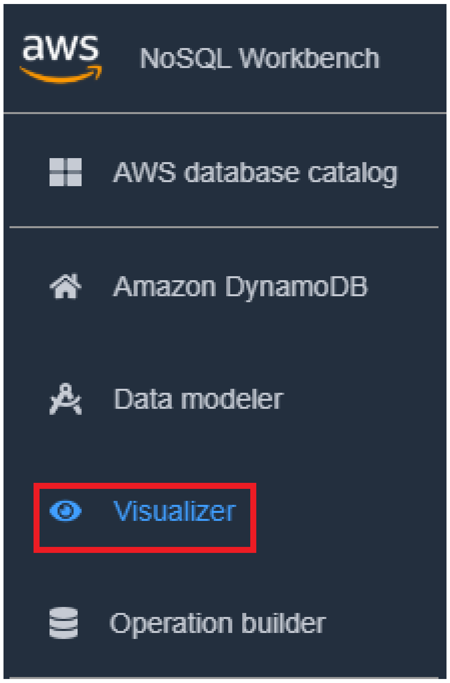
2. ビジュアライザーで、データ型を選択し、テーブルを選択します。
3. [アクション] ドロップダウンを選択し、[データの編集] を選択します。



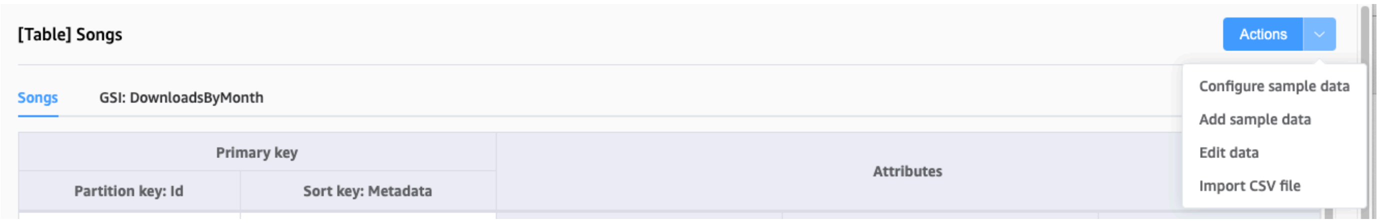
4. [新しいルールを追加] を選択します。空のテキストボックスにサンプルデータを入力し、[新しい行を追加] を再選択して行を追加します。完了したら、[変更を保存] を選択します。

サンプルデータを削除するには

1. 左側にあるナビゲーションペインで、[visualizer] (ビジュアライザー) アイコンを選択します。



2. ビジュアライザーで、データ型を選択し、テーブルを選択します。
3. [アクション] ドロップダウンを選択し、[データの編集] を選択します。



4. 削除したいデータの各行の横にある削除アイコンを選択します。

CSV ファイルからサンプルデータのインポート

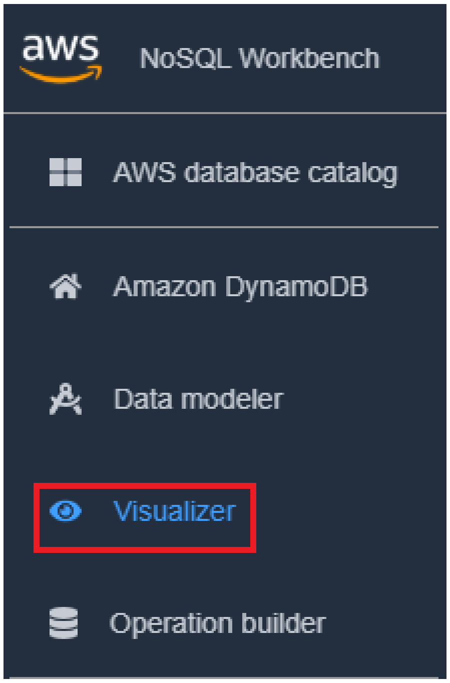
CSV ファイル内に既存のサンプルデータがある場合は、それを NoSQL Workbench にインポートすることができます。これにより、行単位で入力することなく、サンプルデータをモデルにすばやく入力できるようになります。

CSV ファイルの列名は、データモデルの属性名と一致する必要がありますが、同じ順序である必要はありません。例えば、データモデルに `LoginAlias`、`FirstName`、`LastName` と呼ばれる属性がある場合の場合、CSV 列は `LastName`、`FirstName`、`LoginAlias` になる可能性があります。

CSV ファイルからのデータのインポートは、1 回につき 150 行に制限されています。

CSV ファイルから NoSQL Workbench にデータをインポートするには

1. 左側にあるナビゲーションペインで、[visualizer] (ビジュアライザー) アイコンを選択します。



2. ビジュアライザーで、データ型を選択し、テーブルを選択します。
3. [アクション] ドロップダウンを選択し、[データの編集] を選択します。
4. [アクション] ドロップダウンを再び選択し、[CSV ファイルのインポート] を選択します。
5. CSV ファイルを選択して [Open] (開く) を選択します。CSV ファイルのデータがテーブルに追加されます。

> Note
>
> CSV ファイルに、テーブルに既に存在する項目と同じキーを持つ行が含まれている場合は、既存の項目に上書きするか、テーブルの最後に追加するかを選択できます。項目の追加を選択した場合、重複する項目とテーブル内の既存の項目を区別するために、重複する項目のキーにサフィックスの「-Copy」が追加されます。

データアクセスパターンの表示

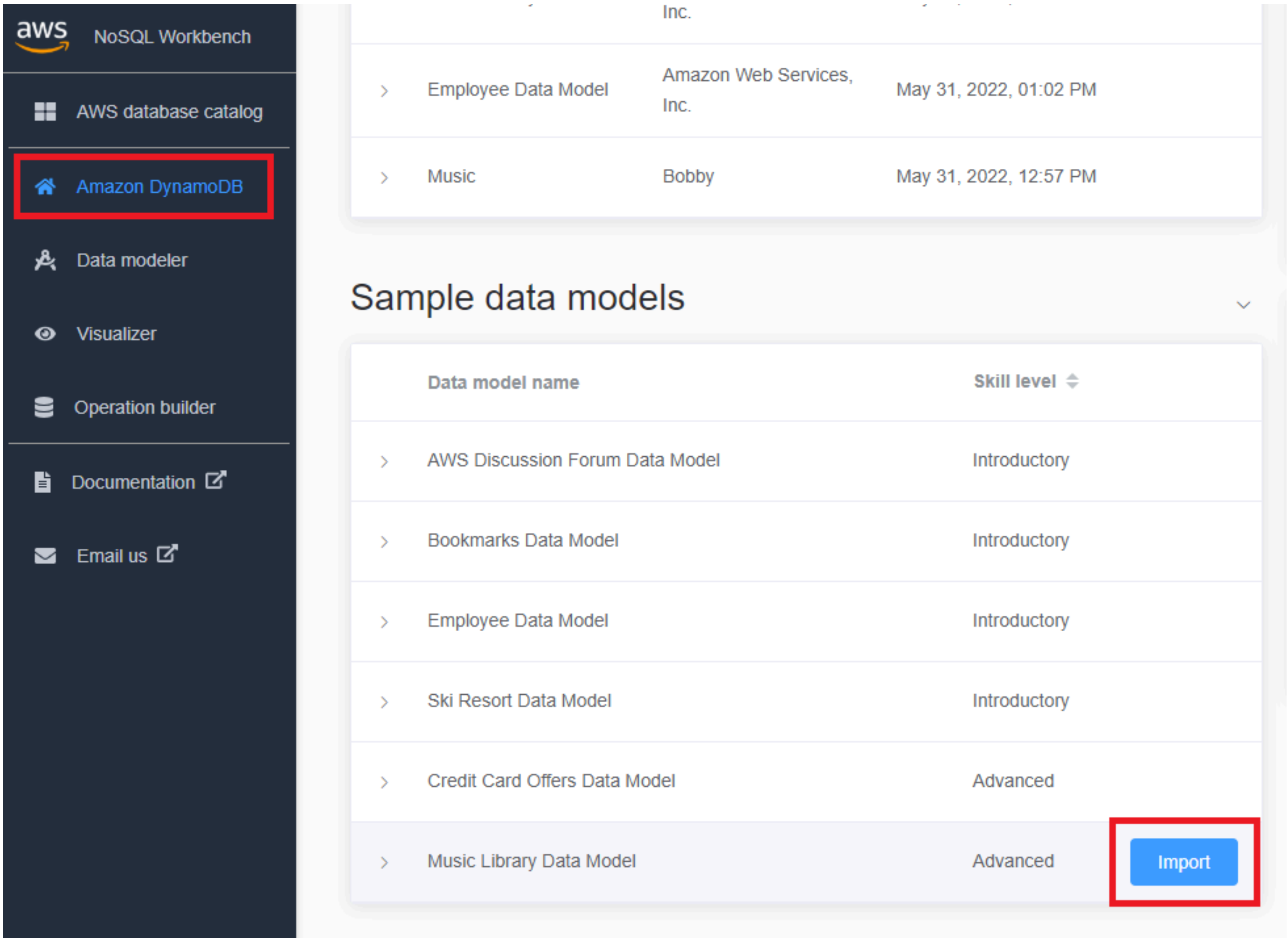
NoSQL Workbench のファセットとは、Amazon DynamoDB に対するアプリケーションのさまざまなデータアクセスパターンを表します。複数のデータ型がソートキーで表される場合、ファセットはデータ型を視覚化するのに役立ちます。ファセットを使用すると、ファセットの制約を満たさないレ

コードを表示することなく、テーブル内のデータのサブセットを表示できます。ファセットはビジュアルデータモデリングツールと見なされ、アクセスパターンのモデリングを純粋に補助するものであるため、DynamoDB で使用できる構造としては存在しません。

ファセットの例を確認するには、ファセット付きのサンプルデータ型のいずれかをデータ型テンプレートの一部としてインポートできます。

サンプルデータモデルのインポート

1. 左側にある [Amazon DynamoDB] を選択します。
2. [Sample data models] (サンプルデータモデル) セクションで、[Music Library Data Model] (ミュージックライブラリデータモデル) の上にポインタを移動し、[Import] (インポート) を選択します。

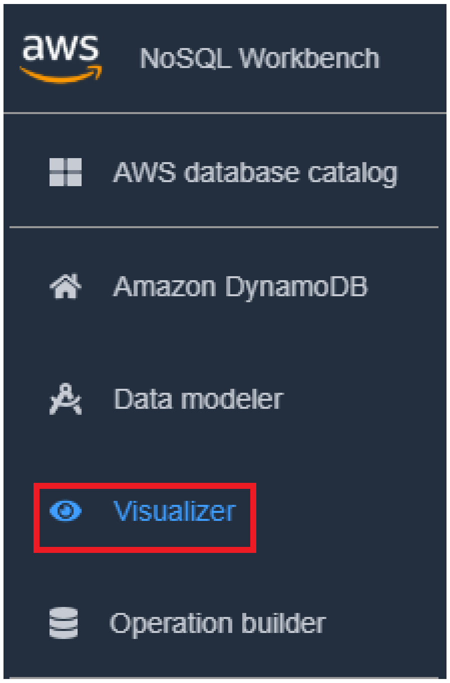


	Inc.	
Employee Data Model	Amazon Web Services, Inc.	May 31, 2022, 01:02 PM
Music	Bobby	May 31, 2022, 12:57 PM

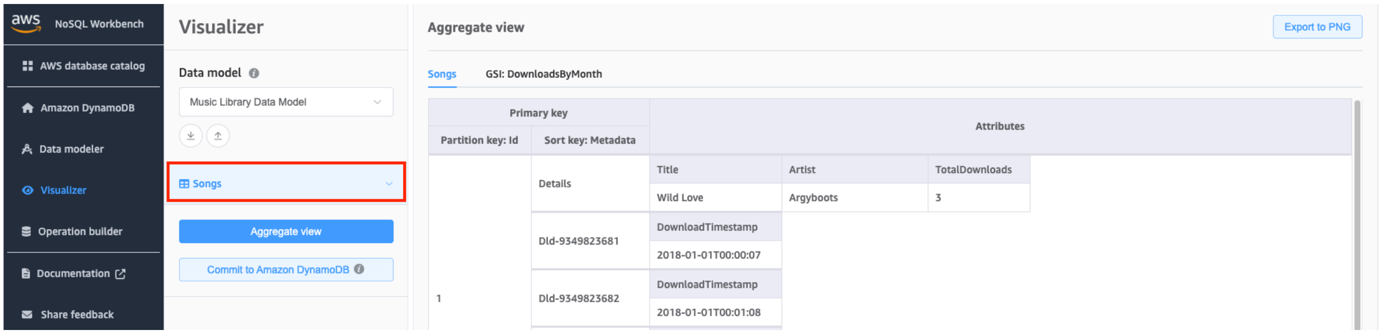
Sample data models

Data model name	Skill level
AWS Discussion Forum Data Model	Introductory
Bookmarks Data Model	Introductory
Employee Data Model	Introductory
Ski Resort Data Model	Introductory
Credit Card Offers Data Model	Advanced
Music Library Data Model	Advanced

3. 左側のナビゲーションペインで、[visualizer] (ビジュアライザー) アイコンを選択します。



4. [Songs] (曲) テーブルを選択して展開します。データの集計ビューが表示されます。



Data model: Music Library Data Model

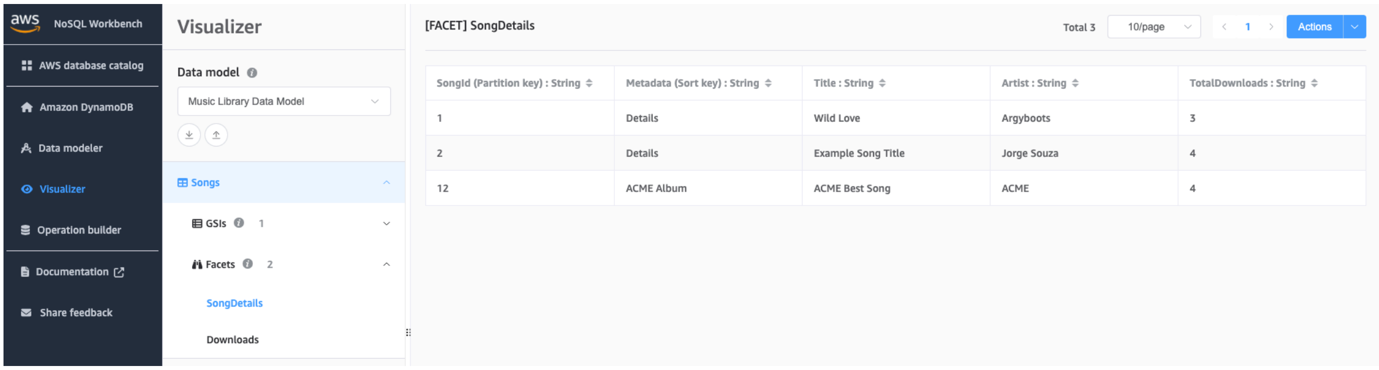
Aggregate view

Songs GSI: DownloadsByMonth

Primary key		Attributes		
Partition key: Id	Sort key: Metadata			
1	Details	Title Wild Love	Artist Argyboots	TotalDownloads 3
	Dld-9349823681	DownloadTimestamp 2018-01-01T00:00:07		
	Dld-9349823682	DownloadTimestamp 2018-01-01T00:01:08		

5. [Facets] (ファセット) ドロップダウンの矢印を選択し、使用可能なファセットを展開します。

6. SongDetails ファセットを選択して、SongDetails ファセットを適用したデータを可視化します。



Data model: Music Library Data Model

[FACET] SongDetails Total 3

SongId (Partition key) : String	Metadata (Sort key) : String	Title : String	Artist : String	TotalDownloads : String
1	Details	Wild Love	Argyboots	3
2	Details	Example Song Title	Jorge Souza	4
12	ACME Album	ACME Best Song	ACME	4

データモデラーを使用してファセット定義を編集することもできます。詳細については、「既存のデータモデルの編集」を参照してください。

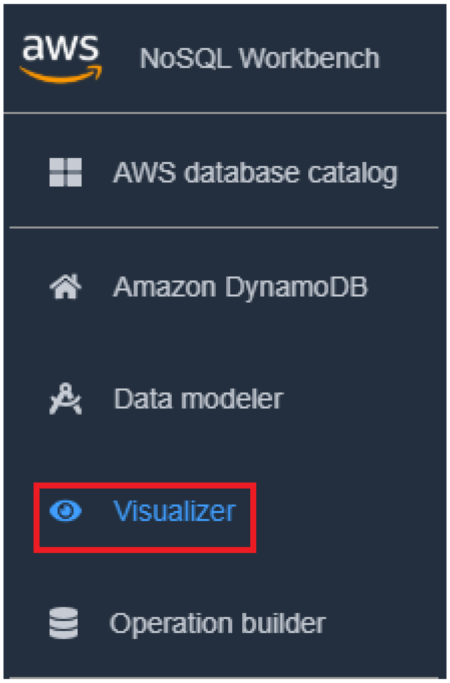
集計ビューを使用したデータモデル内のすべてのテーブルの表示

Amazon DynamoDB 用の NoSQL Workbench の集計ビューは、データモデルのすべてのテーブルを表します。各テーブルについて、以下の情報が表示されます。

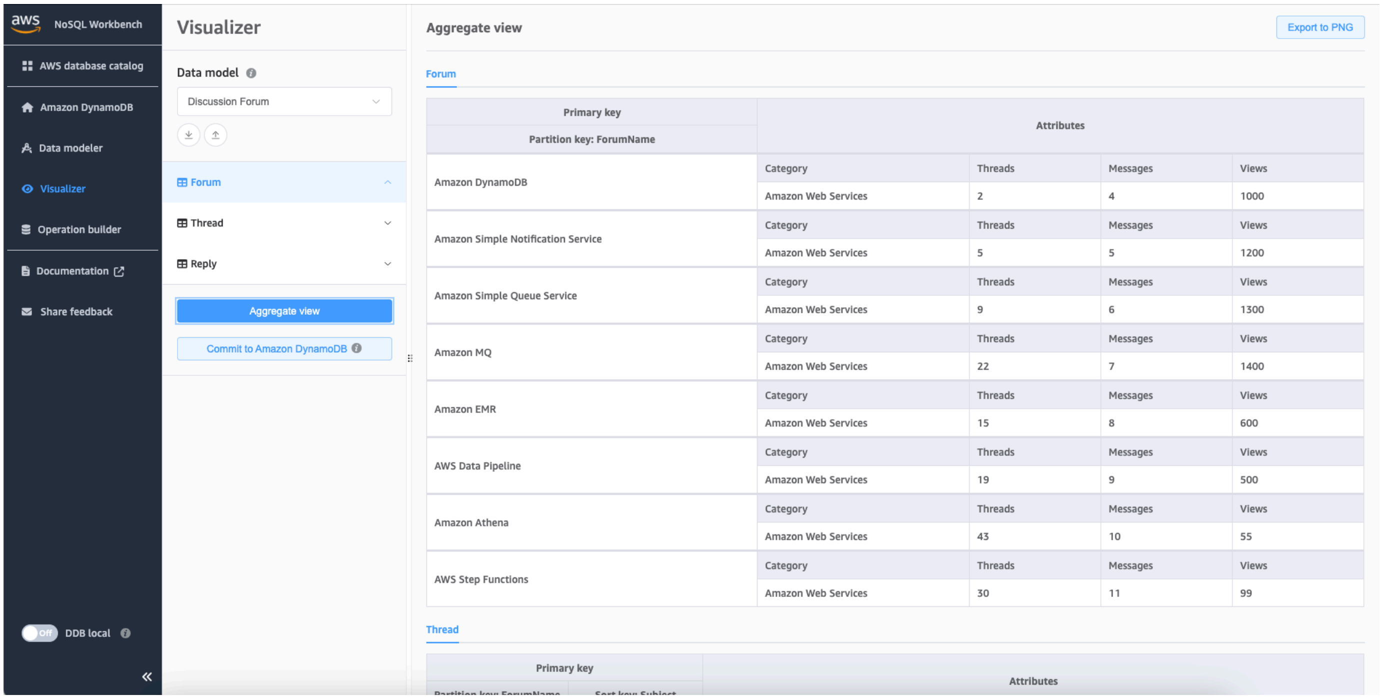
- テーブルの列名
- サンプルデータ
- テーブルに関連付けられているすべてのグローバルセカンダリインデックス。各インデックスに対して、以下の情報が表示されます。
  - インデックス列名
  - サンプルデータ

すべてのテーブル情報を表示するには

1. 左側にあるナビゲーションペインで、[visualizer (ビジュアライザー)] アイコンを選択します。



2. ビジュアライザーで、[Aggregate view (集計ビュー)] を選択します。



Forum

Primary key	Attributes			
Partition key: ForumName				
Amazon DynamoDB	Category	Threads	Messages	Views
	Amazon Web Services	2	4	1000
Amazon Simple Notification Service	Category	Threads	Messages	Views
	Amazon Web Services	5	5	1200
Amazon Simple Queue Service	Category	Threads	Messages	Views
	Amazon Web Services	9	6	1300
Amazon MQ	Category	Threads	Messages	Views
	Amazon Web Services	22	7	1400
Amazon EMR	Category	Threads	Messages	Views
	Amazon Web Services	15	8	600
AWS Data Pipeline	Category	Threads	Messages	Views
	Amazon Web Services	19	9	500
Amazon Athena	Category	Threads	Messages	Views
	Amazon Web Services	43	10	55
AWS Step Functions	Category	Threads	Messages	Views
	Amazon Web Services	30	11	99

Thread

Primary key		Attributes
Partition key: ForumName	Sort key: Subject	

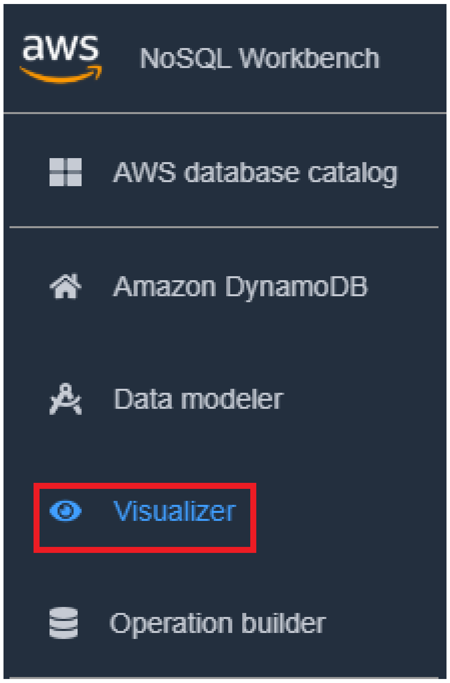
DynamoDB へのデータモデルのコミット

データモデルに満足したら、モデルを Amazon DynamoDB にコミットできます。

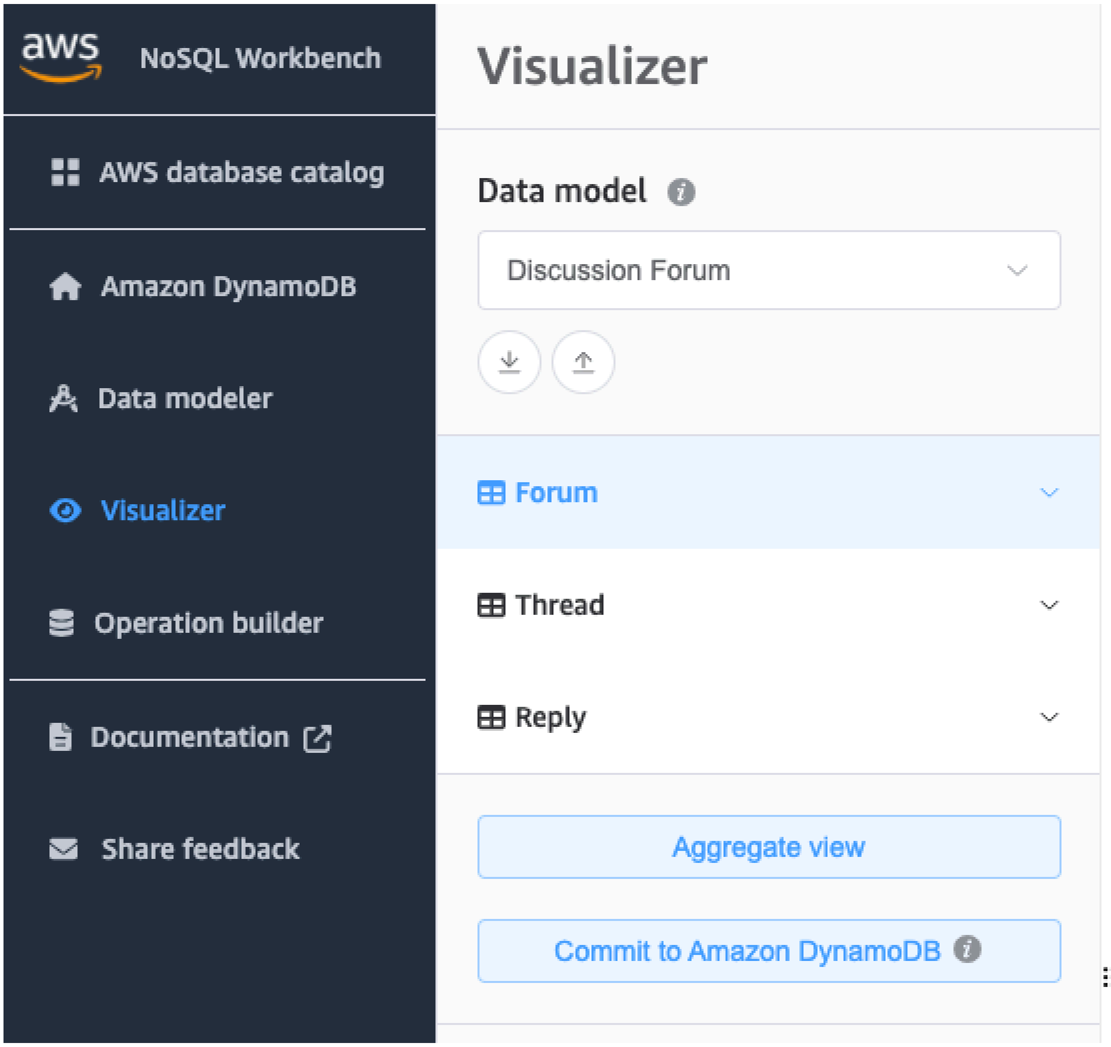
> **Note**
>
> - このアクションにより、データモデルで表されるテーブルとグローバルセカンダリインデックス用のサーバー側リソースが AWS に作成されます。
> - テーブルは、以下の特性で作成されます。
>   - Auto Scaling は、ターゲット使用率 70% に設定されます。
>   - プロビジョニングされたキャパシティーは、5 つの読み取りキャパシティーユニットと 5 つの書き込みキャパシティーユニットに設定されます。
> - グローバルセカンダリインデックスは、10 個の読み取りキャパシティーユニットと 5 個の書き込みキャパシティーユニットのプロビジョニングキャパシティーで作成されます。

データモデルを DynamoDB にコミットするには

1. 左側にあるナビゲーションペインで、[visualizer (ビジュアライザー)] アイコンを選択します。



2. [Commit to DynamoDB (DynamoDB にコミット)] を選択します。



3. 既存の接続を選択するか、[Add new remote connection (新しいリモート接続の追加)] タブを選択して新しい接続を作成します。

- 新しい接続を追加するには、以下の情報を指定します。
  - アカウントエイリアス
  - AWS リージョン
  - アクセスキー ID
  - シークレットアクセスキー

  アクセスキーを取得する方法の詳細については、「AWS アクセスキーの取得」を参照してください。
- オプションで次の値を指定できます。
  - セッショントークン

    - IAM ロール ARN
  - 無料利用枠アカウントにサインアップせず、DynamoDB Local (ダウンロード可能バージョン) を使用する場合:
    1. [Add a new DynamoDB local connection] (新しい DynamoDB ローカル接続の追加) タブを選択します。
    2. [Connection name (接続名)] と [Port (ポート)] を指定します。

4. [Commit (コミット)] を選択します。

> Note
>
> NoSQL Workbench のセットアップの一部として DynamoDB local をインストールした場合は、NoSQL Workbench 画面の左下にある [DynamoDB local サーバー] トグルを使用して、DynamoDB local をオンにする必要があります。このトグルの詳細については、「DynamoDB 用の NoSQL Workbench のインストール」を参照してください。

NoSQL Workbench を使用したデータセットの探索とオペレーションの構築

Amazon DynamoDB 用 NoSQL Workbench は、クエリを開発およびテストするための豊富なグラフィカルユーザーインターフェイスを提供します。NoSQL Workbench のオペレーションビルダーを使用して、ライブデータセットを表示、探索、およびクエリできます。構造化オペレーションビルダーは、射影式、条件式をサポートし、複数の言語でサンプルコードを生成します。ある Amazon DynamoDB アカウントのテーブルのクローンを別のリージョンの別のアカウントで直接作成できます。また、DynamoDB Local と Amazon DynamoDB アカウントの間で直接テーブルのクローンを作成すれば、テーブルのキースキーマ (オプションで GSI スキーマと項目) を開発環境間で迅速にコピーできます。オペレーションビルダーでは、最大 50 の DynamoDB データオペレーションを保存できます。

トピック

- ライブデータセットへの接続
- 複雑なオペレーションの構築
- NoSQL Workbench を使用したテーブルのクローン作成
- CSV ファイルへのデータのエクスポート

ライブデータセットへの接続

NoSQL Workbench を使用して Amazon DynamoDB テーブルに接続するには、まず AWS アカウントに接続する必要があります。

データベースへの接続を追加するには

1. NoSQL Workbench の左側のナビゲーションペインで [Operation builder (オペレーションビルダー)] アイコンを選択します。
2. [Add connection (接続の追加)] を選択します。
3. 以下の情報を指定します。

    - [Account alias (アカウントエイリアス)]
    - AWS リージョン
    - アクセスキー ID
    - シークレットアクセスキー

    アクセスキーを取得する方法の詳細については、「AWS アクセスキーの取得」を参照してください。

    オプションで、次の値を指定できます。

    - セッショントークン
    - IAM ロール ARN
4. [接続] を選択します。

    無料利用枠アカウントにサインアップせず、DynamoDB Local (ダウンロード可能バージョン) を使用する場合:

    a. 接続画面の [Local] (ローカル) タブを選択します
    b. 以下の情報を指定します。

        - 接続名
        - ポート
    c. [接続] ボタンを選択します。

> ⓘ Note
>
> DynamoDB Local に接続するには、ターミナルを使用して DynamoDB Local を手動で起動するか (「Deploying DynamoDB local on your computer」を参照)、NoSQL Workbench ナビゲーションメニューの DDB Local トグルを使用して DynamoDB Local を直接起動します。接続ポートが DynamoDB Local ポートと同じであることを確認してください。

5. 作成した接続で、[Open (開く)] を選択します。

DynamoDB データベースに接続すると、使用可能なテーブルのリストが左ペインに表示されます。テーブルのいずれかを選択すると、テーブルに保存されているデータのサンプルが返されます。

選択したテーブルに対してクエリを実行できるようになりました。

テーブルに対してクエリを実行するには、オペレーションの構築に関する次のセクション「複雑なオペレーションの構築」を参照してください。

複雑なオペレーションの構築

Amazon DynamoDB 用 NoSQL Workbench のオペレーションビルダーは、複雑なデータプレーンオペレーションが実行できる視覚的なインターフェイスを提供します。プロジェクション式と条件式のサポートが含まれています。オペレーションを作成したら、後で使用するために保存できます (最大 50 個のオペレーションを保存できます)。その後、[Saved Operations] (保存されたオペレーション) メニューで、頻繁に使用するデータプレーンオペレーションのリストを参照し、それらを使用して自動的に新しいオペレーションを設定および構築できます。これらのオペレーションのサンプルコードを複数の言語で生成することもできます。

NoSQL Workbench では、DynamoDB ステートメント用の PartiQL の構築をサポートしています。これにより、SQL 互換のクエリ言語を使用して DynamoDB とやり取りできます。NoSQL Workbench では、DynamoDB CRUD API オペレーションの構築もサポートしています。

NoSQL Workbench を使用してオペレーションを構築するには、左側のナビゲーションペインで [Operation builder] (オペレーションビルダー) アイコンを選択します。

トピック

- PartiQL ステートメントの構築

- API オペレーションの構築

PartiQL ステートメントの構築

NoSQL Workbench を使用して PartiQL for DynamoDB ステートメントを構築するには、NoSQL Workbench UI の上部にある [PartiQL エディタ] を選択します。

オペレーションビルダーでは、次の PartiQL ステートメントタイプを作成できます。

トピック

- シングルトンステートメント
- トランザクション
- バッチ

シングルトンステートメント

PartiQL ステートメントのコードを実行または生成するには、次の操作を行います。

1. ウィンドウの上部にある [PartiQL エディタ] を選択します。
2. 有効な PartiQL ステートメントを入力します。
3. ステートメントでパラメータを使用する場合
   a. [オプションのリクエストパラメータ] を選択します。
   b. [パラメータの追加] を選択します。
   c. 属性のタイプと値を入力します。
   d. パラメータを追加する場合は、ステップ b および c を繰り返します。
4. コードを生成する場合は、[Generate code (コードの生成)] を選択します。

   表示されたタブから目的の言語を選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。
5. オペレーションをすぐに実行する場合は、[実行] をクリックします。
6. このオペレーションを後で使用するために保存する場合は、[Save operation] (保存オペレーション) を選択します。次に、オペレーションの名前を入力して [Save] (保存) を選択します。

トランザクション

PartiQL トランザクションのコードを実行または生成するには、次の操作を行います。

1. [その他のオペレーション] ドロップダウンで、[PartiQLTransaction] を選択します。
2. [Add a new statement] (新しいステートメントの追加) を選択します。
3. 有効な PartiQL ステートメントを入力します。

> **Note**
>
> 読み込みおよび書き込みオペレーションは、同じ PartiQL トランザクションのリクエストではサポートされていません。INSERT、UPDATE、および DELETE ステートメントでは、同じリクエストに SELECT ステートメントを指定することはできません。詳細については、「DynamoDB 用 PartiQL を使用したトランザクションの実行」を参照してください。

4. ステートメントでパラメータを使用する場合
   a. [Optional request parameters] (オプションのリクエストパラメータ) を選択します。
   b. [パラメータの追加] を選択します。
   c. 属性のタイプと値を入力します。
   d. パラメータを追加する場合は、ステップ b および c を繰り返します。
5. さらにステートメントを追加する場合は、ステップ 2～4 を繰り返します。
6. コードを生成する場合は、[コードの生成] を選択します。

   表示されたタブから目的の言語を選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。
7. オペレーションをすぐに実行する場合は、[実行] をクリックします。
8. このオペレーションを後で使用するために保存する場合は、[Save operation] (保存オペレーション) を選択します。次に、オペレーションの名前を入力して [Save] (保存) を選択します。

バッチ

PartiQL バッチのコードを実行または生成するには、次の操作を行います。

1. [その他のオペレーション] ドロップダウンで、[PartiQLBatch] を選択します。
2. [Add a new statement] (新しいステートメントの追加) を選択します。

3. 有効な PartiQL ステートメントを入力します。

> **Note**
>
> 読み込みおよび書き込みオペレーションは、同じ PartiQL バッチリクエストではサポートされていません。つまり、SELECT ステートメントを INSERT、UPDATE、DELETE ステートメントと同じリクエストに含めることはできません。同じ項目への書き込みオペレーションは許可されていません。BatchGetItem オペレーションと同様に、シングルトン読み込みオペレーションのみがサポートされています。スキャンおよびクエリオペレーションはサポートされていません。詳細については、「DynamoDB 用 PartiQL を使用した batch オペレーションの実行」を参照してください。

4. ステートメントでパラメータを使用する場合
    a. [オプションのリクエストパラメータ] を選択します。
    b. [パラメータの追加] を選択します。
    c. 属性のタイプと値を入力します。
    d. パラメータを追加する場合は、ステップ b および c を繰り返します。
5. さらにステートメントを追加する場合は、ステップ 2～4 を繰り返します。
6. コードを生成する場合は、[コードの生成] を選択します。

    表示されたタブから目的の言語を選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。
7. オペレーションをすぐに実行する場合は、[実行] をクリックします。
8. このオペレーションを後で使用するために保存する場合は、[Save operation] (保存オペレーション) を選択します。次に、オペレーションの名前を入力して [Save] (保存) を選択します。

API オペレーションの構築

NoSQL Workbench を使用して DynamoDB CRUD API を構築するには、NoSQL Workbench ユーザーインターフェイスの左側で [オペレーションビルダー] を選択します。

次に [開く] を選択して、接続を選択します。

オペレーションビルダーでは、以下の操作を実行できます。

- Delete Table

- Create Table
- Update Table

- Put Item
- Update Item
- Delete Item
- Query
- Scan
- Transact Get Items
- Transact Write Items

テーブルを削除する

`Delete Table` オペレーションを実行するには、次の操作を行います。

1. [テーブル] セクションで、削除するテーブルを見つけます。
2. 横方向の省略記号メニューで、[テーブルを削除] を選択します。
3. [テーブル名] を入力して、テーブルを削除することを確認します。
4. [削除] を選択します。

このオペレーションの詳細については、「Amazon DynamoDB API リファレンス」の「Delete table」(テーブルを削除する) を参照してください。

GSI を削除

`Delete GSI` オペレーションを実行するには、次の操作を行います。

1. [テーブル] セクションで、削除するテーブルの GSI を見つけます。
2. 横方向の省略記号メニューで、[GSI を削除] を選択します。
3. [GSI 名] を入力して、GSI を削除することを確認します。
4. [削除] を選択します。

このオペレーションの詳細については、「Amazon DynamoDB API リファレンス」の「Delete table」(テーブルを削除する) を参照してください。

テーブルの作成

`Create Table` オペレーションを実行するには、次の操作を行います。

1. [テーブル] セクションの横にある [+] アイコンを選択します。
2. 目的のテーブル名を入力します。
3. パーティションキーを作成します。
4. オプション: ソートキーを作成します。
5. キャパシティー設定をカスタマイズするには、[デフォルトのキャパシティー設定を使用] チェックボックスをオフにします。
   - これで、プロビジョンドキャパシティまたはオンデマンドキャパシティを選択できます。

     [プロビジョンド] を選択すると、最小と最大の読み込みおよび書き込みキャパシティーユニットを設定できます。自動スケーリングを有効または無効にすることもできます。
   - テーブルの現在の設定が [オンデマンド] になっている場合は、プロビジョンドスループットを指定できません。
   - [オンデマンド] から [プロビジョンドスループット] に切り替えると、すべての GSI に自動スケーリング (最小: 1、最大: 10、ターゲット: 70%) が自動的に適用されます。
6. このテーブルを GSI なしで作成するには、[GSI をスキップして作成] を選択します。オプションで、[次へ] を選択して、この新しいテーブルで GSI を作成できます。

このオペレーションの詳細については、「Amazon DynamoDB API リファレンス」の「Create table」(テーブルを作成する) を参照してください。

GSI の作成

`Create GSI` オペレーションを実行するには、次の操作を行います。

1. GSI を追加する先のテーブルを見つけます。
2. 横方向の省略記号メニューで、[GSI の作成] を選択します。
3. [インデックス名] で、GSI に名前を付けます。
4. パーティションキーを作成します。
5. オプション: ソートキーを作成します。
6. ドロップダウンから [射影タイプ] オプションを選択します。
7. [GSI の作成] を選択します。

このオペレーションの詳細については、「Amazon DynamoDB API リファレンス」の「Create table」(テーブルを作成する) を参照してください。

テーブルを更新する

`Update Table` オペレーションでテーブルのキャパシティー設定を更新するには、次の操作を行います。

1. キャパシティー設定を更新するテーブルを見つけます。
2. 横方向の省略記号メニューで、[キャパシティー設定を更新] を選択します。
3. [プロビジョンド] または [オンデマンドキャパシティー] を選択します。

   [プロビジョンド] を選択すると、最小と最大の読み込みおよび書き込みキャパシティーユニットを設定できます。自動スケーリングを有効または無効にすることもできます。
4. [Update] (更新) を選択します。

このオペレーションの詳細については、「Amazon DynamoDB API リファレンス」の「Update table」(テーブルの更新) を参照してください。

GSI の更新

`Update Table` オペレーションで GSI のキャパシティー設定を更新するには、次の操作を行います。

> Note
>
> デフォルトでは、グローバルセカンダリインデックスは基本テーブルのキャパシティー設定を継承します。基本テーブルがプロビジョンドキャパシティーモードである場合にのみ、グローバルセカンダリインデックスを別のキャパシティーモードに設定できます。プロビジョニングモードのテーブルでグローバルセカンダリインデックスを作成するときには、そのインデックスに対して予想されるワークロードに応じた読み込み/書き込みキャパシティーユニットを指定する必要があります。詳細については、「グローバルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項」を参照してください。

1. キャパシティー設定を更新する GSI を見つけます。
2. 横方向の省略記号メニューで、[キャパシティー設定を更新] を選択します。
3. これで、プロビジョンドキャパシティまたはオンデマンドキャパシティを選択できます。

[プロビジョンド] を選択すると、最小と最大の読み込みおよび書き込みキャパシティーユニットを設定できます。自動スケーリングを有効または無効にすることもできます。

4. [Update] (更新) を選択します。

このオペレーションの詳細については、「Amazon DynamoDB API リファレンス」の「Update table」(テーブルの更新) を参照してください。

項目を配置する

Put Item オペレーションを使用して項目を作成します。Put Item オペレーションのコードを実行または生成するには、次の操作を行います。

1. 項目を作成するテーブルを見つけます。
2. [アクション] ドロップダウンで、[項目の作成] を選択します。
3. パーティションキー値を入力します。
4. ソートキーの値を入力します (存在する場合)。
5. キーではない属性を指定する場合は、次の操作を行います。

   a. [+ その他の属性を追加] を選択します。
   b. [属性の名前]、[タイプ]、[値] を指定します。
6. Put Item オペレーションを成功させるために条件式を満たす必要がある場合は、次の手順を実行します。

   a. [Condition (条件)] を選択します。
   b. 属性名、比較演算子、属性タイプ、および属性値を指定します。
   c. 他の条件が必要な場合は、[Condition (条件)] を再度選択します。

   詳細については、「条件式」を参照してください。
7. コードを生成する場合は、[コードの生成] を選択します。

   表示されたタブから目的の言語を選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。
8. オペレーションをすぐに実行する場合は、[実行] をクリックします。
9. このオペレーションを後で使用するために保存する場合は、[Save operation] (オペレーションの保存) を選択し、オペレーションの名前を入力して、[Save] (保存) を選択します。

このオペレーションの詳細については、Amazon DynamoDB API リファレンスの 「PutItem」を参照してください。

項目を更新する

`Update Item` オペレーションのコードを実行または生成するには、次の操作を行います。

1. 項目を更新するテーブルを見つけます。
2. 項目を選択します。
3. 選択した式の属性名と属性値を入力します。
4. さらに式を追加する場合は、[式を更新] ドロップダウンリストで別の式を選択し、[+] アイコンを選択します。
5. `Update Item` オペレーションを成功させるために条件式を満たす必要がある場合は、次の手順を実行します。

   a. [Condition (条件)] を選択します。
   b. 属性名、比較演算子、属性タイプ、および属性値を指定します。
   c. 他の条件が必要な場合は、[Condition (条件)] を再度選択します。

   詳細については、「条件式」を参照してください。
6. コードを生成する場合は、[Generate code (コードの生成)] を選択します。

   使用する言語のタブを選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。
7. オペレーションをすぐに実行する場合は、[実行] をクリックします。
8. このオペレーションを後で使用するために保存する場合は、[Save operation] (オペレーションの保存) を選択し、オペレーションの名前を入力して、[Save] (保存) を選択します。

このオペレーションの詳細については、Amazon DynamoDB API リファレンスの 「UpdateItem」を参照してください。

項目を削除する

`Delete Item` オペレーションを実行するには、次の操作を行います。

1. 項目を削除するテーブルを見つけます。
2. 項目を選択します。

3. [アクション] ドロップダウンで、[項目の削除] を選択します。
4. [削除] を選択して、項目を削除することを確認します。

このオペレーションの詳細については、Amazon DynamoDB API リファレンスの 「DeleteItem」を参照してください。

項目の複製

同じ属性を持つ新しい項目を作成することで、項目を複製できます。項目を複製するには、次の操作を行います。

1. 項目を複製するテーブルを見つけます。
2. 項目を選択します。
3. [アクション] ドロップダウンで、[項目の複製] を選択します。
4. 新しいパーティションキーを指定します。
5. 新しいソートキーを指定します (必要な場合)。
6. [実行] を選択します。

このオペレーションの詳細については、Amazon DynamoDB API リファレンスの 「DeleteItem」を参照してください。

Query

`Query` オペレーションのコードを実行または生成するには、次の操作を行います。

1. NoSQL Workbench UI の上部で、[クエリ] を選択します。
2. パーティションキーの値を指定します。
3. `Query` オペレーションにソートキーが必要な場合は、以下の操作を実行します。
    a. [ソートキー] を選択します。
    b. 比較演算子と属性値を指定します。
4. [クエリ] を選択して、このクエリオペレーションを実行します。さらにオプションが必要な場合は、[その他のオプション] チェックボックスをオンにして、次の手順に進みます。
5. オペレーション結果とともにすべての属性を返す必要がない場合は、[Projection expression (プロジェクション式)] を選択します。

6. [+] アイコンを選択します。
7. クエリ結果とともに返される属性を入力します。
8. さらに属性が必要な場合は、[+] を選択します。
9. `Query` オペレーションを成功させるために条件式を満たす必要がある場合は、次の手順を実行します。

    a. [Condition (条件)] を選択します。
    b. 属性名、比較演算子、属性タイプ、および属性値を指定します。
    c. 他の条件が必要な場合は、[Condition (条件)] を再度選択します。

    詳細については、「条件式」を参照してください。
10. コードを生成する場合は、[Generate code (コードの生成)] を選択します。

    使用する言語のタブを選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。
11. オペレーションをすぐに実行する場合は、[実行] をクリックします。
12. このオペレーションを後で使用するために保存する場合は、[Save operation] (オペレーションの保存) を選択し、オペレーションの名前を入力して、[Save] (保存) を選択します。

このオペレーションの詳細については、Amazon DynamoDB API リファレンスの「Query」を参照してください。

Scan

`Scan` オペレーションのコードを実行または生成するには、次の操作を行います。

1. NoSQL Workbench UI の上部で、[スキャン] を選択します。
2. [スキャン] ボタンを選択して、この基本的なスキャンオペレーションを実行します。さらにオプションが必要な場合は、[その他のオプション] チェックボックスをオンにして、次の手順に進みます。
3. 属性名を指定して、スキャン結果をフィルタリングします。
4. オペレーション結果とともにすべての属性を返す必要がない場合は、[Projection expression (プロジェクション式)] を選択します。
5. スキャンオペレーションを成功させるために条件式を満たす必要がある場合は、次の手順を実行します。

   a. [Condition (条件)] を選択します。
   b. 属性名、比較演算子、属性タイプ、および属性値を指定します。
   c. 他の条件が必要な場合は、[Condition (条件)] を再度選択します。

   詳細については、「条件式」を参照してください。

6. コードを生成する場合は、[Generate code (コードの生成)] を選択します。

   使用する言語のタブを選択します。これで、このコードをコピーしてアプリケーションで使用できるようになります。
7. オペレーションをすぐに実行する場合は、[実行] をクリックします。
8. このオペレーションを後で使用するために保存する場合は、[Save operation] (オペレーションの保存) を選択し、オペレーションの名前を入力して、[Save] (保存) を選択します。

TransactGetItems

`TransactGetItems` オペレーションのコードを実行または生成するには、次の操作を行います。

1. NoSQL Workbench UI の上部にある [その他のオペレーション] ドロップダウンで、[TransactGetItems] を選択します。
2. [TransactGetItem] の近くにある [+] アイコンを選択します。
3. パーティションキーを指定します。
4. ソートキーを指定します (必要な場合)。
5. [実行] を選択してオペレーションを実行するか、[オペレーションを保存] を選択して保存するか、[コードを生成] を選択してオペレーションのコードを生成します。

トランザクションの詳細については、「Amazon DynamoDB Transactions」を参照してください。

TransactWriteItems

`TransactWriteItems` オペレーションのコードを実行または生成するには、次の操作を行います。

1. NoSQL Workbench UI の上部にある [その他のオペレーション] ドロップダウンで、[TransactWriteItems] を選択します。
2. [アクション] ドロップダウンで、オペレーションを選択します。

3. [TransactWriteItem] の近くにある [+] アイコンを選択します。
4. [アクション] ドロップダウンで、実行するオペレーションを選択します。

    - `DeleteItem` の場合、項目を削除する オペレーションの手順に従います。
    - `PutItem` の場合、項目を配置する オペレーションの手順に従います。
    - `UpdateItem` の場合、項目を更新する オペレーションの手順に従います。

    アクションの順序を変更するには、左側のリストでアクションを選択し、上矢印または下矢印を選択してリスト内で上下に移動します。

    アクションを削除するには、リストでアクションを選択してから、[削除] (ごみ箱) アイコンを選択します。
5. [実行] を選択してオペレーションを実行するか、[オペレーションを保存] を選択して保存するか、[コードを生成] を選択してオペレーションのコードを生成します。

トランザクションの詳細については、「Amazon DynamoDB Transactions」を参照してください。

NoSQL Workbench を使用したテーブルのクローン作成

テーブルのクローンを作成すると、テーブルのキースキーマ (オプションで GSI スキーマと項目) が開発環境間でコピーされます。DynamoDB Local と Amazon DynamoDB アカウント間でテーブルのクローンを作成できます。また、あるアカウントのテーブルのクローンを別のリージョンのアカウントに作成して、迅速に実験を行うこともできます。

テーブルのクローンを作成するには

1. オペレーションビルダーで、接続とリージョンを選択します (DynamoDB Local ではリージョンの選択はできません)。
2. DynamoDB に接続したら、テーブルを参照し、クローンを作成するテーブルを選択します。
3. 横方向の省略記号メニューで、[クローン作成] オプションを選択します。
4. クローンの保存先の詳細を入力します。

    a. 接続を選択します。

    b. リージョンを選択します (リージョンは DynamoDB Local では使用できません)。

    c. 新しいテーブル名を入力します。

    d. クローン作成のオプションを選択します。

i. [キースキーマ] はデフォルトで選択されており、選択を解除することはできません。デフォルトでは、テーブルのクローンを作成するとプライマリキーとソートキー (使用可能な場合) がコピーされます。

ii. クローン対象のテーブルに GSI がある場合は、デフォルトで [GSI スキーマ] が選択されます。テーブルのクローンを作成すると、GSI プライマリキーとソートキー (使用可能な場合) がコピーされます。[GSI スキーマ] の選択を解除して、GSI スキーマのクローン作成は省略することもできます。テーブルのクローンを作成すると、ベーステーブルの容量設定が GSI の容量設定としてコピーされます。オペレーションビルダーで `UpdateTable` オペレーションを使用して、クローン作成の完了後にテーブルの GSI 容量設定を更新できます。

5. クローンを作成する項目の数を入力します。キースキーマとオプションで GSI スキーマのみをクローン作成する場合は、[クローンする項目] の値を 0 のままにしておくことができます。クローンを作成できる項目の最大数は 5000 です。

6. キャパシティモードを選択します。

   a. デフォルトでは、[オンデマンドモード]が選択されています。DynamoDB オンデマンドには、読み取りおよび書き込みリクエストのリクエストごとの支払い料金が用意されているため、使用した分だけ課金されます。詳細については、「DynamoDB On-demand mode」を参照してください。

   b. [プロビジョンドモード]では、アプリケーションに必要な 1 秒あたりの読み込みと書き込みの回数を指定します。Auto Scaling を使用すると、トラフィックの変更に応じて、テーブルのプロビジョンドキャパシティーを自動的に調整できます。詳細については、「DynamoDB Provisioned mode」を参照してください。

7. [クローン] を選択してクローン作成を開始します。

8. クローン作成のプロセスはバックグラウンドで実行されます。テーブルのクローン作成のステータスが変化すると、[Operation builder] タブに通知が表示されます。このステータスを表示するには、[Operation builder] タブを選択し、矢印ボタンを選択します。矢印ボタンは、メニューサイドバーの下部にあるテーブルのクローン作成ステータスのウィジェットにあります。

CSV ファイルへのデータのエクスポート

クエリの結果を Operation Builder から CSV ファイルにエクスポートできます。これにより、スプレッドシートにデータをロードしたり、好みのプログラミング言語でデータ処理ができるようになります。

CSV へエクスポート

1. Operation Builder で、スキャンやクエリなどの任意の操作を実行します。

> Note
>
> - 読み取り API オペレーションと PartiQL ステートメントの結果のみを CSV ファイルにエクスポートできます。トランザクション読み取りステートメントの結果はエクスポートできません。
> - 現時点では、結果を 1 ページずつ CSV ファイルにエクスポートできます。結果のページが複数ある場合は、各ページを個別にエクスポートする必要があります。

2. 結果からエクスポートする項目を選択します。
3. [アクション] ドロップダウンで、[CSV としてエクスポート] を選択します。
4. CSV ファイルのファイル名と場所を選択し、[Save] (保存) を選択します。

NoSQL Workbench のサンプルデータモデル

モデラーとビジュアライザーのホームページには、NoSQL Workbench に付属のサンプルモデルが多数表示されます。このセクションでは、これらのモデルとその可能性のある用途について説明します。

トピック

- 従業員データモデル
- ディスカッションフォーラムデータモデル
- ミュージックライブラリデータモデル
- スキーリゾートデータモデル
- クレジットカードオファーデータモデル
- ブックマークデータモデル

従業員データモデル

このデータモデルは、入門モデルです。これは、一意のエイリアス、名、姓、職名、マネージャー、およびスキルなどの従業員の基本的な詳細を表します。

このデータモデルは、複数のスキルを持つなどの複雑な属性を扱うなど、いくつかの技術を示しています。このモデルは、セカンダリインデックス DirectReports によって達成されたマネージャーとその部下の従業員による 1 対多リレーションシップの例でもあります。

このデータモデルによって容易になるアクセスパターンは次のとおりです。

- `Employee` というテーブルによって容易になる、従業員のログインエイリアスを使用した従業員レコードの取得。
- `Name` という Employee テーブルのグローバルセカンダリインデックスによって容易になる、名前による従業員の検索。
- `DirectReports` という従業員のテーブルのグローバルセカンダリインデックスによって容易になる、マネージャーのログインエイリアスを使用したマネージャーの直接レポートの取得。

ディスカッションフォーラムデータモデル

このデータモデルは、ディスカッションフォーラムを表します。このモデルを使用すると、お客様は開発者コミュニティとやり取りし、質問をしたり、他のお客さま顧客の投稿に応答することができます。各 AWS サービスには専用フォーラムがあります。フォーラムにメッセージを投稿することで、誰でも新しいディスカッションスレッドを開始できます。各スレッドは任意の数の返信を受け取ります。

このデータモデルによって容易になるアクセスパターンは次のとおりです。

- `Forum` というテーブルによって容易になる、フォーラムの名前を使用したフォーラムのレコードの取得。
- `Thread` というテーブルによって容易になる、フォーラムの特定のスレッドまたはすべてのスレッドの取得。
- `PostedBy-Message-Index` という Reply テーブルのグローバルセカンダリインデックスによって容易になる、投稿ユーザーの E メールアドレスを使用した返信の検索。

ミュージックライブラリデータモデル

このデータモデルは、大量の楽曲コレクションを持ち、ダウンロード数の多い楽曲をほぼリアルタイムで紹介するミュージックライブラリです。

このデータモデルによって容易になるアクセスパターンは次のとおりです。

- `Songs` というテーブルによって容易になる曲レコードの取得。

- `Songs` というテーブルによって容易になる、曲の特定のダウンロードレコードまたはすべてのダウンロードレコードの取得。
- `Song` というテーブルによって容易になる、曲の特定の月別ダウンロード数レコードまたはすべての月のダウンロード数レコードの取得
- `Songs` というテーブルによって容易になる曲のすべてのレコード (曲のレコード、ダウンロードレコード、月別ダウンロード数レコードを含む) の取得。
- `DownloadsByMonth` という Songs テーブルのグローバルセカンダリインデックスによって容易になる、ほとんどのダウンロード曲の検索。

スキーリゾートデータモデル

このデータモデルは、毎日収集される各スキーリフトについて大量のデータを収集するスキーリゾートを表します。

このデータモデルによって容易になるアクセスパターンは次のとおりです。

- `SkiLifts` というテーブルによって容易になる、指定されたスキーリフトやリゾート全体の動的および静的なすべてのデータの取得。
- `SkiLifts` というテーブルによって容易になる、特定の日付のスキーリフトまたはリゾート全体のすべての動的データ (一意のリフトライダー、積雪、雪崩の危険、リフト状況を含む) の取得。
- `SkiLifts` というテーブルによって容易になる、特定のスキーリフトのすべての静的データ (リフトが経験豊富なライダーのみを対象とする場合、リフトが上昇する垂直フィート、リフトのライディング時間を含む) の取得。
- `SkiLiftsByRiders` という SkiLifts テーブルのグローバルセカンダリインデックスによって容易になる、一意のライダーの総数によってソートされる特定のスキーリフトまたはリゾート全体について記録されたデータの日付の取得。

クレジットカードオファーデータモデル

このデータモデルは、クレジットカードオファーアプリケーションで使用されます。

クレジットカードプロバイダは、時間の経過とともにオファーを生成します。これらのオファーには、手数料なしの残高振替、与信限度額の増加、金利の低減、キャッシュバック、航空会社マイルが含まれます。お客様がこれらのオファーを承認または拒否すると、それに応じてそれぞれのオファーステータスが更新されます。

このデータモデルによって容易になるアクセスパターンは次のとおりです。

- メインテーブルによって容易になる、AccountId を使用した取引先レコードの取得。
- セカンダリインデックス AccountIndex によって容易になる、いくつかの予測項目を持つすべてのアカウントの取得。
- メインテーブルによって容易になる、AccountId を使用したアカウントおよびそれらのアカウントに関連付けられているすべてのオファーレコードの取得。
- メインテーブルによって容易になる、AccountId および OfferId を使用した取引先およびそれらの取引先に関連付けられた特定のオファーレコードの取得。
- セカンダリインデックス ACCEPTED/DECLINED によって容易になる、OfferType、AccountId、および OfferType を使用した取引先に関連付けられた特定の Status のすべての GSI1 オファーレコードの取得
- メインテーブルによって容易になる、OfferId を使用したオファーおよび関連するオファー項目レコードの取得。

ブックマークデータモデル

このデータモデルは、お客様のストアブックマークに使用されます。

お客様は多くのブックマークを持つことができ、ブックマークは多くのお客様に属することができます。このデータモデルは、多対多の関係を表します。

このデータモデルによって容易になるアクセスパターンは次のとおりです。

- customerId による単一のクエリで、ブックマークだけでなくお客様データも返せるようになりました。
- クエリ ByEmail インデックスは、E メールアドレスごとにお客様データを返します。ブックマークは、このインデックスによって取得されないことに注意してください。
- クエリ ByUrl インデックスは、URL ごとにブックマークデータを取得します。同じ URL を複数のお客様からブックマークすることができるため、インデックスのソートキーとして customerId があることに注意してください。
- クエリ ByCustomerFolder インデックスは、各お客様のフォルダごとにブックマークを取得します。

NoSQL Workbench のリリース履歴

NoSQL Workbench クライアントツールの各リリースにおける重要な変更点を次の表に示します。

バージョン	変更	説明	日付
3.13.0	NoSQL Workbench オペレーションビルダーの改善点	NoSQL Workbench には、ダークモードのネイティブサポートが含まれるようになりました。オペレーションビルダーのテーブルと項目のオペレーションを改善しました。項目の結果とオペレーションビルダーのリクエスト情報は JSON 形式で利用できます。	2024 年 4 月 24 日
3.12.0	NoSQL Workbench によるテーブルのクローン作成と消費されたキャパシティの返却	DynamoDB Local と DynamoDB ウェブサービスアカウント間、または DynamoDB ウェブサービスアカウント間でテーブルをクローンして、開発の反復を迅速に行えるようになりました。Operation builder を使用してオペレーションを実行した後に消費された RCU または WCU を表示します。CSV ファイルからのインポート時にデータが上書きされる問題を修正しました。	2024 年 2 月 26 日

バージョン	変更	説明	日付
3.11.0	DynamoDB Local の機能強化	ビルトイン DynamoDB Local インスタンスを起動するときにポートを指定できるようになりました。NoSQL Workbench を管理者権限なしで Windows にインストールできるようになりました。データモデルテンプレートを更新しました。	2024 年 1 月 17 日
3.10.0	Apple シリコンのネイティブサポート	NoSQL Workbench で、Apple シリコンを搭載した Mac をネイティブにサポートするようになりました。Number 型の属性にサンプルデータ生成形式を設定できるようになりました。	2023 年 12 月 5 日
3.9.0	データモデラーの改善	Visualizer では、既存のテーブルを上書きするオプションを使用して、データモデルを DynamoDB local にコミットできるようになりました。	2023 年 11 月 3 日

バージョン	変更	説明	日付
3.8.0	サンプルデータ生成	NoSQL Workbench は DynamoDB データ型のサンプルデータの生成をサポートするようになりました。	2023 年 9 月 25 日
3.6.0	オペレーションビルダーの改善	オペレーションビルダーの接続管理が改善されました。データモデラーの属性名は、データを削除せずに変更できるようになりました。その他のバグ修正。	2023 年 4 月 11 日
3.5.0	新しい AWS リージョン向けのサポート	NoSQL Workbench は、ap-south-2、ap-southeast-3、ap-southeast-4、eu-central-2、eu-south-2、me-central-1、および me-west-1 リージョンのサポートを開始しました。	2023 年 2 月 23 日
3.4.0	DynamoDB local のサポート	NoSQL Workbench は、インストールプロセスの一部として、DynamoDB local のインストールをサポートするようになりました。	2022 年 12 月 6 日

バージョン	変更	説明	日付
3.3.0	コントロールプレーンオペレーションのサポート	Operation Builder でコントロールプレーンオペレーションがサポートされるようになりました。	2022 年 6 月 1 日
3.2.0	CSV のインポートとエクスポート	ビジュアライザーツールの CSV ファイルからサンプルデータをインポートできるようになりました。また、Operation Builder クエリの結果を CSV ファイルにエクスポートすることもできます。	2021 年 10 月 11 日
3.1.0	オペレーションの保存	NoSQL Workbench のオペレーションビルダーで、後で使用するためにオペレーションを保存することがサポートされるようになりました。	2021 年 7 月 12 日

バージョン	変更	説明	日付
3.0.0	容量設定と CloudFormation のインポート/エクスポート	Amazon DynamoDB 用の NoSQL Workbench は、テーブルの読み込み/書き込み容量モードの指定をサポートするようになり、AWS CloudFormation 形式でデータモデルをインポートおよびエクスポートできるようになりました。	2021 年 4 月 21 日
2.2.0	PartiQL のサポート	Amazon DynamoDB 用の NoSQL Workbench が、DynamoDB 用 PartiQL ステートメントを構築するためのサポートを追加します。	2020 年 12 月 4 日
1.1.0	Linux のサポート。	Amazon DynamoDB 用の NoSQL Workbench は Linux —Ubuntu、Fedora および Debian でサポートされています。	2020 年 5 月 4 日
1.0.0	NoSQL Workbench for Amazon DynamoDB – GA。	Amazon DynamoDB 用 NoSQL Workbench の一般提供が開始されました。	2020 年 3 月 2 日

バージョン	変更	説明	日付
0.4.1	IAM ロールと一時的なセキュリティ認証情報のサポート。	Amazon DynamoDB 用の NoSQL Workbench が、AWS Identity and Access Management (IAM) ロールと一時的なセキュリティ認証情報のサポートを追加。	2019 年 12 月 19 日
0.3.1	DynamoDB local (ダウンロード可能バージョン) のサポート。	NoSQL Workbench では、DynamoDB local (ダウンロード可能バージョン) に接続して、DynamoDB テーブルを設計、作成、クエリ、管理できるようになりました。	2019年11月8日
0.2.1	NoSQL Workbench プレビュー版がリリースされました。	これは DynamoDB 用 NoSQL Workbench の初版リリースです。NoSQL Workbench を使用して、DynamoDB テーブルを設計、作成、クエリ、および管理します。	2019 年 9 月 16 日

AWS SDK を使用した DynamoDB のコード例

以下は、AWS ソフトウェア開発キット (SDK) で DynamoDB を使用する方法を説明するコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

クロスサービスの例は、複数の AWS のサービス で動作するサンプルアプリケーションです。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello DynamoDB

次のコード例は、DynamoDB の使用を開始する方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace DynamoDB_Actions;

public static class HelloDynamoDB
```

```
{
    static async Task Main(string[] args)
    {
        var dynamoDbClient = new AmazonDynamoDBClient();

        Console.WriteLine($"Hello Amazon Dynamo DB! Following are some of your
 tables:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five tables.
        var response = await dynamoDbClient.ListTablesAsync(
            new ListTablesRequest()
            {
                Limit = 5
            });

        foreach (var table in response.TableNames)
        {
            Console.WriteLine($"\tTable: {table}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListTables」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

CMakeLists.txt CMake ファイルのコード。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_dynamodb.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
```

```
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListTables」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListTables」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListTables」を参照してください。

コードの例

- AWS SDK を使用した DynamoDB 向けアクション
  - AWS SDK または CLI で BatchExecuteStatement を使用する
  - AWS SDK または CLI で BatchGetItem を使用する
  - AWS SDK または CLI で BatchWriteItem を使用する
  - AWS SDK または CLI で CreateTable を使用する
  - AWS SDK または CLI で DeleteItem を使用する
  - AWS SDK または CLI で DeleteTable を使用する
  - AWS SDK または CLI で DescribeTable を使用する
  - AWS SDK または CLI で ExecuteStatement を使用する
  - AWS SDK または CLI で GetItem を使用する
  - AWS SDK または CLI で ListTables を使用する
  - AWS SDK または CLI で PutItem を使用する
  - AWS SDK または CLI で Query を使用する
  - AWS SDK または CLI で Scan を使用する
  - AWS SDK または CLI で UpdateItem を使用する
  - AWS SDK または CLI で UpdateTable を使用する
- AWS SDK を使用した DynamoDB のシナリオ
  - AWS SDK を使用して DAX で DynamoDB の読み取りを高速化する
  - AWS SDK で DynamoDB テーブル、項目、クエリの使用を開始する
  - PartiQL ステートメントのバッチと AWS SDK を使用して DynamoDB テーブルにクエリを実行する
  - PartiQL と AWS SDK を使用して DynamoDB テーブルに対してクエリを実行する
  - AWS SDK を使用して DynamoDB のドキュメントモデルを使用する
  - AWS SDK を使用して DynamoDB 用の高レベルのオブジェクト永続性モデルを使用する

- AWS SDK を使用した DynamoDB 用のサーバーレスサンプル

  - DynamoDB トリガーから Lambda 関数を呼び出す
  - DynamoDB トリガーで Lambda 関数のバッチアイテムの失敗をレポートする
- AWS SDK を使用した DynamoDB のクロスサービスの例
  - DynamoDB テーブルにデータを送信するアプリケーションを構築する
  - COVID-19 データを追跡する API Gateway REST API を作成する
  - ステップ関数でメッセンジャーアプリケーションを作成する
  - ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成
  - DynamoDB データを追跡するウェブアプリケーションを作成する
  - API Gateway で WebSocket チャットアプリケーションを作成する
  - AWS SDK を使用して、Amazon Rekognition で画像内の PPE を検出する
  - ブラウザからの Lambda 関数の呼び出し
  - AWS SDK を使用した Amazon DynamoDB のパフォーマンスのモニタリング
  - AWS SDK を使用して、EXIF とその他の画像情報を保存する
  - API Gateway を使用して Lambda 関数を呼び出す
  - Step Functions を使用して Lambda 関数を呼び出す
  - スケジュールされたイベントを使用した Lambda 関数の呼び出し

AWS SDK を使用した DynamoDB 向けアクション

以下は、AWS SDK を使用して個々の DynamoDB アクションを実行する方法を説明するコード例です。これらは DynamoDB API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、Amazon DynamoDB API リファレンスを参照してください。

例

- AWS SDK または CLI で BatchExecuteStatement を使用する
- AWS SDK または CLI で BatchGetItem を使用する
- AWS SDK または CLI で BatchWriteItem を使用する
- AWS SDK または CLI で CreateTable を使用する
- AWS SDK または CLI で DeleteItem を使用する

- AWS SDK または CLI で DeleteTable を使用する
- AWS SDK または CLI で DescribeTable を使用する
- AWS SDK または CLI で ExecuteStatement を使用する
- AWS SDK または CLI で GetItem を使用する
- AWS SDK または CLI で ListTables を使用する
- AWS SDK または CLI で PutItem を使用する
- AWS SDK または CLI で Query を使用する
- AWS SDK または CLI で Scan を使用する
- AWS SDK または CLI で UpdateItem を使用する
- AWS SDK または CLI で UpdateTable を使用する

AWS SDK または CLI で `BatchExecuteStatement` を使用する

以下のコード例は、`BatchExecuteStatement` の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- PartiQL ステートメントのバッチを使用してテーブルにクエリを実行する

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

INSERT ステートメントのバッチを使用して項目を追加します。

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
```

```
       /// information will be inserted.</param>
       /// <param name="movieFileName">The name of the JSON file that contains
       /// movie information.</param>
       /// <returns>A Boolean value that indicates the success or failure of
       /// the insert operation.</returns>
       public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
       {
           // Get the list of movies from the JSON file.
           var movies = ImportMovies(movieFileName);

           var success = false;

           if (movies is not null)
           {
               // Insert the movies in a batch using PartiQL. Because the
               // batch can contain a maximum of 25 items, insert 25 movies
               // at a time.
               string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
               var statements = new List<BatchStatementRequest>();

               try
               {
                   for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                   {
                       for (var i = indexOffset; i < indexOffset + 25; i++)
                       {
                           statements.Add(new BatchStatementRequest
                           {
                               Statement = insertBatch,
                               Parameters = new List<AttributeValue>
                               {
                                   new AttributeValue { S = movies[i].Title },
                                   new AttributeValue { N =
movies[i].Year.ToString() },
                               },
                           });
                       }

                       var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                       {
                           Statements = statements,
                       });

                       // Wait between batches for movies to be successfully
added.
                       System.Threading.Thread.Sleep(3000);

                       success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                       // Clear the list of statements for the next batch.
                       statements.Clear();
                   }
               }
               catch (AmazonDynamoDBException ex)
               {
                   Console.WriteLine(ex.Message);
               }
           }

           return success;
       }

       /// <summary>
       /// Loads the contents of a JSON file into a list of movies to be
       /// added to the DynamoDB table.
       /// </summary>
       /// <param name="movieFileName">The full path to the JSON file.</param>
       /// <returns>A generic list of movie objects.</returns>
       public static List<Movie> ImportMovies(string movieFileName)
       {
           if (!File.Exists(movieFileName))
           {
               return null!;
           }

           using var sr = new StreamReader(movieFileName);
           string json = sr.ReadToEnd();
           var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

           if (allMovies is not null)
           {
               // Return the first 250 entries.
               return allMovies.GetRange(0, 250);
           }
           else
           {
               return null!;
           }
       }
```

SELECT ステートメントのバッチを使用して項目を取得します。

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year
 = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

UPDATE ステートメントのバッチを使用して項目を更新します。

```
        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

DELETE ステートメントのバッチを使用して項目を削除します。

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「BatchExecuteStatement」を参照してください。

C++

SDK for C++

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

INSERT ステートメントのバッチを使用して項目を追加します。

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
 = Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
 Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add the movies: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

SELECT ステートメントのバッチを使用して項目を取得します。

```
   // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);
           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);

       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (outcome.IsSuccess()) {
           const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

           for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
               const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

               printMovieInfo(item);
           }
       }
       else {
           std::cerr << "Failed to retrieve the movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
```

```
       }
   }
```

UPDATE ステートメントのバッチを使用して項目を更新します。

```
   // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

   for (size_t i = 0; i < titles.size(); ++i) {
       ratings[i] = askQuestionForFloatRange(
               Aws::String("\nLet's update your the movie, \"") + titles[i] +
               ".\nYou rated it  " + std::to_string(ratings[i])
               + ", what new rating would you give it? ", 1, 10);
   }

   std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());

       std::stringstream sqlStream;
       sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);

           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }
```

```
        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

DELETE ステートメントのバッチを使用して項目を削除します。

```
    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「BatchExecuteStatement」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

INSERT ステートメントのバッチを使用して項目を追加します。

```

// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(movies []Movie) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
   "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}
```

SELECT ステートメントのバッチを使用して項目を取得します。

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(movies []Movie) ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}
```

UPDATE ステートメントのバッチを使用して項目を更新します。

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(movies []Movie, ratings []float64)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }
 _, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

DELETE ステートメントのバッチを使用して項目を削除します。

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(movies []Movie) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

この例で使用している Movie struct を定義します。

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「BatchExecuteStatement」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

PartiQL を使用して項目のバッチを作成します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL を使用して項目のバッチを取得します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL を使用して項目のバッチを更新します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL を使用して項目のバッチを削除します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「BatchExecuteStatement」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「BatchExecuteStatement」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
 with
                           each statement. This list must be in the same order as
 the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
 table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
 %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「BatchExecuteStatement」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

PartiQL を使用して項目のバッチを読み取ります。

```
class DynamoDBPartiQLBatch
```

```
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Selects a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_select(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "SELECT * FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({statements: request_items})
  end
```

PartiQL を使用して項目のバッチを削除します。

```
class DynamoDBPartiQLBatch

  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_write(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({statements: request_items})
  end
```

- API の詳細については、AWS SDK for Ruby API リファレンスの「BatchExecuteStatement」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **`BatchGetItem`** を使用する

以下のコード例は、`BatchGetItem` の使用方法を示しています。

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";
```

```
       public static async void 
RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient client)
       {
           var request = new BatchGetItemRequest
           {
               RequestItems = new Dictionary<string, KeysAndAttributes>()
           {
               { _table1Name,
                 new KeysAndAttributes
                 {
                     Keys = new List<Dictionary<string, AttributeValue> >()
                     {
                         new Dictionary<string, AttributeValue>()
                         {
                             { "Name", new AttributeValue {
                           S = "Amazon DynamoDB"
                       } }
                         },
                         new Dictionary<string, AttributeValue>()
                         {
                             { "Name", new AttributeValue {
                           S = "Amazon S3"
                       } }
                         }
                     }
                 }},
               {
                   _table2Name,
                   new KeysAndAttributes
                   {
                       Keys = new List<Dictionary<string, AttributeValue> >()
                       {
                           new Dictionary<string, AttributeValue>()
                           {
                               { "ForumName", new AttributeValue {
                                     S = "Amazon DynamoDB"
                                 } },
                               { "Subject", new AttributeValue {
                                     S = "DynamoDB Thread 1"
                                 } }
                           },
                           new Dictionary<string, AttributeValue>()
                           {
                               { "ForumName", new AttributeValue {
                                     S = "Amazon DynamoDB"
                                 } },
                               { "Subject", new AttributeValue {
                                     S = "DynamoDB Thread 2"
                                 } }
                           },
                           new Dictionary<string, AttributeValue>()
                           {
                               { "ForumName", new AttributeValue {
                                     S = "Amazon S3"
                                 } },
                               { "Subject", new AttributeValue {
                                     S = "S3 Thread 1"
                                 } }
                           }
                       }
                   }
               }
           }
           };

           BatchGetItemResponse response;
           do
           {
               Console.WriteLine("Making request");
               response = await client.BatchGetItemAsync(request);

               // Check the response.
               var responses = response.Responses; // Attribute list in the
response.

               foreach (var tableResponse in responses)
               {
                   var tableResults = tableResponse.Value;
                   Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                   foreach (var item1 in tableResults)
                   {
                       PrintItem(item1);
                   }
               }
```

```
                // Any unprocessed keys? could happen if you exceed 
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys = 
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> 
attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",", 
value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",", 
value.NS.ToArray()) + "]")
                    );
            }

Console.WriteLine("************************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();
            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「BatchGetItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
#############################################################################
# function dynamodb_batch_get_item
#
# This function gets a batch of items from a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the keys of the items to get.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
##########################################################################
function dynamodb_batch_get_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_batch_get_item"
    echo "Get a batch of items from a DynamoDB table."
    echo " -i item  -- Path to json file containing the keys of the items to
 get."
    echo ""
  }

  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  response=$(aws dynamodb batch-get-item \
    --request-items file://"$item")
  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-get-item operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「BatchGetItem」を参照してください。

C++

SDK for C++

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Batch get items from different Amazon DynamoDB tables.
/*!
  \sa batchGetItem()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::batchGetItem(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::BatchGetItemRequest request;

    // Table1: Forum.
    Aws::String table1Name = "Forum";
    Aws::DynamoDB::Model::KeysAndAttributes table1KeysAndAttributes;

    // Table1: Projection expression.
    table1KeysAndAttributes.SetProjectionExpression("#n, Category, Messages,
 #v");
```

```
   // Table1: Expression attribute names.
   Aws::Http::HeaderValueCollection headerValueCollection;
   headerValueCollection.emplace("#n", "Name");
   headerValueCollection.emplace("#v", "Views");
   table1KeysAndAttributes.SetExpressionAttributeNames(headerValueCollection);

   // Table1: Set key name, type, and value to search.
   std::vector<Aws::String> nameValues = {"Amazon DynamoDB", "Amazon S3"};
   for (const Aws::String &name: nameValues) {
       Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
       Aws::DynamoDB::Model::AttributeValue key;
       key.SetS(name);
       keys.emplace("Name", key);
       table1KeysAndAttributes.AddKeys(keys);
   }

   Aws::Map<Aws::String, Aws::DynamoDB::Model::KeysAndAttributes> requestItems;
   requestItems.emplace(table1Name, table1KeysAndAttributes);

   // Table2: ProductCatalog.
   Aws::String table2Name = "ProductCatalog";
   Aws::DynamoDB::Model::KeysAndAttributes table2KeysAndAttributes;
   table2KeysAndAttributes.SetProjectionExpression("Title, Price, Color");

   // Table2: Set key name, type, and value to search.
   std::vector<Aws::String> idValues = {"102", "103", "201"};
   for (const Aws::String &id: idValues) {
       Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
       Aws::DynamoDB::Model::AttributeValue key;
       key.SetN(id);
       keys.emplace("Id", key);
       table2KeysAndAttributes.AddKeys(keys);
   }

   requestItems.emplace(table2Name, table2KeysAndAttributes);

   bool result = true;
   do {  // Use a do loop to handle pagination.
       request.SetRequestItems(requestItems);
       const Aws::DynamoDB::Model::BatchGetItemOutcome &outcome =
dynamoClient.BatchGetItem(
               request);

       if (outcome.IsSuccess()) {
           for (const auto &responsesMapEntry:
outcome.GetResult().GetResponses()) {
               Aws::String tableName = responsesMapEntry.first;
               const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &tableResults = responsesMapEntry.second;
               std::cout << "Retrieved " << tableResults.size()
                         << " responses for table '" << tableName << "'.\n"
                         << std::endl;
               if (tableName == "Forum") {

                   std::cout << "Name | Category | Message | Views" <<
std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Name").GetS() << " | ";
                       std::cout << item.at("Category").GetS() << " | ";
                       std::cout << (item.count("Message") == 0 ? "" : item.at(
                               "Messages").GetN()) << " | ";
                       std::cout << (item.count("Views") == 0 ? "" : item.at(
                               "Views").GetN()) << std::endl;
                   }
               }
               else {
                   std::cout << "Title | Price | Color" << std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Title").GetS() << " | ";
                       std::cout << (item.count("Price") == 0 ? "" : item.at(
                               "Price").GetN());
                       if (item.count("Color")) {
                           std::cout << " | ";
                           for (const
std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> &listItem: item.at(
                                   "Color").GetL())
                               std::cout << listItem->GetS() << " ";
                       }
                       std::cout << std::endl;
                   }
               }
               std::cout << std::endl;
           }

           // If necessary, repeat request for remaining items.
           requestItems = outcome.GetResult().GetUnprocessedKeys();
```

```
        }
        else {
            std::cerr << "Batch get item failed: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
            break;
        }
    } while (!requestItems.empty());

    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「BatchGetItem」を参照してください。

CLI

AWS CLI

テーブルから複数の項目を取得するには

次の `batch-get-items` の例では、3 つの `GetItem` リクエストのバッチを使用して `MusicCollection` テーブルから複数の項目を読み込み、この操作で使用された読み込み容量ユニットの数を取得します。このコマンドは `AlbumTitle` 属性のみを返します。

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

`request-items.json` の内容:

```
{
    "MusicCollection": {
        "Keys": [
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Call Me Today"}
            },
            {
                "Artist": {"S": "Acme Band"},
                "SongTitle": {"S": "Happy Day"}
            },
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Scared of My Shadow"}
            }
        ],
        "ProjectionExpression":"AlbumTitle"
    }
}
```

出力:

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「バッチオペレーション」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「BatchGetItem」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

サービスクライアントを使用してバッチアイテムを取得する方法を示します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchReadItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItems(dynamoDbClient, tableName);
    }

    public static void getBatchItems(DynamoDbClient dynamoDbClient, String
 tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Make the batchGetItem request.
        BatchGetItemResponse batchGetItemResponse =
 dynamoDbClient.batchGetItem(batchGetItemRequest);

        // Extract and print the retrieved items.
        Map<String, List<Map<String, AttributeValue>>> responses =
 batchGetItemResponse.responses();
        if (responses.containsKey(tableName)) {
            List<Map<String, AttributeValue>> musicItems =
 responses.get(tableName);
            for (Map<String, AttributeValue> item : musicItems) {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            }
        } else {
            System.out.println("No items retrieved.");
        }
    }
}
```

サービスクライアントおよびページネーターを使用してバッチアイテムを取得する方法を示します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BatchGetItemsPaginator {

    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();
        getBatchItemsPaginator(dynamoDbClient, tableName) ;
    }

    public static void getBatchItemsPaginator(DynamoDbClient dynamoDbClient,
 String tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Use batchGetItemPaginator for paginated requests.
        dynamoDbClient.batchGetItemPaginator(batchGetItemRequest).stream()
            .flatMap(response -> response.responses().getOrDefault(tableName,
 Collections.emptyList()).stream())
            .forEach(item -> {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            });
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「BatchGetItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「BatchGet」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses["Books"]);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「BatchGetItem」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「BatchGetItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: DynamoDB テーブル「Music」および「Songs」から「Somewhere Down The Road」という SongTitle の項目を取得します。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

出力:

```
Name                           Value
----                           -----
```

```
Artist                          No One You Know
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
CriticRating                    10
Genre                           Country
Price                           1.94
Artist                          No One You Know
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
CriticRating                    10
Genre                           Country
Price                           1.94
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「BatchGetItem」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import decimal
import json
import logging
import os
import pprint
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
dynamodb = boto3.resource("dynamodb")

MAX_GET_SIZE = 100  # Amazon DynamoDB rejects a get batch larger than 100 items.
```

```
def do_batch_get(batch_keys):
    """
    Gets a batch of items from Amazon DynamoDB. Batches can contain keys from
    more than one table.

    When Amazon DynamoDB cannot process all items in a batch, a set of
 unprocessed
    keys is returned. This function uses an exponential backoff algorithm to
 retry
    getting the unprocessed keys until all are retrieved or the specified
    number of tries is reached.

    :param batch_keys: The set of keys to retrieve. A batch can contain at most
 100
                       keys. Otherwise, Amazon DynamoDB returns an error.
    :return: The dictionary of retrieved items grouped under their respective
             table names.
    """
    tries = 0
    max_tries = 5
    sleepy_time = 1  # Start with 1 second of sleep, then exponentially increase.
    retrieved = {key: [] for key in batch_keys}
    while tries < max_tries:
        response = dynamodb.batch_get_item(RequestItems=batch_keys)
        # Collect any retrieved items and retry unprocessed keys.
        for key in response.get("Responses", []):
            retrieved[key] += response["Responses"][key]
        unprocessed = response["UnprocessedKeys"]
        if len(unprocessed) > 0:
            batch_keys = unprocessed
            unprocessed_count = sum(
                [len(batch_key["Keys"]) for batch_key in batch_keys.values()]
            )
            logger.info(
                "%s unprocessed keys returned. Sleep, then retry.",
 unprocessed_count
            )
            tries += 1
            if tries < max_tries:
                logger.info("Sleeping for %s seconds.", sleepy_time)
                time.sleep(sleepy_time)
                sleepy_time = min(sleepy_time * 2, 32)
        else:
            break
```

```
return retrieved
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「BatchGetItem」を参照してください。

Swift

SDK for Swift

Note

これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// Gets an array of `Movie` objects describing all the movies in the
/// specified list. Any movies that aren't found in the list have no
/// corresponding entry in the resulting array.
///
/// - Parameters
///     - keys: An array of tuples, each of which specifies the title and
///       release year of a movie to fetch from the table.
///
/// - Returns:
///     - An array of `Movie` objects describing each match found in the
///     table.
///
/// - Throws:
///     - `MovieError.ClientUninitialized` if the DynamoDB client has not
///     been initialized.
```

```
    ///     - DynamoDB errors are thrown without change.
    func batchGet(keys: [(title: String, year: Int)]) async throws -> [Movie] {
        guard let client = self.ddbClient else {
            throw MovieError.ClientUninitialized
        }

        var movieList: [Movie] = []
        var keyItems: [[Swift.String:DynamoDBClientTypes.AttributeValue]] = []

        // Convert the list of keys into the form used by DynamoDB.

        for key in keys {
            let item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
                "title": .s(key.title),
                "year": .n(String(key.year))
            ]
            keyItems.append(item)
        }

        // Create the input record for `batchGetItem()`. The list of requested
        // items is in the `requestItems` property. This array contains one
        // entry for each table from which items are to be fetched. In this
        // example, there's only one table containing the movie data.
        //
        // If we wanted this program to also support searching for matches
        // in a table of book data, we could add a second `requestItem`
        // mapping the name of the book table to the list of items we want to
        // find in it.
        let input = BatchGetItemInput(
            requestItems: [
                self.tableName: .init(
                    consistentRead: true,
                    keys: keyItems
                )
            ]
        )

        // Fetch the matching movies from the table.

        let output = try await client.batchGetItem(input: input)

        // Get the set of responses. If there aren't any, return the empty
        // movie list.
        guard let responses = output.responses else {
            return movieList
        }

        // Get the list of matching items for the table with the name
        // `tableName`.

        guard let responseList = responses[self.tableName] else {
            return movieList
        }

        // Create `Movie` items for each of the matching movies in the table
        // and add them to the `MovieList` array.

        for response in responseList {
            movieList.append(try Movie(withItem: response))
        }

        return movieList
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「BatchGetItem」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **BatchWriteItem** を使用する

以下のコード例は、BatchWriteItem の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

項目のバッチをムービーテーブルに書き込みます。

```
        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonSerializer.Deserialize<List<Movie>>(
                json,
                new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });

            // Now return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }

        /// <summary>
        /// Writes 250 items to the movie table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="movieFileName">A string containing the full path to
        /// the JSON file containing movie data.</param>
        /// <returns>A long integer value representing the number of movies
        /// imported from the JSON file.</returns>
        public static async Task<long> BatchWriteItemsAsync(
            AmazonDynamoDBClient client,
            string movieFileName)
        {
            var movies = ImportMovies(movieFileName);
            if (movies is null)
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContext(client);

            var movieBatch = context.CreateBatchWrite<Movie>();
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「BatchWriteItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
##############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

この例で使用されているユーティリティ関数。

```
################################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「BatchWriteItem」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Batch write items from a JSON file.
/*!
  \sa batchWriteItem()
  \param jsonFilePath: JSON file path.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

/*
 * The input for this routine is a JSON file that you can download from the
 following URL:
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
SampleData.html.
 *
 * The JSON data uses the BatchWriteItem API request syntax. The JSON strings are
 * converted to AttributeValue objects. These AttributeValue objects will then
 generate
 * JSON strings when constructing the BatchWriteItem request, essentially
 outputting
 * their input.
 *
 * This is perhaps an artificial example, but it demonstrates the APIs.
 */

bool AwsDoc::DynamoDB::batchWriteItem(const Aws::String &jsonFilePath,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    std::ifstream fileStream(jsonFilePath);
```

```
   if (!fileStream) {
       std::cerr << "Error: could not open file '" << jsonFilePath << "'."
                 << std::endl;
   }

   std::stringstream stringStream;
   stringStream << fileStream.rdbuf();
   Aws::Utils::Json::JsonValue jsonValue(stringStream);

   Aws::DynamoDB::Model::BatchWriteItemRequest batchWriteItemRequest;
   Aws::Map<Aws::String, Aws::Utils::Json::JsonView> level1Map =
jsonValue.View().GetAllObjects();
   for (const auto &level1Entry: level1Map) {
       const Aws::Utils::Json::JsonView &entriesView = level1Entry.second;
       const Aws::String &tableName = level1Entry.first;
       // The JSON entries at this level are as follows:
       //  key - table name
       //  value - list of request objects
       if (!entriesView.IsListType()) {
           std::cerr << "Error: JSON file entry '"
                     << tableName << "' is not a list." << std::endl;
           continue;
       }

       Aws::Utils::Array<Aws::Utils::Json::JsonView> entries =
entriesView.AsArray();

       Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;
       if (AwsDoc::DynamoDB::addWriteRequests(tableName, entries,
                                              writeRequests)) {
           batchWriteItemRequest.AddRequestItems(tableName, writeRequests);
       }
   }

   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   Aws::DynamoDB::Model::BatchWriteItemOutcome outcome =
dynamoClient.BatchWriteItem(
           batchWriteItemRequest);

   if (outcome.IsSuccess()) {
       std::cout << "DynamoDB::BatchWriteItem was successful." << std::endl;
   }
```

```
    else {
        std::cerr << "Error with DynamoDB::BatchWriteItem. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return true;
}

//! Convert requests in JSON format to a vector of WriteRequest objects.
/*!
  \sa addWriteRequests()
  \param tableName: Name of the table for the write operations.
  \param requestsJson: Request data in JSON format.
  \param writeRequests: Vector to receive the WriteRequest objects.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::addWriteRequests(const Aws::String &tableName,
                                        const
 Aws::Utils::Array<Aws::Utils::Json::JsonView> &requestsJson,

 Aws::Vector<Aws::DynamoDB::Model::WriteRequest> &writeRequests) {
    for (size_t i = 0; i < requestsJson.GetLength(); ++i) {
        const Aws::Utils::Json::JsonView &requestsEntry = requestsJson[i];
        if (!requestsEntry.IsObject()) {
            std::cerr << "Error: incorrect requestsEntry type "
                      << requestsEntry.WriteReadable() << std::endl;
            return false;
        }

        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> requestsMap =
 requestsEntry.GetAllObjects();

        for (const auto &request: requestsMap) {
            const Aws::String &requestType = request.first;
            const Aws::Utils::Json::JsonView &requestJsonView = request.second;

            if (requestType == "PutRequest") {
                if (!requestJsonView.ValueExists("Item")) {
                    std::cerr << "Error: item key missing for requests "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 attributes;
                if (!getAttributeObjectsMap(requestJsonView.GetObject("Item"),
                                            attributes)) {
                    std::cerr << "Error getting attributes "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }

                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(attributes);
                writeRequests.push_back(
                        Aws::DynamoDB::Model::WriteRequest().WithPutRequest(
                                putRequest));
            }
            else {
                std::cerr << "Error: unimplemented request type '" << requestType
                          << "'." << std::endl;
            }
        }
    }

    return true;
}

//! Generate a map of AttributeValue objects from JSON records.
/*!
  \sa getAttributeObjectsMap()
  \param jsonView: JSONView of attribute records.
  \param writeRequests: Map to receive the AttributeValue objects.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::getAttributeObjectsMap(const Aws::Utils::Json::JsonView
 &jsonView,
                                         Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &attributes) {
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> objectsMap =
 jsonView.GetAllObjects();
    for (const auto &entry: objectsMap) {
        const Aws::String &attributeKey = entry.first;
        const Aws::Utils::Json::JsonView &attributeJsonView = entry.second;

        if (!attributeJsonView.IsObject()) {
            std::cerr << "Error: attribute not an object "
                      << attributeJsonView.WriteReadable() << std::endl;
            return false;
        }

        attributes.emplace(attributeKey,

 Aws::DynamoDB::Model::AttributeValue(attributeJsonView));
    }

    return true;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「BatchWriteItem」を参照してください。

CLI

AWS CLI

テーブルに複数の項目を追加するには

次の `batch-write-item` の例は、3 つの `PutItem` リクエストのバッチを使用して、`MusicCollection` テーブルに 3 つの新しい項目を追加します。また、このオペレーションによって消費される書き込み容量ユニットの数、およびオペレーションによって変更されるアイテムコレクションに関する情報も要求します。

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

`request-items.json` の内容:

```
{
    "MusicCollection": [
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Call Me Today"},
                    "AlbumTitle": {"S": "Somewhat Famous"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "Acme Band"},
                    "SongTitle": {"S": "Happy Day"},
                    "AlbumTitle": {"S": "Songs About Life"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Scared of My Shadow"},
                    "AlbumTitle": {"S": "Blue Sky Blues"}
                }
            }
        }
    ]
}
```

出力:

```
{
    "UnprocessedItems": {},
    "ItemCollectionMetrics": {
        "MusicCollection": [
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "No One You Know"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            },
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "Acme Band"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            }
        ]
    },
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 6.0,
            "Table": {
                "CapacityUnits": 3.0
            },
            "LocalSecondaryIndexes": {
                "AlbumTitleIndex": {
                    "CapacityUnits": 3.0
                }
            }
        }
    ]
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「バッチオペレーション」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「BatchWriteItem」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(movies []Movie, maxMovies int) (int,
 error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(context.TODO(),
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「BatchWriteItem」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

サービスクライアントを使用して、多数の項目をテーブルに挿入します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutRequest;
import software.amazon.awssdk.services.dynamodb.model.WriteRequest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchWriteItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        addBatchItems(dynamoDbClient, tableName);
    }

    public static void addBatchItems(DynamoDbClient dynamoDbClient, String
 tableName) {
        // Specify the updates you want to perform.
        List<WriteRequest> writeRequests = new ArrayList<>();

        // Set item 1.
        Map<String, AttributeValue> item1Attributes = new HashMap<>();
        item1Attributes.put("Artist",
 AttributeValue.builder().s("Artist1").build());
        item1Attributes.put("Rating", AttributeValue.builder().s("5").build());
        item1Attributes.put("Comments", AttributeValue.builder().s("Great
 song!").build());
```

```
        item1Attributes.put("SongTitle",
 AttributeValue.builder().s("SongTitle1").build());

 writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item1Attri

        // Set item 2.
        Map<String, AttributeValue> item2Attributes = new HashMap<>();
        item2Attributes.put("Artist",
 AttributeValue.builder().s("Artist2").build());
        item2Attributes.put("Rating", AttributeValue.builder().s("4").build());
        item2Attributes.put("Comments", AttributeValue.builder().s("Nice
 melody.").build());
        item2Attributes.put("SongTitle",
 AttributeValue.builder().s("SongTitle2").build());

 writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item2Attri

        try {
            // Create the BatchWriteItemRequest.
            BatchWriteItemRequest batchWriteItemRequest =
 BatchWriteItemRequest.builder()
                .requestItems(Map.of(tableName, writeRequests))
                .build();

            // Execute the BatchWriteItem operation.
            BatchWriteItemResponse batchWriteItemResponse =
 dynamoDbClient.batchWriteItem(batchWriteItemRequest);

            // Process the response.
            System.out.println("Batch write successful: " +
 batchWriteItemResponse);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

拡張クライアントを使用して、多数の項目をテーブルに挿入します。

```
import com.example.dynamodb.Customer;
import com.example.dynamodb.Music;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import
 software.amazon.awssdk.enhanced.dynamodb.model.BatchWriteItemEnhancedRequest;
import software.amazon.awssdk.enhanced.dynamodb.model.WriteBatch;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneOffset;

/*
 * Before running this code example, create an Amazon DynamoDB table named
 Customer with these columns:
 *   - id - the id of the record that is the key
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnhancedBatchWriteItems {
        public static void main(String[] args) {
                Region region = Region.US_EAST_1;
                DynamoDbClient ddb = DynamoDbClient.builder()
                                .region(region)
                                .build();
                DynamoDbEnhancedClient enhancedClient =
 DynamoDbEnhancedClient.builder()
                                .dynamoDbClient(ddb)
                                .build();
                putBatchRecords(enhancedClient);
                ddb.close();
```

```
       }

       public static void putBatchRecords(DynamoDbEnhancedClient enhancedClient)
{
               try {
                       DynamoDbTable<Customer> customerMappedTable =
enhancedClient.table("Customer",
                                       TableSchema.fromBean(Customer.class));
                       DynamoDbTable<Music> musicMappedTable =
enhancedClient.table("Music",
                                       TableSchema.fromBean(Music.class));
                       LocalDate localDate = LocalDate.parse("2020-04-07");
                       LocalDateTime localDateTime = localDate.atStartOfDay();
                       Instant instant =
localDateTime.toInstant(ZoneOffset.UTC);

                       Customer record2 = new Customer();
                       record2.setCustName("Fred Pink");
                       record2.setId("id110");
                       record2.setEmail("fredp@noserver.com");
                       record2.setRegistrationDate(instant);

                       Customer record3 = new Customer();
                       record3.setCustName("Susan Pink");
                       record3.setId("id120");
                       record3.setEmail("spink@noserver.com");
                       record3.setRegistrationDate(instant);

                       Customer record4 = new Customer();
                       record4.setCustName("Jerry orange");
                       record4.setId("id101");
                       record4.setEmail("jorange@noserver.com");
                       record4.setRegistrationDate(instant);

                       BatchWriteItemEnhancedRequest
batchWriteItemEnhancedRequest = BatchWriteItemEnhancedRequest
                                       .builder()
                                       .writeBatches(

WriteBatch.builder(Customer.class) // add items to the Customer

         // table

.mappedTableResource(customerMappedTable)
```

```
 .addPutItem(builder -> builder.item(record2))

 .addPutItem(builder -> builder.item(record3))

 .addPutItem(builder -> builder.item(record4))
                                                                        .build(),

 WriteBatch.builder(Music.class) // delete an item from the Music

       // table

 .mappedTableResource(musicMappedTable)

 .addDeleteItem(builder -> builder.key(

       Key.builder().partitionValue(

                       "Famous Band")

                       .build()))
                                                                        .build())
                                        .build();

                        // Add three items to the Customer table and delete one
 item from the Music
                        // table.

 enhancedClient.batchWriteItem(batchWriteItemEnhancedRequest);
                        System.out.println("done");

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
        }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「BatchWriteItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「BatchWrite」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year'
 is recommended
        // to account for duplicate titles.
        ["BatchWriteMoviesTable"]: putRequests,
      },
    });

    await docClient.send(command);
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「BatchWriteItem」を参照してください。

SDK for JavaScript (v2)

ⓘ Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「BatchWriteItem」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    public function writeBatch(string $TableName, array $Batch, int $depth = 2)
    {
        if (--$depth <= 0) {
            throw new Exception("Max depth exceeded. Please try with fewer batch
 items or increase depth.");
        }

        $marshal = new Marshaler();
        $total = 0;
        foreach (array_chunk($Batch, 25) as $Items) {
            foreach ($Items as $Item) {
                $BatchWrite['RequestItems'][$TableName][] = ['PutRequest' =>
 ['Item' => $marshal->marshalItem($Item)]];
            }
            try {
                echo "Batching another " . count($Items) . " for a total of " .
 ($total += count($Items)) . " items!\n";
                $response = $this->dynamoDbClient->batchWriteItem($BatchWrite);
                $BatchWrite = [];
            } catch (Exception $e) {
                echo "uh oh...";
                echo $e->getMessage();
                die();
            }
            if ($total >= 250) {
                echo "250 movies is probably enough. Right? We can stop there.
\n";
                break;
            }
        }
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「BatchWriteItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: 新しい項目を作成する、または既存の項目を「Music 」DynamoDB テーブルおよび「Songs」DynamoDB テーブルの新しい項目で置き換えます。

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

出力:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「BatchWriteItem」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
 sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
 least
                       the keys required by the schema that was specified when
 the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「BatchWriteItem」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Fills an Amazon DynamoDB table with the specified data. Items are sent in
  # batches of 25 until all items are written.
  #
  # @param movies [Enumerable] The data to put in the table. Each item must
 contain at least
```

```
  #                              the keys required by the schema that was specified
 when the
  #                              table was created.
  def write_batch(movies)
    index = 0
    slice_size = 25
    while index < movies.length
      movie_items = []
      movies[index, slice_size].each do |movie|
        movie_items.append({put_request: { item: movie }})
      end
      @dynamo_resource.client.batch_write_item({request_items: { @table.name =>
 movie_items }})
      index += slice_size
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts(
      "Couldn't load data into table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「BatchWriteItem」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        // Create a Swift `URL` and use it to load the file into a `Data`
        // object. Then decode the JSON into an array of `Movie` objects.

        let fileUrl = URL(fileURLWithPath: jsonPath)
        let jsonData = try Data(contentsOf: fileUrl)

        var movieList = try JSONDecoder().decode([Movie].self, from: jsonData)

        // Truncate the list to the first 200 entries or so for this example.

        if movieList.count > 200 {
            movieList = Array(movieList[...199])
        }

        // Before sending records to the database, break the movie list into
        // 25-entry chunks, which is the maximum size of a batch item request.

        let count = movieList.count
        let chunks = stride(from: 0, to: count, by: 25).map {
            Array(movieList[$0 ..< Swift.min($0 + 25, count)])
        }

        // For each chunk, create a list of write request records and populate
        // them with `PutRequest` requests, each specifying one movie from the
        // chunk. Once the chunk's items are all in the `PutRequest` list,
        // send them to Amazon DynamoDB using the
        // `DynamoDBClient.batchWriteItem()` function.

        for chunk in chunks {
            var requestList: [DynamoDBClientTypes.WriteRequest] = []

            for movie in chunk {
                let item = try await movie.getAsItem()
                let request = DynamoDBClientTypes.WriteRequest(
                    putRequest: .init(
                        item: item
                    )
                )
                requestList.append(request)
            }

            let input = BatchWriteItemInput(requestItems: [tableName: 
 requestList])
            _ = try await client.batchWriteItem(input: input)
        }
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「BatchWriteItem」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **CreateTable** を使用する

以下のコード例は、CreateTable の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DAX で読み取りを高速化
- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Creates a new Amazon DynamoDB table and then waits for the new
        /// table to become active.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.CreateTableAsync(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5,
                },
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await
 client.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreateTable」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#       -p provisioned_throughput -- Provisioned throughput settings for the
 table.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema provisioned_throughput
 response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
```

```
   echo " -k key_schema -- JSON file path of a list of attributes and their key
types."
   echo " -p provisioned_throughput -- Provisioned throughput settings for the
table."
   echo ""
 }

 # Retrieve the calling parameters.
 while getopts "n:a:k:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     a) attribute_definitions="${OPTARG}" ;;
     k) key_schema="${OPTARG}" ;;
     p) provisioned_throughput="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_definitions" ]]; then
   errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
   usage
   return 1
 fi

 if [[ -z "$key_schema" ]]; then
   errecho "ERROR: You must provide a key schema json file path the -k
parameter."
   usage
```

```
    return 1
  fi

  if [[ -z "$provisioned_throughput" ]]; then
    errecho "ERROR: You must provide a provisioned throughput json file path the
 -p parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho "    provisioned_throughput:   $provisioned_throughput"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --key-schema file://"$key_schema" \
    --provisioned-throughput "$provisioned_throughput")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
```

```
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
```

```
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「CreateTable」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;
    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「CreateTable」を参照してください。

CLI

AWS CLI

例 1: タグ付きのテーブルを作成するには

次の create-table の例では、指定された属性とキースキーマを使用して、MusicCollection という名前のテーブルを作成します。このテーブルはプロビジョニングされたスループットを使用し、保存時にはデフォルトの AWS 所有の CMK を使用して暗号化されます。またこのコマンドは、Owner キーと blueTeam 値を使用して、テーブルにタグを適用します。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 2: オンデマンドモードでテーブルを作成するには

次の例は、プロビジョニングされたスループットモードではなく、オンデマンドモードを使用して MusicCollection というテーブルを作成します。これは、ワークロードが予測できないテーブルに役立ちます。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 3: テーブルを作成してカスタマーマネージド CMK で暗号化するには

次の例では、MusicCollection という名前のテーブルを作成し、カスタマーマネージド CMK を使用して暗号化します。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 4: ローカルセカンダリインデックスを持つテーブルを作成するには

次の例では、指定された属性とキースキーマを使用して、AlbumTitleIndex という名前のローカルセカンダリインデックスを持つ MusicCollection という名前のテーブルを作成します。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S AttributeName=AlbumTitle,AttributeType=S
 \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 5: グローバルセカンダリインデックスを持つテーブルを作成するには

次の例では、GameTitleIndex という名前のグローバルセカンダリインデックスを持つ GameScores という名前のテーブルを作成します。ベーステーブルには、パーティションキー UserId とソートキー GameTitle があり、特定のゲームの個々のユーザーのベストスコアを効率的に見つけることができます。一方、GSI にはパーティションキー GameTitle とソートキー TopScore があり、特定のゲームの全体的な最高スコアをすばやく見つけることができます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S AttributeName=TopScore,AttributeType=N \
    --key-schema AttributeName=UserId,KeyType=HASH \
                AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 6: 複数のグローバルセカンダリインデックスを持つテーブルを一度に作成するには

次の例では、2 つのグローバルセカンダリインデックスを持つ `GameScores` という名前のテーブルを作成します。GSI スキーマはコマンドラインではなくファイルを介して渡されます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S AttributeName=TopScore,AttributeType=N
 AttributeName=Date,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

`gsi.json` の内容:

```
[
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 7: ストリームが有効なテーブルを作成するには

次の例では、DynamoDB ストリームを有効にした状態の GameScores という名前のテーブルを作成します。各アイテムの新しいイメージと古いイメージの両方がストリームに書き込まれます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB テーブルの基本的なオペレーション」を参照してください。

例 8: Keys-Only ストリームが有効なテーブルを作成するには

次の例では、DynamoDB ストリームを有効にした状態の `GameScores` という名前のテーブルを作成します。変更された項目のキー属性のみがストリームに書き込まれます。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB Streams の変更データキャプチャ」を参照してください。

例 9: Standard Infrequent Access クラスでテーブルを作成するには

次の例は、`GameScores` という名前のテーブルを作成し、Standard-Infrequent Access (DynamoDB Standard-IA) テーブルクラスを割り当てます。このテーブルクラスは、ストレージが主なコストとなるように最適化されています。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「テーブルクラス」を参照してください。

例 10: 削除保護を有効にしたテーブルを作成するには

次の例では、GameScores というテーブルを作成し、削除保護を有効にします。

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S
 AttributeName=GameTitle,AttributeType=S \
    --key-schema AttributeName=UserId,KeyType=HASH
 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「削除保護の使用」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「CreateTable」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable() (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(context.TODO(),
 &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName: aws.String(basics.TableName),
  ProvisionedThroughput: &types.ProvisionedThroughput{
   ReadCapacityUnits:  aws.Int64(10),
   WriteCapacityUnits: aws.Int64(10),
  },
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(context.TODO(), &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
 }
 return tableDesc, err
}
```

- API の詳細については、『AWS SDK for Go API リファレンス』の「CreateTable」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key>

                Where:
                    tableName - The Amazon DynamoDB table to create (for example,
 Music3).
                    key - The key for the Amazon DynamoDB table (for example,
 Artist).
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
```

```
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        String result = createTable(ddb, tableName, key);
        System.out.println("New table is " + result);
        ddb.close();
    }

    public static String createTable(DynamoDbClient ddb, String tableName, String
key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        CreateTableRequest request = CreateTableRequest.builder()
                .attributeDefinitions(AttributeDefinition.builder()
                        .attributeName(key)
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(KeySchemaElement.builder()
                        .attributeName(key)
                        .keyType(KeyType.HASH)
                        .build())
                .provisionedThroughput(ProvisionedThroughput.builder()
                        .readCapacityUnits(10L)
                        .writeCapacityUnits(10L)
                        .build())
                .tableName(tableName)
                .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[CreateTable](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateTable」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateTable」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun createNewTable(tableNameVal: String, key: String): String? {
    val attDef = AttributeDefinition {
        attributeName = key
        attributeType = ScalarAttributeType.S
    }

    val keySchemaVal = KeySchemaElement {
        attributeName = key
        keyType = KeyType.Hash
    }

    val provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef)
        keySchema = listOf(keySchemaVal)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->

        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists { // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「CreateTable」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

テーブルを作成します。

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
 'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
 'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
 'WriteCapacityUnits' => 10],
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「CreateTable」を参照してください。

PowerShell

Tools for PowerShell

例 1: この例では、「ForumName」(キータイプハッシュ) と「Subject」(キータイプ範囲) で構成されるプライマリキーを持つ「Thread」という名前のテーブルを作成します。テーブルの作成に使用したスキーマは、図のように各 cmdlet にパイプ処理するか、-Schema パラメータを使用して指定できます。

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

出力:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

例 2: この例では、「ForumName」(キータイプハッシュ) と「Subject」(キータイプ範囲) で構成されるプライマリキーを持つ Thread という名前のテーブルを作成します。ローカルセカンダリインデックスも定義されます。ローカルセカンダリインデックスのキーは、テーブルのプライマリハッシュキー (ForumName) から自動的に設定されます。テーブルの作成に使用したスキーマは、図のように各 cmdlet にパイプ処理するか、-Schema パラメータを使用して指定できます。

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

出力:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

例 3: この例では、単一のパイプラインを使用して、「ForumName」(キータイプハッシュ) と「Subject」(キータイプ範囲) で構成されるプライマリキー、およびローカルセカンダリインデックスを持つ「Thread」という名前のテーブルを作成する方法を示します。TableSchema がパイプラインまたは -Schema パラメータから提供されない場合、Add-DDBKeySchema と Add-DDBIndexSchema によって新しい TableSchema オブジェクトが作成されます。

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

出力:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「CreateTable」を参照してください。

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

映画データを格納するためのテーブルを作成します。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                ProvisionedThroughput={
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10,
                },
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「CreateTable」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource
  attr_reader :table_name
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        {attribute_name: "year", key_type: "HASH"},  # Partition key
        {attribute_name: "title", key_type: "RANGE"}  # Sort key
      ],
      attribute_definitions: [
        {attribute_name: "year", attribute_type: "N"},
        {attribute_name: "title", attribute_type: "S"}
      ],
      provisioned_throughput: {read_capacity_units: 10, write_capacity_units:
 10})
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「CreateTable」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .provisioned_throughput(pt)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateTable」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
      " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
```

```
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「CreateTable」を参照してください。

Swift

SDK for Swift

Note

これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = CreateTableInput(
            attributeDefinitions: [
                DynamoDBClientTypes.AttributeDefinition(attributeName: "year", 
attributeType: .n),
                DynamoDBClientTypes.AttributeDefinition(attributeName: "title", 
attributeType: .s),
            ],
            keySchema: [
                DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
            ],
            provisionedThroughput: DynamoDBClientTypes.ProvisionedThroughput(
                readCapacityUnits: 10,
                writeCapacityUnits: 10
            ),
            tableName: self.tableName
        )
        let output = try await client.createTable(input: input)
        if output.tableDescription == nil {
            throw MoviesError.TableNotFound
        }
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「CreateTable」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteItem** を使用する

以下のコード例は、DeleteItem の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Deletes a single item from a DynamoDB table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table from which the item
        /// will be deleted.</param>
        /// <param name="movieToDelete">A movie object containing the title and
        /// year of the movie to delete.</param>
        /// <returns>A Boolean value indicating the success or failure of the
        /// delete operation.</returns>
        public static async Task<bool> DeleteItemAsync(
            AmazonDynamoDBClient client,
            string tableName,
            Movie movieToDelete)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = key,
            };

            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
##############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
```

```
   echo " -k keys  -- Path to json file containing the keys that identify the
item to delete."
   echo ""
 }
 while getopts "n:k:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
   return 1
 fi

 iecho "Parameters:\n"
 iecho "    table_name:   $table_name"
 iecho "    keys:   $keys"
 iecho ""

 response=$(aws dynamodb delete-item \
   --table-name "$table_name" \
   --key file://"$keys")

 local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
```

```
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteItem」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Delete an item from an Amazon DynamoDB table.
/*!
  \sa deleteItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::deleteItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteItemRequest request;

    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteItemOutcome &outcome =
 dynamoClient.DeleteItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item \"" << partitionValue << "\" deleted!" << std::endl;
    }
    else {
        std::cerr << "Failed to delete item: " << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteItem」を参照してください。

CLI

AWS CLI

例 1: 項目を削除するには

次の delete-item の例は、MusicCollection テーブルから項目を削除し、削除した項目とそのリクエストで使用された容量に関する詳細を取得します。

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

key.json の内容:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

出力:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Blue Sky Blues"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Scared of My Shadow"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 2.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

例 2: 条件付きで項目を削除するには

次の例では、ProductCategory が Sporting Goods または Gardening Supplies で、その価格が 500 および 600 の場合のみ、ProductCatalog テーブルから項目を削除します。削除された項目に関する詳細が返されます。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

names.json の内容:

```
{
    "#P": "Price"
}
```

values.json の内容:

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

出力:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteItem」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(context.TODO(),
 &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「DeleteItem」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyval>

                Where:
                    tableName - The Amazon DynamoDB table to delete the item from
(for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to delete
(for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Deleting item \"%s\" from %s\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();
        try {
            ddb.deleteItem(deleteReq);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「DeleteCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteItem」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルから項目を削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルから項目を削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteItem」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteDynamoDBItem(tableNameVal: String, keyName: String, keyVal:
 String) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request = DeleteItemRequest {
        tableName = tableNameVal
        key = keyToGet
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「DeleteItem」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「DeleteItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定されたキーと一致する DynamoDB 項目を削除します。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「DeleteItem」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

項目が特定の条件を満たす場合にのみ削除されるように条件を指定できます。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def delete_underrated_movie(self, title, year, rating):
        """
        Deletes a movie only if it is rated below a specified value. By using a
        condition expression in a delete operation, you can specify that an item
is
        deleted only when it meets certain criteria.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        :param rating: The rating threshold to check before deleting the movie.
        """
        try:
            self.table.delete_item(
                Key={"year": year, "title": title},
                ConditionExpression="info.rating <= :val",
                ExpressionAttributeValues={":val": Decimal(str(rating))},
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"ConditionalCheckFailedException":
                logger.warning(
                    "Didn't delete %s because its rating is greater than %s.",
                    title,
                    rating,
                )
            else:
                logger.error(
                    "Couldn't delete movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DeleteItem」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Deletes a movie from the table.
  #
  # @param title [String] The title of the movie to delete.
  # @param year [Integer] The release year of the movie to delete.
  def delete_item(title, year)
    @table.delete_item(key: {"year" => year, "title" => title})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete movie #{title}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteItem」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   TRY.
       DATA(lo_resp) = lo_dyn->deleteitem(
         iv_tablename                = iv_table_name
         it_key                      = it_key_input ).
       MESSAGE 'Deleted one item.' TYPE 'I'.
     CATCH /aws1/cx_dyncondalcheckfaile00.
       MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
     CATCH /aws1/cx_dynresourcenotfoundex.
       MESSAGE 'The table or index does not exist' TYPE 'E'.
     CATCH /aws1/cx_dyntransactconflictex.
       MESSAGE 'Another transaction is using the item' TYPE 'E'.
   ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「DeleteItem」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// Delete a movie, given its title and release year.
///
/// - Parameters:
///   - title: The movie's title.
///   - year: The movie's release year.
///
func delete(title: String, year: Int) async throws {
    guard let client = self.ddbClient else {
        throw MoviesError.UninitializedClient
    }

    let input = DeleteItemInput(
        key: [
            "year": .n(String(year)),
            "title": .s(title)
        ],
        tableName: self.tableName
    )
    _ = try await client.deleteItem(input: input)
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「DeleteItem」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteTable** を使用する

以下のコード例は、DeleteTable の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DAX で読み取りを高速化
- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient
client, string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteTable」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
################################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
################################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function iecho
#
```

```
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteTable」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);
    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteTable」を参照してください。

CLI

AWS CLI

テーブルを削除するには

以下の delete-table の例は MusicCollection テーブルを削除します。

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

出力:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「テーブルの削除」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteTable」を参照してください。

Go

SDK for Go V2

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable() error {
 _, err := basics.DynamoDbClient.DeleteTable(context.TODO(),
 &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
```

```
    return err
}
```

- API の詳細については、『AWS SDK for Go API リファレンス』の「DeleteTable」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
 Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteTable」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteTable」を参照してください。

SDK for JavaScript (v2)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteTable」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
    val request = DeleteTableRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「DeleteTable」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「DeleteTable」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定されたテーブルを削除します。操作を続行する前に確認画面が表示されます。

```
Remove-DDBTable -TableName "myTable"
```

例 2: 指定されたテーブルを削除します。操作を続行する前に確認画面は表示されません。

```
Remove-DDBTable -TableName "myTable" -Force
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「DeleteTable」を参照してください。

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DeleteTable」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource
  attr_reader :table_name
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteTable」を参照してください。

Rust

SDK for Rust

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteTable」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
TRY.
    lo_dyn->deletetable( iv_tablename = iv_table_name ).
    " Wait till the table is actually deleted.
    lo_dyn->get_waiter( )->tablenotexists(
      iv_max_wait_time = 200
      iv_tablename     = iv_table_name ).
    MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
  CATCH /aws1/cx_dynresourceinuseex.
    MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「DeleteTable」を参照してください。

Swift

SDK for Swift

Note

これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
///
/// Deletes the table from Amazon DynamoDB.
///
func deleteTable() async throws {
    guard let client = self.ddbClient else {
        throw MoviesError.UninitializedClient
    }

    let input = DeleteTableInput(
        tableName: self.tableName
    )
    _ = try await client.deleteTable(input: input)
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「DeleteTable」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeTable** を使用する

以下のコード例は、DescribeTable の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
private static async Task GetTableInformation()
{
    Console.WriteLine("\n*** Retrieving table information ***");

    var response = await Client.DescribeTableAsync(new DescribeTableRequest
    {
        TableName = ExampleTableName
    });

    var table = response.Table;
    Console.WriteLine($"Name: {table.TableName}");
    Console.WriteLine($"# of items: {table.ItemCount}");
    Console.WriteLine($"Provision Throughput (reads/sec): " +
                      $"{table.ProvisionedThroughput.ReadCapacityUnits}");
    Console.WriteLine($"Provision Throughput (writes/sec): " +
                      $"{table.ProvisionedThroughput.WriteCapacityUnits}");
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeTable」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
```

```
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「DescribeTable」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Describe an Amazon DynamoDB table.
/*!
  \sa describeTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::describeTable(const Aws::String &tableName,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DescribeTableOutcome &outcome =
 dynamoClient.DescribeTable(
            request);

    if (outcome.IsSuccess()) {
        const Aws::DynamoDB::Model::TableDescription &td =
 outcome.GetResult().GetTable();
        std::cout << "Table name  : " << td.GetTableName() << std::endl;
        std::cout << "Table ARN   : " << td.GetTableArn() << std::endl;
        std::cout << "Status      : "
                  <<
 Aws::DynamoDB::Model::TableStatusMapper::GetNameForTableStatus(
                          td.GetTableStatus()) << std::endl;
        std::cout << "Item count  : " << td.GetItemCount() << std::endl;
        std::cout << "Size (bytes): " << td.GetTableSizeBytes() << std::endl;
        const Aws::DynamoDB::Model::ProvisionedThroughputDescription &ptd =
 td.GetProvisionedThroughput();
        std::cout << "Throughput" << std::endl;
        std::cout << "  Read Capacity : " << ptd.GetReadCapacityUnits() <<
 std::endl;
        std::cout << "  Write Capacity: " << ptd.GetWriteCapacityUnits() <<
 std::endl;

        const Aws::Vector<Aws::DynamoDB::Model::AttributeDefinition> &ad =
 td.GetAttributeDefinitions();
        std::cout << "Attributes" << std::endl;
        for (const auto &a: ad)
            std::cout << "  " << a.GetAttributeName() << " (" <<

 Aws::DynamoDB::Model::ScalarAttributeTypeMapper::GetNameForScalarAttributeType(
                              a.GetAttributeType()) <<
                      ")" << std::endl;
    }
    else {
        std::cerr << "Failed to describe table: " <<
 outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeTable」を参照してください。

CLI

AWS CLI

テーブルを記述するには

次の describe-table の例は、MusicCollection テーブルを記述します。

```
aws dynamodb describe-table \
    --table-name MusicCollection
```

出力:

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "ACTIVE",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1421866952.062
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「表の説明」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DescribeTable」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists() (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  context.TODO(), &dynamodb.DescribeTableInput{TableName:
 aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「DescribeTable」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import
 software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughputDescription;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>
```

```
               Where:
                   tableName - The Amazon DynamoDB table to get information
about (for example, Music3).
               """;

       if (args.length != 1) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       System.out.format("Getting description for %s\n\n", tableName);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
               .region(region)
               .build();

       describeDymamoDBTable(ddb, tableName);
       ddb.close();
   }

   public static void describeDymamoDBTable(DynamoDbClient ddb, String
tableName) {
       DescribeTableRequest request = DescribeTableRequest.builder()
               .tableName(tableName)
               .build();

       try {
           TableDescription tableInfo = ddb.describeTable(request).table();
           if (tableInfo != null) {
               System.out.format("Table name  : %s\n", tableInfo.tableName());
               System.out.format("Table ARN   : %s\n", tableInfo.tableArn());
               System.out.format("Status      : %s\n", tableInfo.tableStatus());
               System.out.format("Item count  : %d\n", tableInfo.itemCount());
               System.out.format("Size (bytes): %d\n",
tableInfo.tableSizeBytes());

               ProvisionedThroughputDescription throughputInfo =
tableInfo.provisionedThroughput();
               System.out.println("Throughput");
               System.out.format("  Read Capacity : %d\n",
throughputInfo.readCapacityUnits());
```

```
                System.out.format("  Write Capacity: %d\n",
 throughputInfo.writeCapacityUnits());

                List<AttributeDefinition> attributes =
 tableInfo.attributeDefinitions();
                System.out.println("Attributes");
                for (AttributeDefinition a : attributes) {
                    System.out.format("  %s (%s)\n", a.attributeName(),
 a.attributeType());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("\nDone!");
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeTable」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DescribeTable」を参照してください。

SDK for JavaScript (v2)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「DescribeTable」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定されたテーブルの詳細を返します。

```
Get-DDBTable -TableName "myTable"
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「DescribeTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            self.table = table
        return exists
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DescribeTable」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
# Encapsulates an Amazon DynamoDB table of movie data.
```

```
class Scaffold
  attr_reader :dynamo_resource
  attr_reader :table_name
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DescribeTable」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
TRY.
    oo_result = lo_dyn->describetable( iv_tablename = iv_table_name ).
    DATA(lv_tablename) = oo_result->get_table( )->ask_tablename( ).
    DATA(lv_tablearn) = oo_result->get_table( )->ask_tablearn( ).
    DATA(lv_tablestatus) = oo_result->get_table( )->ask_tablestatus( ).
    DATA(lv_itemcount) = oo_result->get_table( )->ask_itemcount( ).
    MESSAGE 'The table name is ' && lv_tablename
        && '. The table ARN is ' && lv_tablearn
        && '. The tablestatus is ' && lv_tablestatus
        && '. Item count is ' && lv_itemcount TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && lv_tablename && ' does not exist' TYPE 'E'.
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「DescribeTable」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **ExecuteStatement** を使用する

以下のコード例は、ExecuteStatement の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- PartiQL を使用してテーブルに対してクエリを実行する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

INSERT ステートメントを使用して項目を追加します。

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
 movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
 'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

SELECT ステートメントを使用して項目を取得します。

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

SELECT ステートメントを使用して項目の一覧を取得します。

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetMovies(string tableName, int year)
```

```
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new 
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

UPDATE ステートメントを使用して項目を更新します。

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string 
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title 
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new 
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

DELETE ステートメントを使用して映画を 1 つ削除します。

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
 movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
 = ?";

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ExecuteStatement」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

INSERT ステートメントを使用して項目を追加します。

```
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
   Aws::String title;
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                          1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                 << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                 << INFO_KEY << "': ?}";

       request.SetStatement(sqlStream.str());

       // Create the parameter attributes.
       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
```

```
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

SELECT ステートメントを使用して項目を取得します。

```
    //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
 outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
 retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }
```

UPDATE ステートメントを使用して項目を更新します。

```
    //  4. Update the data for the movie using an "Update" statement.
 (ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
        }
    }
```

DELETE ステートメントを使用して項目を削除します。

```
    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ExecuteStatement」を参照してください。

Go

SDK for Go V2

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

INSERT ステートメントを使用して項目を追加します。

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

SELECT ステートメントを使用して項目を取得します。

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(title string, year int) (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

SELECT ステートメントを使用して、項目のリストを取得し、結果を射影します。

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

UPDATE ステートメントを使用して項目を更新します。

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(movie Movie, rating float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

DELETE ステートメントを使用して項目を削除します。

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

この例で使用している Movie struct を定義します。

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「ExecuteStatement」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

PartiQL を使用して項目を作成します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL を使用して項目を取得します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL を使用して項目を更新します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

PartiQL を使用して項目を削除します。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ExecuteStatement」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「ExecuteStatement」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ExecuteStatement」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

PartiQL を使用して項目を 1 つ選択します。

```
class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Gets a single record from a table using PartiQL.
  # Note: To perform more fine-grained selects,
  # use the Client.query instance method instead.
  #
  # @param title [String] The title of the movie to search.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def select_item_by_title(title)
    request = {
      statement: "SELECT * FROM \"#{@table.name}\" WHERE title=?",
      parameters: [title]
    }
    @dynamodb.client.execute_statement(request)
  end
```

PartiQL を使用して項目を更新します。

```
class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Updates a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def update_rating_by_title(title, year, rating)
    request = {
      statement: "UPDATE \"#{@table.name}\" SET info.rating=? WHERE title=? and
 year=?",
      parameters: [{ "N": rating }, title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

PartiQL を使用して項目を 1 つ追加します。

```
class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Adds a single record to a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param plot [String] The plot of the movie.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def insert_item(title, year, plot, rating)
    request = {
      statement: "INSERT INTO \"#{@table.name}\" VALUE {'title': ?, 'year': ?,
 'info': ?}",
      parameters: [title, year, {'plot': plot, 'rating': rating}]
    }
    @dynamodb.client.execute_statement(request)
  end
```

PartiQL を使用して項目を 1 つ削除します。

```
class DynamoDBPartiQLSingle

  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
```

```
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def delete_item_by_title(title, year)
    request = {
      statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
      parameters: [title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ExecuteStatement」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetItem** を使用する

以下のコード例は、GetItem の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DAX で読み取りを高速化
- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Gets information about an existing movie from the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing information about
        /// the movie to retrieve.</param>
        /// <param name="tableName">The name of the table containing the movie.</
param>
        /// <returns>A Dictionary object containing information about the item
        /// retrieved.</returns>
        public static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await client.GetItemAsync(request);
            return response.Item;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
#############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
```

```
   echo " -k keys  -- Path to json file containing the keys that identify the 
item to get."
   echo " [-q query]  -- Optional JMESPath query expression."
   echo ""
 }
 query=""
 while getopts "n:k:q:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     q) query="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
   return 1
 fi

 if [[ -n "$query" ]]; then
   response=$(aws dynamodb get-item \
     --table-name "$table_name" \
     --key file://"$keys" \
     --output text \
     --query "$query")
 else
   response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetItem」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「GetItem」を参照してください。

CLI

AWS CLI

例 1: テーブル内の項目を読み込むには

次の get-item の例は、MusicCollection テーブルから項目を取得します。テーブルにはハッシュおよび範囲プライマリキー (ArtistおよびSongTitle) があるため、これらの属性の両方を指定する必要があります。このコマンドは、オペレーションによって消費される読み込み容量に関する情報も要求します。

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

key.json の内容:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

出力:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目の読み込み」を参照してください。

例 2: 整合性のある読み込みを使用して項目を読み込むには

次の例では、強力な整合性のある読み込みを使用して MusicCollection テーブルから項目を読み込みます。

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

key.json の内容:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

出力:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目の読み込み」を参照してください。

例 3: 項目の特定の属性を取得するには

次の例は、射影式を使用して目的のアイテムの 3 つの属性のみを取得します。

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

`names.json` の内容:

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

出力:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目の読み込み」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetItem」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(title string, year int) (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(context.TODO(),
 &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「GetItem」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用して、テーブルから項目を取得します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
 retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
 tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }
    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「GetCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetItem」を参照してください。

SDK for JavaScript (v2)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルから項目を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルから項目を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「GetItem」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun getSpecificItem(tableNameVal: String, keyName: String, keyVal:
 String) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request = GetItemRequest {
        key = keyToGet
        tableName = tableNameVal
    }
    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetItem」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「GetItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: パーティションキー SongTitle とソートキー Artist を含む DynamoDB 項目を返します。

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「GetItem」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""
    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「GetItem」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Gets movie data from the table for a specific movie.
  #
  # @param title [String] The title of the movie.
  # @param year [Integer] The release year of the movie.
  # @return [Hash] The data about the requested movie.
  def get_item(title, year)
    @table.get_item(key: {"year" => year, "title" => title})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「GetItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename                = iv_table_name
      it_key                      = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
    MESSAGE 'Movie name is: ' && lo_title->get_s( )
      && 'Movie year is: ' && lo_year->get_n( )
      && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table or index does not exist' TYPE 'E'.
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「GetItem」を参照してください。

Swift

SDK for Swift

> **Note**
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// Return a `Movie` record describing the specified movie from the Amazon
/// DynamoDB table.
///
/// - Parameters:
///   - title: The movie's title (`String`).
///   - year: The movie's release year (`Int`).
///
/// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
///
/// - Returns: A `Movie` record with the movie's details.
func get(title: String, year: Int) async throws -> Movie {
    guard let client = self.ddbClient else {
        throw MoviesError.UninitializedClient
    }

    let input = GetItemInput(
        key: [
            "year": .n(String(year)),
            "title": .s(title)
        ],
        tableName: self.tableName
    )
    let output = try await client.getItem(input: input)
    guard let item = output.item else {
        throw MoviesError.ItemNotFound
    }

    let movie = try Movie(withItem: item)
    return movie
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「GetItem」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **ListTables** を使用する

以下のコード例は、ListTables の使用方法を示しています。

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    private static async Task ListMyTables()
    {
        Console.WriteLine("\n*** Listing tables ***");

        string lastTableNameEvaluated = null;
        do
        {
            var response = await Client.ListTablesAsync(new ListTablesRequest
            {
                Limit = 2,
                ExclusiveStartTableName = lastTableNameEvaluated
            });

            foreach (var name in response.TableNames)
            {
                Console.WriteLine(name);
            }

            lastTableNameEvaluated = response.LastEvaluatedTableName;
        } while (lastTableNameEvaluated != null);
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListTables」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
##############################################################################
# function dynamodb_list_tables
#
# This function lists all the tables in a DynamoDB.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_list_tables() {
  response=$(aws dynamodb list-tables \
    --output text \
    --query "TableNames")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  echo "$response" | tr -s "[:space:]" "\n"

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi
  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「ListUsers」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! List the Amazon DynamoDB tables for the current AWS account.
/*!
  \sa listTables()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::listTables(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
    listTablesRequest.SetLimit(50);
    do {
        const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamoClient.ListTables(
                listTablesRequest);
        if (!outcome.IsSuccess()) {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
        for (const auto &tableName: outcome.GetResult().GetTableNames())
            std::cout << tableName << std::endl;
        listTablesRequest.SetExclusiveStartTableName(
                outcome.GetResult().GetLastEvaluatedTableName());

    } while (!listTablesRequest.GetExclusiveStartTableName().empty());

    return true;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListTables」を参照してください。

CLI

AWS CLI

例 1: テーブルを一覧表示するには

次の `list-tables` の例では、現在の AWS アカウントとリージョンに関連するすべてのテーブルを一覧表示します。

```
aws dynamodb list-tables
```

出力:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「テーブル名のリスト化」を参照してください。

例 2: ページサイズを制限するには

次の例は、既存のすべてのテーブルのリストを返しますが、1 回の呼び出しで 1 つの項目のみを取得し、必要な場合は複数の呼び出しを実行してリスト全体を取得します。デフォルトのページサイズ (1000) を使用して大量のリソースに対してリストコマンドを実行する際に「タイムアウト」が発生する場合、ページサイズを制限してください。

```
aws dynamodb list-tables \
    --page-size 1
```

出力:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「テーブル名のリスト化」を参照してください。

例 3: 返される項目の数を制限するには

次の例は、返される項目の数を 2 に制限します。レスポンスには、次の結果ページの取得に使用する NextToken 値が含まれます。

```
aws dynamodb list-tables \
    --max-items 2
```

出力:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「テーブル名のリスト化」を参照してください。

例 4: 次の結果ページを取得するには

次のコマンドは、`list-tables` コマンドへの前の呼び出しの `NextToken` 値を使用して、次の結果ページを取得します。この場合のレスポンスには `NextToken` 値が含まれていないため、結果の最後のページに達したことがわかります。

```
aws dynamodb list-tables \
    --starting-token
 abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

出力:

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「テーブル名のリスト化」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「ListUsers」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
```

```
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables() ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}
```

- API の詳細については、『AWS SDK for Go API リファレンス』の「ListTables」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }
                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListTables」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListTables」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListTables」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun listAllTables() {
    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の「ListTables」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
            'ExclusiveStartTableName' => $exclusiveStartTableName,
            'Limit' => $limit,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「ListTables」を参照してください。

PowerShell

Tools for PowerShell

例 1: すべてのテーブルの詳細を返し、サービスが他にテーブルがないことを知らせるまで自動で繰り返します。

```
Get-DDBTableList
```

例 2: 呼び出しごとに 10 個のテーブルの詳細を返し、サービスが他にテーブルがないことを知らせるまで手動で繰り返します。

```
$nextToken = $null
do {
  Get-DDBTableList -ExclusiveStartTableName $nextToken -Limit 10
  $nextToken = $AWSHistory.LastServiceResponse.LastEvaluatedTableName
} while ($nextToken -ne $null)
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「ListTables」を参照してください。

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ListTables」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルが存在するかどうかを確認します。

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource
  attr_reader :table_name
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ListTables」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;

    println!("Tables:");

    for name in &table_names {
        println!("  {}", name);
    }

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

テーブルが存在するかどうかを確認します。

```
pub async fn table_exists(client: &Client, table: &str) -> Result<bool, Error> {
    debug!("Checking for table: {table}");
    let table_list = client.list_tables().send().await;

    match table_list {
        Ok(list) => Ok(list.table_names().contains(&table.into())),
        Err(e) => Err(e.into()),
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「ListTables」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        oo_result = lo_dyn->listtables( ).
        " You can loop over the oo_result to get table properties like this.
        LOOP AT oo_result->get_tablenames( ) INTO DATA(lo_table_name).
          DATA(lv_tablename) = lo_table_name->get_value( ).
        ENDLOOP.
        DATA(lv_tablecount) = lines( oo_result->get_tablenames( ) ).
        MESSAGE 'Found ' && lv_tablecount && ' tables' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「ListTables」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// Get a list of the DynamoDB tables available in the specified Region.
///
/// - Returns: An array of strings listing all of the tables available
///   in the Region specified when the session was created.
public func getTableList() async throws -> [String] {
    var tableList: [String] = []
    var lastEvaluated: String? = nil

    // Iterate over the list of tables, 25 at a time, until we have the
    // names of every table. Add each group to the `tableList` array.
    // Iteration is complete when `output.lastEvaluatedTableName` is `nil`.

    repeat {
        let input = ListTablesInput(
            exclusiveStartTableName: lastEvaluated,
            limit: 25
        )
        let output = try await self.session.listTables(input: input)
        guard let tableNames = output.tableNames else {
            return tableList
        }
        tableList.append(contentsOf: tableNames)
        lastEvaluated = output.lastEvaluatedTableName
    } while lastEvaluated != nil
```

```
        return tableList
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「ListTables」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **PutItem** を使用する

以下のコード例は、PutItem の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DAX で読み取りを高速化
- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Adds a new item to the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing informtation for
        /// the movie to add to the table.</param>
        /// <param name="tableName">The name of the table where the item will be
added.</param>
        /// <returns>A Boolean value that indicates the results of adding the
item.</returns>
        public static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
Movie newMovie, string tableName)
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「PutItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> ⓘ Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_put_item
#
```

```
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
```

```
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
```

```
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「PutItem」を参照してください。

C++

SDK for C++

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「PutItem」を参照してください。

CLI

AWS CLI

例 1: テーブルに項目を追加するには

次の `put-item` の例は、MusicCollection テーブルに新しい項目を追加します。

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

`item.json` の内容:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

出力:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

例 2: テーブル内の項目を条件付きで上書きするには

次の put-item の例は、MusicCollection テーブル内の既存の項目に Greatest Hits の値を持つ AlbumTitle 属性がある場合にのみ、その項目を上書きします。このコマンドは、その項目の以前の値を返します。

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

item.json の内容:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

names.json の内容:

```
{
    "#A": "AlbumTitle"
}
```

values.json の内容:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

出力:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

キーが存在する場合は、次のような出力が表示されます。

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「PutItem」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
```

```
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(context.TODO(), &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
```

```
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「PutItem」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用してテーブルに項目を配置します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is
 placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
                    keyval - The key value that represents the item to get (for
 example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
 About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
 Happy Day).
                **Warning** This program will  place an item that you specify
 into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
```

```
       String awards = args[5];
       String awardVal = args[6];
       String songTitle = args[7];
       String songTitleVal = args[8];

       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
               .region(region)
               .build();

       putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
               songTitleVal);
       System.out.println("Done!");
       ddb.close();
   }

   public static void putItemInTable(DynamoDbClient ddb,
           String tableName,
           String key,
           String keyVal,
           String albumTitle,
           String albumTitleValue,
           String awards,
           String awardVal,
           String songTitle,
           String songTitleVal) {

       HashMap<String, AttributeValue> itemValues = new HashMap<>();
       itemValues.put(key, AttributeValue.builder().s(keyVal).build());
       itemValues.put(songTitle,
AttributeValue.builder().s(songTitleVal).build());
       itemValues.put(albumTitle,
AttributeValue.builder().s(albumTitleValue).build());
       itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

       PutItemRequest request = PutItemRequest.builder()
               .tableName(tableName)
               .item(itemValues)
               .build();

       try {
           PutItemResponse response = ddb.putItem(request);
```

```
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「PutItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「PutCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「PutItem」を参照してください。

SDK for JavaScript (v2)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルに項目を配置します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
```

```
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

DynamoDB ドキュメントクライアントを使用して、テーブルに項目を配置します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「PutItem」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request = PutItemRequest {
        tableName = tableNameVal
        item = itemValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「PutItem」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「PutItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: 新しい項目を作成する、または既存の項目を新しい項目で置き換えます。

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「PutItem」を参照してください。

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「PutItem」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        "year" => movie[:year],
        "title" => movie[:title],
        "info" => {"plot" => movie[:plot], "rating" => movie[:rating]}})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「PutItem」を参照してください。

Rust

SDK for Rust

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「PutItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「PutItem」を参照してください。

Swift

SDK for Swift

> **Note**
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
/// table.
///
/// - Parameter movie: The `Movie` to add to the table.
///
func add(movie: Movie) async throws {
    guard let client = self.ddbClient else {
        throw MoviesError.UninitializedClient
    }

    // Get a DynamoDB item containing the movie data.
    let item = try await movie.getAsItem()

    // Send the `PutItem` request to Amazon DynamoDB.

    let input = PutItemInput(
        item: item,
        tableName: self.tableName
    )
    _ = try await client.putItem(input: input)
}
```

```
    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「PutItem」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **Query** を使用する

以下のコード例は、Query の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DAX で読み取りを高速化
- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Queries the table for movies released in a particular year and
        /// then displays the information for the movies returned.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table to query.</param>
        /// <param name="year">The release year for which we want to
        /// view movies.</param>
        /// <returns>The number of movies that match the query.</returns>
        public static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
client, string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            var filter = new QueryFilter("year", QueryOperator.Equal, year);
            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "title",
                  "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            Search search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Query」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
```

```
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_names" ]]; then
   errecho "ERROR: You must provide a attribute names with the -a parameter."
   usage
```

```
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
```

```
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「Query」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::QueryRequest request;
```

```
   request.SetTableName(tableName);

   if (!projectionExpression.empty()) {
       request.SetProjectionExpression(projectionExpression);
   }

   // Set query key condition expression.
   request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

   // Set Expression AttributeValues.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
   attributeValues.emplace(":valueToMatch", partitionValue);

   request.SetExpressionAttributeValues(attributeValues);

   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
                         << std::endl;
               // Iterate each item and print.
               for (const auto &item: items) {
                   std::cout
                           <<
"******************************************************"
                           << std::endl;
                   // Output each retrieved field and its value.
```

```
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「Query」を参照してください。

CLI

AWS CLI

例 1: テーブルにクエリを実行するには

次の query の例では、MusicCollection テーブルの項目にクエリを実行します。テーブルにはハッシュおよび範囲プライマリキー (ArtistおよびSongTitle) がありますが、このクエリではハッシュキー値のみを指定します。「No One You Know」という名前のアーティストの曲タイトルが返されます。

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

`expression-attributes.json` の内容:

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 2: 強力な整合性のある読み込みを使用してテーブルにクエリを実行し、インデックスを降順で走査するには

次の例では、最初の例と同じクエリを実行しますが、結果は逆の順序で返され、強力な整合性のある読み込みが使用されます。

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

`expression-attributes.json` の内容:

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 3: 特定の結果を除外するには

次の例は、MusicCollection をクエリしますが、AlbumTitle 属性に特定の値を含む結果を除外します。このフィルターは項目が読み込まれた後に適用されるため、ScannedCount または ConsumedCapacity には影響しないことに注意してください。

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

values.json の内容:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

names.json の内容:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

出力:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 4: 項目数だけを取得するには

次の例は、クエリに一致する項目数を取得しますが、項目自体は取得しません。

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

`expression-attributes.json` の内容:

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

例 5: インデックスをクエリするには

次の例は、ローカルセカンダリインデックス `AlbumTitleIndex` をクエリします。クエリは、ローカルセカンダリインデックスに射影されたベーステーブルのすべての属性を返します。ローカルセカンダリインデックスまたはグローバルセカンダリインデックスをクエリする場合は、`table-name` パラメータを使用してベーステーブルの名前も指定する必要があることに注意してください。

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

`expression-attributes.json` の内容:

```
{
    ":v1": {"S": "No One You Know"}
}
```

出力:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB のクエリオペレーション」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「Query」を参照してください。

Go

SDK for Go V2

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
```

```
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(releaseYear int) ([]Movie, error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(context.TODO())
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「Query」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用してテーブルに対してクエリを実行します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
```

```
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

DynamoDbClient とセカンダリインデックスを使用してテーブルに対してクエリを実行します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));
```

```
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「Query」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「QueryCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「Query」を参照してください。

SDK for JavaScript (v2)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「Query」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request = QueryRequest {
        tableName = tableNameVal
        keyConditionExpression = "$partitionAlias = :$partitionKeyName"
        expressionAttributeNames = attrNameAlias
        this.expressionAttributeValues = attrValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
```

```
        val response = ddb.query(request)
        return response.count
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「Query」を参照してください。

PHP

SDK for PHP

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「Query」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定された SongTitle と Artist を含む DynamoDB 項目を返すクエリを呼び出します。

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
```

```
AlbumTitle                     Somewhat Famous
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「Query」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

キー条件式を使用して項目に対してクエリを実行します。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

項目に対してクエリを実行し、データのサブセットを返すように射影します。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「Query」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: "#yr = :year",
      expression_attribute_names: {"#yr" => "year"},
      expression_attribute_values: {":year" => year})
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「Query」を参照してください。

Rust

SDK for Rust

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

指定した年に作られた映画を検索します。

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「Query」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「Query」を参照してください。

Swift

SDK for Swift

> **Note**
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// Get all the movies released in the specified year.
///
/// - Parameter year: The release year of the movies to return.
///
/// - Returns: An array of `Movie` objects describing each matching movie.
///
```

```
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = QueryInput(
            expressionAttributeNames: [
                "#y": "year"
            ],
            expressionAttributeValues: [
                ":y": .n(String(year))
            ],
            keyConditionExpression: "#y = :y",
            tableName: self.tableName
        )
        let output = try await client.query(input: input)

        guard let items = output.items else {
            throw MoviesError.ItemNotFound
        }

        // Convert the found movies into `Movie` objects and return an array
        // of them.

        var movieList: [Movie] = []
        for item in items {
            let movie = try Movie(withItem: item)
            movieList.append(movie)
        }
        return movieList
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「Query」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **Scan** を使用する

以下のコード例は、Scan の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DAX で読み取りを高速化
- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                  { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「Scan」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> ### Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
```

```
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
```

```
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$filter_expression" ]]; then
   errecho "ERROR: You must provide a filter expression with the -f parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_names" ]]; then
   errecho "ERROR: You must provide expression attribute names with the -a 
parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_values" ]]; then
   errecho "ERROR: You must provide expression attribute values with the -v 
parameter."
   usage
   return 1
 fi

 if [[ -z "$projection_expression" ]]; then
   response=$(aws dynamodb scan \
     --table-name "$table_name" \
     --filter-expression "$filter_expression" \
     --expression-attribute-names file://"$expression_attribute_names" \
     --expression-attribute-values file://"$expression_attribute_values")
```

```
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
```

```
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「Scan」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Scan an Amazon DynamoDB table.
/*!
  \sa scanTable()
  \param tableName: Name for the DynamoDB table.
  \param projectionExpression: An optional projection expression, ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                 const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::ScanRequest request;
    request.SetTableName(tableName);

    if (!projectionExpression.empty())
        request.SetProjectionExpression(projectionExpression);

    Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
 all_items;
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 last_evaluated_key; // Used for pagination;
    do {
        if (!last_evaluated_key.empty()) {
            request.SetExclusiveStartKey(last_evaluated_key);
        }
        const Aws::DynamoDB::Model::ScanOutcome &outcome =
 dynamoClient.Scan(request);
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            all_items.insert(all_items.end(), items.begin(), items.end());

            last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Scan items: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }

    } while (!last_evaluated_key.empty());

    if (!all_items.empty()) {
        std::cout << "Number of items retrieved from scan: " << all_items.size()
                  << std::endl;
        // Iterate each item and print.
        for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 &itemMap: all_items) {
            std::cout << "*******************************************************"
                      << std::endl;
            // Output each retrieved field and its value.
            for (const auto &itemEntry: itemMap)
                std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
                          << std::endl;
        }
    }

    else {
        std::cout << "No items found in table: " << tableName << std::endl;
    }

    return true;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「Scan」を参照してください。

CLI

AWS CLI

テーブルをスキャンするには

次の scan の例は、MusicCollection テーブル全体をスキャンし、その結果をアーティスト「No One You Know」の曲に絞り込みます。各項目について、アルバムタイトルと曲タイトルのみが返されます。

```
aws dynamodb scan \
    --table-name MusicCollection \
    --filter-expression "Artist = :a" \
    --projection-expression "#ST, #AT" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json
```

`expression-attribute-names.json` の内容:

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

`expression-attribute-values.json` の内容:

```
{
    ":a": {"S": "No One You Know"}
}
```

出力:

```
{
    "Count": 2,
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "AlbumTitle": {
                "S": "Somewhat Famous"
            }
        },
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            },
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            }
        }
    ],
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「DynamoDB でのスキャンの使用」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「Scan」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(startYear int, endYear int) ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(context.TODO())
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
```

```
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「Scan」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用して Amazon DynamoDB テーブルをスキャンします。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To scan items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, See the EnhancedScanRecords example.
 */

public class DynamoDBScanItems {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <tableName>
                Where:
                    tableName - The Amazon DynamoDB table to get information from
 (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        scanItems(ddb, tableName);
        ddb.close();
    }

    public static void scanItems(DynamoDbClient ddb, String tableName) {
        try {
            ScanRequest scanRequest = ScanRequest.builder()
                    .tableName(tableName)
                    .build();

            ScanResponse response = ddb.scan(scanRequest);
            for (Map<String, AttributeValue> item : response.items()) {
                Set<String> keys = item.keySet();
                for (String key : keys) {
                    System.out.println("The key name is " + key + "\n");
                    System.out.println("The value is " + item.get(key).s());
                }
            }

        } catch (DynamoDbException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「Scan」を参照してください。

JavaScript

SDK for JavaScript (v3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「ScanCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「Scan」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values
 you want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

- 詳細については、「AWS SDK for JavaScript デベロッパーガイド」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「Scan」を参照してください。

Kotlin

SDK for Kotlin

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun scanItems(tableNameVal: String) {
    val request = ScanRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「Scan」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

    public function scan(string $tableName, array $key, string $filters)
    {
        $query = [
            'ExpressionAttributeNames' => ['#year' => 'year'],
            'ExpressionAttributeValues' => [
                ":min" => ['N' => '1990'],
                ":max" => ['N' => '1999'],
            ],
            'FilterExpression' => "#year between :min and :max",
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->scan($query);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「Scan」を参照してください。

PowerShell

Tools for PowerShell

例 1: Music テーブルのすべての項目を返します。

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

例 2: Music テーブル内の CriticRating が 9 以上の項目を返します。

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「Scan」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「Scan」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Scans for movies that were released in a range of years.
  # Uses a projection expression to return a subset of data for each movie.
  #
  # @param year_range [Hash] The range of years to retrieve.
  # @return [Array] The list of movies released in the specified years.
  def scan_items(year_range)
    movies = []
    scan_hash = {
      filter_expression: "#yr between :start_yr and :end_yr",
      projection_expression: "#yr, title, info.rating",
      expression_attribute_names: {"#yr" => "year"},
      expression_attribute_values: {
        ":start_yr" => year_range[:start], ":end_yr" => year_range[:end]}
    }
    done = false
    start_key = nil
    until done
      scan_hash[:exclusive_start_key] = start_key unless start_key.nil?
      response = @table.scan(scan_hash)
      movies.concat(response.items) unless response.items.empty?
      start_key = response.last_evaluated_key
      done = start_key.nil?
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't scan for movies. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    movies
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「Scan」を参照してください。

Rust

SDK for Rust

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
 Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("   {:?}", item);
    }
    Ok(())
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「Scan」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    TRY.
        " Scan movies for rating greater than or equal to the rating specified
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_rating }| ) ) ).
        DATA(lt_filter_conditions) = VALUE /aws1/
cl_dyncondition=>tt_filterconditionmap(
          ( VALUE /aws1/cl_dyncondition=>ts_filterconditionmap_maprow(
          key = 'rating'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |GE|
          ) ) ) ).
        oo_scan_result = lo_dyn->scan( iv_tablename = iv_table_name
          it_scanfilter = lt_filter_conditions ).
        DATA(lt_items) = oo_scan_result->get_items( ).
        LOOP AT lt_items INTO DATA(lo_item).
          " You can loop over to get individual attributes.
          DATA(lo_title) = lo_item[ key = 'title' ]-value.
          DATA(lo_year) = lo_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_scan_result->get_count( ).
        MESSAGE 'Found ' && lv_count && ' items' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
```

```
       MESSAGE 'The table or index does not exist' TYPE 'E'.
   ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「Scan」を参照してください。

Swift

SDK for Swift

> **Note**
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// Return an array of `Movie` objects released in the specified range of
   /// years.
   ///
   /// - Parameters:
   ///   - firstYear: The first year of movies to return.
   ///   - lastYear: The last year of movies to return.
   ///   - startKey: A starting point to resume processing; always use `nil`.
   ///
   /// - Returns: An array of `Movie` objects describing the matching movies.
   ///
   /// > Note: The `startKey` parameter is used by this function when
   ///   recursively calling itself, and should always be `nil` when calling
   ///   directly.
   ///
   func getMovies(firstYear: Int, lastYear: Int,
                  startKey: [Swift.String:DynamoDBClientTypes.AttributeValue]? =
nil)
```

```
                   async throws -> [Movie] {
        var movieList: [Movie] = []

        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = ScanInput(
            consistentRead: true,
            exclusiveStartKey: startKey,
            expressionAttributeNames: [
                "#y": "year"             // `year` is a reserved word, so use `#y` 
instead.
            ],
            expressionAttributeValues: [
                ":y1": .n(String(firstYear)),
                ":y2": .n(String(lastYear))
            ],
            filterExpression: "#y BETWEEN :y1 AND :y2",
            tableName: self.tableName
        )

        let output = try await client.scan(input: input)

        guard let items = output.items else {
            return movieList
        }

        // Build an array of `Movie` objects for the returned items.

        for item in items {
            let movie = try Movie(withItem: item)
            movieList.append(movie)
        }

        // Call this function recursively to continue collecting matching
        // movies, if necessary.

        if output.lastEvaluatedKey != nil {
            let movies = try await self.getMovies(firstYear: firstYear, lastYear: 
lastYear,
                         startKey: output.lastEvaluatedKey)
            movieList += movies
        }
        return movieList
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「Scan」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **UpdateItem** を使用する

以下のコード例は、UpdateItem の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- テーブル、項目、クエリで使用を開始する

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        /// <summary>
        /// Updates an existing item in the movies table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing information for
        /// the movie to update.</param>
        /// <param name="newInfo">A MovieInfo object that contains the
        /// information that will be changed.</param>
```

```
       /// <param name="tableName">The name of the table that contains the
movie.</param>
       /// <returns>A Boolean value that indicates the success of the
operation.</returns>
       public static async Task<bool> UpdateItemAsync(
           AmazonDynamoDBClient client,
           Movie newMovie,
           MovieInfo newInfo,
           string tableName)
       {
           var key = new Dictionary<string, AttributeValue>
           {
               ["title"] = new AttributeValue { S = newMovie.Title },
               ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
           };
           var updates = new Dictionary<string, AttributeValueUpdate>
           {
               ["info.plot"] = new AttributeValueUpdate
               {
                   Action = AttributeAction.PUT,
                   Value = new AttributeValue { S = newInfo.Plot },
               },

               ["info.rating"] = new AttributeValueUpdate
               {
                   Action = AttributeAction.PUT,
                   Value = new AttributeValue { N = newInfo.Rank.ToString() },
               },
           };

           var request = new UpdateItemRequest
           {
               AttributeUpdates = updates,
               Key = key,
               TableName = tableName,
           };

           var response = await client.UpdateItemAsync(request);

           return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
       }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「UpdateItem」を参照してください。

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
##############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item to update.
#       -e update expression  -- An expression that defines one or more attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
#############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
```

```
   echo " -n table_name  -- The name of the table."
   echo " -k keys  -- Path to json file containing the keys that identify the
item to update."
   echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
   echo " -v values  -- Path to json file containing the update values."
   echo ""
 }

 while getopts "n:k:e:v:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     e) update_expression="${OPTARG}" ;;
     v) values="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
   return 1
 fi
 if [[ -z "$update_expression" ]]; then
   errecho "ERROR: You must provide an update expression with the -e parameter."
   usage
   return 1
 fi
```

```
  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

この例で使用されているユーティリティ関数。

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「UpdateItem」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                 const Aws::String &attributeKey,
                                 const Aws::String &attributeValue,
                                 const Aws::Client::ClientConfiguration
&clientConfiguration) {
   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   // *** Define UpdateItem request arguments.
   // Define TableName argument.
   Aws::DynamoDB::Model::UpdateItemRequest request;
   request.SetTableName(tableName);

   // Define KeyName argument.
   Aws::DynamoDB::Model::AttributeValue attribValue;
   attribValue.SetS(partitionValue);
   request.AddKey(partitionKey, attribValue);

   // Construct the SET update expression argument.
   Aws::String update_expression("SET #a = :valueA");
   request.SetUpdateExpression(update_expression);

   // Construct attribute name argument.
   Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
   expressionAttributeNames["#a"] = attributeKey;
   request.SetExpressionAttributeNames(expressionAttributeNames);

   // Construct attribute value argument.
   Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
   attributeUpdatedValue.SetS(attributeValue);
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
expressionAttributeValues;
   expressionAttributeValues[":valueA"] = attributeUpdatedValue;
   request.SetExpressionAttributeValues(expressionAttributeValues);

   // Update the item.
   const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
dynamoClient.UpdateItem(
           request);
   if (outcome.IsSuccess()) {
       std::cout << "Item was updated" << std::endl;
   }
   else {
       std::cerr << outcome.GetError().GetMessage() << std::endl;
   }
```

```
    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「UpdateItem」を参照してください。

CLI

AWS CLI

例 1: テーブル内の項目を更新するには

次の update-item の例では、MusicCollection テーブルの項目を更新します。新しい属性 (Year) を追加して、AlbumTitle 属性を更新します。更新後に表示される項目内の属性はすべて、レスポンスで返されます。

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

key.json の内容:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

expression-attribute-names.json の内容:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

expression-attribute-values.json の内容:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

出力:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

例 2: 項目を条件付きで更新するには

次の例は、既存の項目に Year 属性がない場合にのみ、MusicCollection テーブル内の項目を更新します。

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

key.json の内容:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

expression-attribute-names.json の内容:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

expression-attribute-values.json の内容:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

項目にすでに Year 属性がある場合、DynamoDB は次の出力を返します。

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

詳細については、「Amazon DynamoDB ディベロッパーガイド」の「項目を書き込みます」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「UpdateItem」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(context.TODO(),
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}



// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「UpdateItem」を参照してください。

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDbClient を使用してテーブルで項目を更新します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「UpdateItem」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

この例では、ドキュメントクライアントを使用して DynamoDB での項目の操作を簡略化しています。API の詳細については、「UpdateCommand」を参照してください。

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「UpdateItem」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] = AttributeValueUpdate {
        value = AttributeValue.S(updateVal)
        action = AttributeAction.Put
    }

    val request = UpdateItemRequest {
        tableName = tableNameVal
        key = itemKey
        attributeUpdates = updatedValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「UpdateItem」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「UpdateItem」を参照してください。

PowerShell

Tools for PowerShell

例 1: パーティションキー SongTitle とソートキー Artist を含む DynamoDB 項目のジャンル属性を「Rap」に設定します。

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

出力:

```
Name                           Value
----                           -----
Genre                          Rap
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「UpdateItem」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

更新式を使用して項目を更新します。

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

算術演算を含む更新式を使用して、項目を更新します。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

特定の条件を満たす場合にのみ、項目を更新します。

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「UpdateItem」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class DynamoDBBasics
  attr_reader :dynamo_resource
  attr_reader :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: "us-east-1")
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
```

```
 end

 # Updates rating and plot data for a movie in the table.
 #
 # @param movie [Hash] The title, year, plot, rating of the movie.
 def update_item(movie)

   response = @table.update_item(
     key: {"year" => movie[:year], "title" => movie[:title]},
     update_expression: "set info.rating=:r",
     expression_attribute_values: { ":r" => movie[:rating] },
     return_values: "UPDATED_NEW")
 rescue Aws::DynamoDB::Errors::ServiceError => e
   puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
   puts("\t#{e.code}: #{e.message}")
   raise
 else
   response.attributes
 end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「UpdateItem」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   TRY.
       oo_output = lo_dyn->updateitem(
         iv_tablename        = iv_table_name
         it_key              = it_item_key
         it_attributeupdates = it_attribute_updates ).
       MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
```

```
     CATCH /aws1/cx_dyncondalcheckfaile00.
       MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
     CATCH /aws1/cx_dynresourcenotfoundex.
       MESSAGE 'The table or index does not exist' TYPE 'E'.
     CATCH /aws1/cx_dyntransactconflictex.
       MESSAGE 'Another transaction is using the item' TYPE 'E'.
   ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「UpdateItem」を参照してください。

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// Update the specified movie with new `rating` and `plot` information.
   ///
   /// - Parameters:
   ///   - title: The title of the movie to update.
   ///   - year: The release year of the movie to update.
   ///   - rating: The new rating for the movie.
   ///   - plot: The new plot summary string for the movie.
   ///
   /// - Returns: An array of mappings of attribute names to their new
   ///   listing each item actually changed. Items that didn't need to change
   ///   aren't included in this list. `nil` if no changes were made.
```

```
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
                -> [Swift.String:DynamoDBClientTypes.AttributeValue]? {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        // Build the update expression and the list of expression attribute
        // values. Include only the information that's changed.

        var expressionParts: [String] = []
        var attrValues: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]

        if rating != nil {
            expressionParts.append("info.rating=:r")
            attrValues[":r"] = .n(String(rating!))
        }
        if plot != nil {
            expressionParts.append("info.plot=:p")
            attrValues[":p"] = .s(plot!)
        }
        let expression: String = "set \(expressionParts.joined(separator: ", "))"

        let input = UpdateItemInput(
            // Create substitution tokens for the attribute values, to ensure
            // no conflicts in expression syntax.
            expressionAttributeValues: attrValues,
            // The key identifying the movie to update consists of the release
            // year and title.
            key: [
                "year": .n(String(year)),
                "title": .s(title)
            ],
            returnValues: .updatedNew,
            tableName: self.tableName,
            updateExpression: expression
        )
        let output = try await client.updateItem(input: input)

        guard let attributes: [Swift.String:DynamoDBClientTypes.AttributeValue] =
output.attributes else {
            throw MoviesError.InvalidAttributes
        }
        return attributes
    }

```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「UpdateItem」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **UpdateTable** を使用する

以下のコード例は、UpdateTable の使用方法を示しています。

CLI

AWS CLI

例 1: テーブルの請求モードを変更するには

次の update-table 例は、MusicCollection テーブルにプロビジョニングされた読み取り/書き込みキャパシティを増やします。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T13:18:18.921000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        }
    }
}
```

詳細については、「Amazon DynamoDB デベロッパーガイド」の「テーブルの更新」を参照してください。

例 2: グローバルセカンダリインデックスを作成するには

次の例は、MusicCollection テーブルにグローバルセカンダリインデックスを追加します。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

gsi-updates.json の内容:

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "Backfilling": false,
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ]
    }
}
```

詳細については、「Amazon DynamoDB デベロッパーガイド」の「テーブルの更新」を参照してください。

例 3: テーブルで DynamoDB Streams を有効化するには

次のコマンドは、`MusicCollection` テーブルで DynamoDB Streams を有効化します。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112"
    }
}
```

詳細については、「Amazon DynamoDB デベロッパーガイド」の「テーブルの更新」を参照してください。

例 4: サーバー側の暗号化を有効化するには

次の例は、MusicCollection テーブルでサーバー側の暗号化を有効します。

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

出力:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112",
        "SSEDescription": {
            "Status": "UPDATING"
        }
    }
}
```

詳細については、「Amazon DynamoDB デベロッパーガイド」の「テーブルの更新」を参照してください。

- API の詳細については、「AWS CLI Command Reference」の「UpdateTable」を参照してください。

PowerShell

Tools for PowerShell

例 1: 指定されたテーブルのプロビジョンされたスループットを更新します。

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference」の「UpdateTable」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した DynamoDB のシナリオ

次のコード例は、AWS SDK を使用して DynamoDB で一般的なシナリオを実装する方法を示しています。これらのシナリオは、DynamoDB 内で複数の関数を呼び出すことによって特定のタスクを実行する方法を示しています。それぞれのシナリオには、GitHub へのリンクがあり、コードを設定および実行する方法についての説明が記載されています。

例

AWS SDK を使用して DAX で DynamoDB の読み取りを高速化する

次のコードサンプルは、以下の操作方法を示しています。

- DAX クライアントと SDK クライアントの両方でデータを作成してテーブルに書き込みます。
- 両方のクライアントでテーブルを取得、クエリ、スキャンし、パフォーマンスを比較します。

詳細については、「DynamoDB Accelerator クライアントで開発する」を参照してください。

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DAX または Boto3 クライアントでテーブルを作成します。

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "ProvisionedThroughput": {"ReadCapacityUnits": 10, "WriteCapacityUnits":
 10},
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

テーブルにテストデータを書き込みます。

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate
 the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

DAX クライアントと Boto3 クライアントの両方について、いくつかのイテレーションの項目を取得し、それぞれに費やした時間を報告します。

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each
 iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")
    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(
                    Key={"partition_key": partition_key, "sort_key": sort_key}
                )
                print(".", end="")
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the DAX client."
        )
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax
            )
    else:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the Boto3 client."
        )
        test_start, test_end = get_item_test(test_key_count, test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/ test_iterations}."
    )
```

DAX クライアントと Boto3 クライアントの両方について、いくつかのイテレーションのテーブルに対してクエリを実行し、それぞれに費やした時間を報告します。

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this
 value.
    :param sort_keys: The range of sort key values for the query. The query
 returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)
    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations,
 dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

DAX クライアントと Boto3 クライアントの両方について、いくつかのイテレーションのテーブルをスキャンし、それぞれに費やした時間を報告します。

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

テーブルを削除する。

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Query
  - Scan

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK で DynamoDB テーブル、項目、クエリの使用を開始する

次のコード例は、以下を実行する方法を示しています。

- 映画データを保持できるテーブルを作成する。
- テーブルに 1 つの映画を入れ、取得して更新する。
- サンプル JSON ファイルから映画データをテーブルに書き込む。
- 特定の年にリリースされた映画を照会する。
- 何年もの間にリリースされた映画をスキャンする。
- テーブルからムービーを削除し、テーブルを削除します。

.NET

AWS SDK for .NET

Note

GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// This example application performs the following basic Amazon DynamoDB
// functions:
//
//     CreateTableAsync
//     PutItemAsync
//     UpdateItemAsync
//     BatchWriteItemAsync
//     GetItemAsync
//     DeleteItemAsync
//     Query
//     Scan
//     DeleteItemAsync
//
using Amazon.DynamoDBv2;
using DynamoDB_Actions;

public class DynamoDB_Basics
{
    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();

        var tableName = "movie_table";

        // Relative path to moviedata.json in the local repository.
        var movieFileName = @"..\..\..\..\..\..\..\..\resources\sample_files
\movies.json";

        DisplayInstructions();
        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await DynamoDbMethods.CreateMovieTableAsync(client,
 tableName);

        if (success)
        {
            Console.WriteLine($"\nTable: {tableName} successfully created.");
        }
        else
        {
            Console.WriteLine($"\nCould not create {tableName}.");
        }

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await DynamoDbMethods.PutItemAsync(client, newMovie,
 tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous"
 +
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await DynamoDbMethods.UpdateItemAsync(client, newMovie,
 newInfo, tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie:
 {newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await DynamoDbMethods.BatchWriteItemsAsync(client,
 movieFileName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await DynamoDbMethods.GetItemAsync(client, lookupMovie,
 tableName);
        if (item.Count > 0)
        {
            DynamoDbMethods.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
```

```
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await DynamoDbMethods.DeleteItemAsync(client, tableName,
movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await DynamoDbMethods.QueryMoviesAsync(client,
tableName, findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await DynamoDbMethods.ScanTableAsync(client, tableName,
startYear, endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();
        // Delete the table.
        success = await DynamoDbMethods.DeleteTableAsync(client, tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is done.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using
 DynamoDB with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
 sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
 just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in
 a given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released
 within a range of years.");
```

```
        Console.WriteLine("\t9. Finally, it deletes the table that was just
 created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        Console.WriteLine("\nPress <Enter> to continue.");
        Console.WriteLine(SepBar);
        _ = Console.ReadLine();
    }
}
```

ムービーデータを含めるテーブルを作成します。

```
        /// <summary>
        /// Creates a new Amazon DynamoDB table and then waits for the new
        /// table to become active.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="tableName">The name of the table to create.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> CreateMovieTableAsync(AmazonDynamoDBClient
 client, string tableName)
        {
            var response = await client.CreateTableAsync(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 5,
                    WriteCapacityUnits = 5,
                },
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                System.Threading.Thread.Sleep(sleepDuration);

                var describeTableResponse = await
 client.DescribeTableAsync(request);
```

```
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
```

1 つのムービーをテーブルに追加します。

```
        /// <summary>
        /// Adds a new item to the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing informtation for
        /// the movie to add to the table.</param>
        /// <param name="tableName">The name of the table where the item will be
 added.</param>
        /// <returns>A Boolean value that indicates the results of adding the
 item.</returns>
        public static async Task<bool> PutItemAsync(AmazonDynamoDBClient client,
 Movie newMovie, string tableName)
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            var response = await client.PutItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

テーブルの 1 つの項目を更新します。

```
        /// <summary>
        /// Updates an existing item in the movies table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing information for
        /// the movie to update.</param>
        /// <param name="newInfo">A MovieInfo object that contains the
        /// information that will be changed.</param>
        /// <param name="tableName">The name of the table that contains the
 movie.</param>
        /// <returns>A Boolean value that indicates the success of the
 operation.</returns>
        public static async Task<bool> UpdateItemAsync(
            AmazonDynamoDBClient client,
            Movie newMovie,
            MovieInfo newInfo,
            string tableName)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            var response = await client.UpdateItemAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

ムービーテーブルから 1 つの項目を取得します。

```
        /// <summary>
        /// Gets information about an existing movie from the table.
        /// </summary>
        /// <param name="client">An initialized Amazon DynamoDB client object.</
param>
        /// <param name="newMovie">A Movie object containing information about
        /// the movie to retrieve.</param>
        /// <param name="tableName">The name of the table containing the movie.</
param>
        /// <returns>A Dictionary object containing information about the item
        /// retrieved.</returns>
        public static async Task<Dictionary<string, AttributeValue>>
 GetItemAsync(AmazonDynamoDBClient client, Movie newMovie, string tableName)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await client.GetItemAsync(request);
            return response.Item;
        }
```

項目のバッチをムービーテーブルに書き込みます。

```
        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonSerializer.Deserialize<List<Movie>>(
                json,
                new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });

            // Now return the first 250 entries.
            return allMovies.GetRange(0, 250);
        }

        /// <summary>
        /// Writes 250 items to the movie table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="movieFileName">A string containing the full path to
        /// the JSON file containing movie data.</param>
        /// <returns>A long integer value representing the number of movies
        /// imported from the JSON file.</returns>
        public static async Task<long> BatchWriteItemsAsync(
            AmazonDynamoDBClient client,
            string movieFileName)
        {
            var movies = ImportMovies(movieFileName);
            if (movies is null)
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContext(client);

            var movieBatch = context.CreateBatchWrite<Movie>();
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
```

テーブルから 1 つの項目を削除します。

```
        /// <summary>
        /// Deletes a single item from a DynamoDB table.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table from which the item
        /// will be deleted.</param>
        /// <param name="movieToDelete">A movie object containing the title and
        /// year of the movie to delete.</param>
        /// <returns>A Boolean value indicating the success or failure of the
        /// delete operation.</returns>
        public static async Task<bool> DeleteItemAsync(
            AmazonDynamoDBClient client,
            string tableName,
            Movie movieToDelete)
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
 movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = key,
            };

            var response = await client.DeleteItemAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

特定の年にリリースされたムービーのテーブルにクエリを実行します。

```
        /// <summary>
        /// Queries the table for movies released in a particular year and
        /// then displays the information for the movies returned.
        /// </summary>
        /// <param name="client">The initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table to query.</param>
        /// <param name="year">The release year for which we want to
        /// view movies.</param>
        /// <returns>The number of movies that match the query.</returns>
        public static async Task<int> QueryMoviesAsync(AmazonDynamoDBClient
 client, string tableName, int year)
        {
            var movieTable = Table.LoadTable(client, tableName);
            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "title",
                  "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            Search search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
```

数年にわたってリリースされたムービーのテーブルをスキャンします。

```
        public static async Task<int> ScanTableAsync(
            AmazonDynamoDBClient client,
            string tableName,
            int startYear,
            int endYear)
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                  { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await client.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response.LastEvaluatedKey.Count > 0);
            return foundCount;
        }
```

ムービーテーブルを削除します。

```
        public static async Task<bool> DeleteTableAsync(AmazonDynamoDBClient
client, string tableName)
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await client.DeleteTableAsync(request);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
                return true;
            }
            else
            {
                Console.WriteLine("Could not delete table.");
                return false;
            }
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Bash

Bash スクリプトを使用した AWS CLI

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDB の開始シナリオ。

```
###############################################################################
# function dynamodb_getting_started_movies
#
# Scenario to create an Amazon DynamoDB table and perform a series of operations
 on the table.
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
###############################################################################
function dynamodb_getting_started_movies() {

  source ./dynamodb_operations.sh

  key_schema_json_file="dynamodb_key_schema.json"
  attribute_definitions_json_file="dynamodb_attr_def.json"
  item_json_file="movie_item.json"
  key_json_file="movie_key.json"
  batch_json_file="batch.json"
  attribute_names_json_file="attribute_names.json"
  attributes_values_json_file="attribute_values.json"

  echo_repeat "*" 88
  echo
  echo "Welcome to the Amazon DynamoDB getting started demo."
  echo
  echo_repeat "*" 88
  echo

  local table_name
  echo -n "Enter a name for a new DynamoDB table: "
  get_input
  table_name=$get_input_result

  local provisioned_throughput="ReadCapacityUnits=5,WriteCapacityUnits=5"

  echo '[
  {"AttributeName": "year", "KeyType": "HASH"},
   {"AttributeName": "title", "KeyType": "RANGE"}
  ]' >"$key_schema_json_file"

  echo '[
  {"AttributeName": "year", "AttributeType": "N"},
   {"AttributeName": "title", "AttributeType": "S"}
  ]' >"$attribute_definitions_json_file"

  if dynamodb_create_table -n "$table_name" -a "$attribute_definitions_json_file"
 \
    -k "$key_schema_json_file" -p "$provisioned_throughput" 1>/dev/null; then
    echo "Created a DynamoDB table named $table_name"
  else
    errecho "The table failed to create. This demo will exit."
    clean_up
    return 1
  fi

  echo "Waiting for the table to become active...."

  if dynamodb_wait_table_active -n "$table_name"; then
    echo "The table is now active."
  else
    errecho "The table failed to become active. This demo will exit."
    cleanup "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo -n "Enter the title of a movie you want to add to the table: "
  get_input
  local added_title
  added_title=$get_input_result
  local added_year
  get_int_input "What year was it released? "
  added_year=$get_input_result

  local rating
  get_float_input "On a scale of 1 - 10, how do you rate it? " "1" "10"
  rating=$get_input_result

  local plot
  echo -n "Summarize the plot for me: "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"},
    "info": {"M" : {"plot": {"S" : "'"$plot"'"}, "rating":
{"N" :"'"$rating"'"} } }
   }' >"$item_json_file"

  if dynamodb_put_item -n "$table_name" -i "$item_json_file"; then
    echo "The movie '$added_title' was successfully added to the table
'$table_name'."
  else
    errecho "Put item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "Let's update your movie '$added_title'."
  get_float_input "You rated it $rating, what new rating would you give it? " "1"
"10"
  rating=$get_input_result

  echo -n "You summarized the plot as '$plot'."
  echo "What would you say now? "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"}
    }' >"$key_json_file"

  echo '{
    ":r": {"N" :"'"$rating"'"},
    ":p": {"S" : "'"$plot"'"}
   }' >"$item_json_file"

  local update_expression="SET info.rating = :r, info.plot = :p"

  if dynamodb_update_item -n "$table_name" -k "$key_json_file" -e
"$update_expression" -v "$item_json_file"; then
    echo "Updated '$added_title' with new attributes."
  else
    errecho "Update item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "We will now use batch write to upload 150 movie entries into the table."

  local batch_json
  for batch_json in movie_files/movies_*.json; do
    echo "{ \"$table_name\" : $(<"$batch_json") }" >"$batch_json_file"
    if dynamodb_batch_write_item -i "$batch_json_file" 1>/dev/null; then
      echo "Entries in $batch_json added to table."
    else
      errecho "Batch write failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  done

  local title="The Lord of the Rings: The Fellowship of the Ring"
  local year="2001"

  if get_yes_no_input "Let's move on...do you want to get info about '$title'?
(y/n) "; then
    echo '{
```

```
  "year": {"N" :"'"$year"'"},
  "title": {"S" :  "'"$title"'"}
  }' >"$key_json_file"
    local info
    info=$(dynamodb_get_item -n "$table_name" -k "$key_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Get item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$info"
  fi

  local ask_for_year=true
  while [[ "$ask_for_year" == true ]]; do
    echo "Let's get a list of movies released in a given year."
    get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
    year=$get_input_result
    echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

    echo '{
    ":v": {"N" :"'"$year"'"}
    }' >"$attributes_values_json_file"

    response=$(dynamodb_query -n "$table_name" -k "#n=:v" -a
 "$attribute_names_json_file" -v "$attributes_values_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Query table failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$response"

    if ! get_yes_no_input "Try another year? (y/n) "; then
```

```
      ask_for_year=false
    fi
  done

  echo "Now let's scan for movies released in a range of years. Enter a year: "
  get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
  local start=$get_input_result

  get_int_input "Enter another year: " "1972" "2018"
  local end=$get_input_result

  echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

  echo '{
    ":v1": {"N" : "'"$start"'"},
    ":v2": {"N" : "'"$end"'"}
    }' >"$attributes_values_json_file"

  response=$(dynamodb_scan -n "$table_name" -f "#n BETWEEN :v1 AND :v2" -a
 "$attribute_names_json_file" -v "$attributes_values_json_file")

  # shellcheck disable=SC2181
  if [[ ${?} -ne 0 ]]; then
    errecho "Scan table failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo "Here is what I found:"
  echo "$response"

  echo
  echo_repeat "*" 88
  echo

  echo "Let's remove your movie '$added_title' from the table."

  if get_yes_no_input "Do you want to remove '$added_title'? (y/n) "; then
    echo '{
  "year": {"N" :"'"$added_year"'"},
  "title": {"S" :  "'"$added_title"'"}
  }' >"$key_json_file"
```

```
    if ! dynamodb_delete_item -n "$table_name" -k "$key_json_file"; then
      errecho "Delete item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  fi

  if get_yes_no_input "Do you want to delete the table '$table_name'? (y/n) ";
 then
    if ! clean_up "$table_name"; then
      return 1
    fi
  else
    if ! clean_up; then
      return 1
    fi
  fi

  return 0
}
```

このシナリオで使用される DynamoDB 関数。

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#       -p provisioned_throughput -- Provisioned throughput settings for the
 table.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
```

```
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema provisioned_throughput response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key types."
    echo " -p provisioned_throughput -- Provisioned throughput settings for the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      p) provisioned_throughput="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
   return 1
 fi

 if [[ -z "$attribute_definitions" ]]; then
   errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
   usage
   return 1
 fi

 if [[ -z "$key_schema" ]]; then
   errecho "ERROR: You must provide a key schema json file path the -k
parameter."
   usage
   return 1
 fi

 if [[ -z "$provisioned_throughput" ]]; then
   errecho "ERROR: You must provide a provisioned throughput json file path the
-p parameter."
   usage
   return 1
 fi

 iecho "Parameters:\n"
 iecho "    table_name:   $table_name"
 iecho "    attribute_definitions:   $attribute_definitions"
 iecho "    key_schema:   $key_schema"
 iecho "    provisioned_throughput:   $provisioned_throughput"
 iecho ""

 response=$(aws dynamodb create-table \
   --table-name "$table_name" \
   --attribute-definitions file://"$attribute_definitions" \
   --key-schema file://"$key_schema" \
   --provisioned-throughput "$provisioned_throughput")

 local error_code=${?}

 if [[ $error_code -ne 0 ]]; then
   aws_cli_error_log $error_code
   errecho "ERROR: AWS reports create-table operation failed.$response"
   return 1
 fi
```

```
  return 0
}

###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}

##############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
```

```
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi
  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
##############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.
  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
    echo " -e update expression  -- An expression that defines one or more
 attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
  }

  while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
```

```
    return 1
  fi
  if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
```

```
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  #######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}

#############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
```

```
   echo " -k keys  -- Path to json file containing the keys that identify the
item to get."
   echo " [-q query]  -- Optional JMESPath query expression."
   echo ""
 }
 query=""
 while getopts "n:k:q:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     q) query="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
   return 1
 fi

 if [[ -n "$query" ]]; then
   response=$(aws dynamodb get-item \
     --table-name "$table_name" \
     --key file://"$keys" \
     --output text \
     --query "$query")
 else
   response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}

#############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
 parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi
```

```
  echo "$response"

  return 0
}

#############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
 parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
 parameter."
```

```
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

##############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###########################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  #######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

このシナリオで使用されるユーティリティ関数。

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」で以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    {
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table with partition: year (N) and sort: title (S).
 (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::dynamodbGettingStartedScenario(clientConfig);

            // 9. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }
    }

//! Scenario to modify and query a DynamoDB table.
/*!
  \sa dynamodbGettingStartedScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::dynamodbGettingStartedScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;
    std::cout << "Welcome to the Amazon DynamoDB getting started demo." <<
 std::endl;
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie.
    Aws::String title;
```

```
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate 
it? ",
                                         1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::PutItemRequest putItemRequest;
       putItemRequest.SetTableName(MOVIE_TABLE_NAME);

       putItemRequest.AddItem(YEAR_KEY,
                              
Aws::DynamoDB::Model::AttributeValue().SetN(year));
       putItemRequest.AddItem(TITLE_KEY,
                              
Aws::DynamoDB::Model::AttributeValue().SetS(title));

       // Create attribute for the info map.
       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute = 
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute = 
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);

       putItemRequest.AddItem(INFO_KEY, infoMapAttribute);

       Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
               putItemRequest);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to add an item: " << 
outcome.GetError().GetMessage()
```

```
                     << std::endl;
           return false;
       }
   }

   std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
             << std::endl;

   // 3. Update the rating and plot of the movie by using an update expression.
   {
       rating = askQuestionForFloatRange(
               Aws::String("\nLet's update your movie.\nYou rated it  ") +
               std::to_string(rating)
               + ", what new rating would you give it? ", 1, 10);
       plot = askQuestion(Aws::String("You summarized the plot as '") + plot +
                          "'.\nWhat would you say now? ");

       Aws::DynamoDB::Model::UpdateItemRequest request;
       request.SetTableName(MOVIE_TABLE_NAME);
       request.AddKey(TITLE_KEY,
Aws::DynamoDB::Model::AttributeValue().SetS(title));
       request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
       std::stringstream expressionStream;
       expressionStream << "set " << INFO_KEY << "." << RATING_KEY << " =:r, "
                        << INFO_KEY << "." << PLOT_KEY << " =:p";
       request.SetUpdateExpression(expressionStream.str());
       request.SetExpressionAttributeValues({
                                                    {":r",
Aws::DynamoDB::Model::AttributeValue().SetN(
                                                            rating)},
                                                    {":p",
Aws::DynamoDB::Model::AttributeValue().SetS(
                                                            plot)}
                                            });

       request.SetReturnValues(Aws::DynamoDB::Model::ReturnValue::UPDATED_NEW);

       const Aws::DynamoDB::Model::UpdateItemOutcome &result =
dynamoClient.UpdateItem(
               request);
       if (!result.IsSuccess()) {
           std::cerr << "Error updating movie " + result.GetError().GetMessage()
                     << std::endl;
           return false;
       }
   }

   std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

   // 4. Put 250 movies in the table from moviedata.json.
   {
       std::cout << "Adding movies from a json file to the database." <<
std::endl;
       const size_t MAX_SIZE_FOR_BATCH_WRITE = 25;
       const size_t MOVIES_TO_WRITE = 10 * MAX_SIZE_FOR_BATCH_WRITE;
       Aws::String jsonString = getMovieJSON();
       if (!jsonString.empty()) {
           Aws::Utils::Json::JsonValue json(jsonString);
           Aws::Utils::Array<Aws::Utils::Json::JsonView> movieJsons =
json.View().AsArray();
           Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;

           // To add movies with a cross-section of years, use an appropriate
increment
           // value for iterating through the database.
           size_t increment = movieJsons.GetLength() / MOVIES_TO_WRITE;
           for (size_t i = 0; i < movieJsons.GetLength(); i += increment) {
               writeRequests.push_back(Aws::DynamoDB::Model::WriteRequest());
               Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
putItems = movieJsonViewToAttributeMap(
                       movieJsons[i]);
               Aws::DynamoDB::Model::PutRequest putRequest;
               putRequest.SetItem(putItems);
               writeRequests.back().SetPutRequest(putRequest);
               if (writeRequests.size() == MAX_SIZE_FOR_BATCH_WRITE) {
                   Aws::DynamoDB::Model::BatchWriteItemRequest request;
                   request.AddRequestItems(MOVIE_TABLE_NAME, writeRequests);
                   const Aws::DynamoDB::Model::BatchWriteItemOutcome &outcome =
dynamoClient.BatchWriteItem(
                           request);
                   if (!outcome.IsSuccess()) {
                       std::cerr << "Unable to batch write movie data: "
                                 << outcome.GetError().GetMessage()
                                 << std::endl;
                       writeRequests.clear();
                       break;
                   }
                   else {
                       std::cout << "Added batch of " << writeRequests.size()
                                 << " movies to the database."
                                 << std::endl;
                   }
                   writeRequests.clear();
               }
           }
       }
   }

   std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
             << std::endl;

   // 5. Get a movie by Key (partition + sort).
   {
       Aws::String titleToGet("King Kong");
       Aws::String answer = askQuestion(Aws::String(
               "Let's move on...Would you like to get info about '" + titleToGet
+
               "'? (y/n) "));
       if (answer == "y") {
           Aws::DynamoDB::Model::GetItemRequest request;
           request.SetTableName(MOVIE_TABLE_NAME);
           request.AddKey(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(titleToGet));
           request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(1933));

           const Aws::DynamoDB::Model::GetItemOutcome &result =
dynamoClient.GetItem(
                   request);
           if (!result.IsSuccess()) {
               std::cerr << "Error " << result.GetError().GetMessage();
           }
           else {
               const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = result.GetResult().GetItem();
               if (!item.empty()) {
                   std::cout << "\nHere's what I found:" << std::endl;
                   printMovieInfo(item);
               }
               else {
                   std::cout << "\nThe movie was not found in the database."
                             << std::endl;
               }
           }
       }
   }

   // 6. Use Query with a key condition expression to return all movies
   //    released in a given year.
   Aws::String doAgain = "n";
   do {
       Aws::DynamoDB::Model::QueryRequest req;

       req.SetTableName(MOVIE_TABLE_NAME);

       // "year" is a DynamoDB reserved keyword and must be replaced with an
       // expression attribute name.
       req.SetKeyConditionExpression("#dynobase_year = :valueToMatch");
       req.SetExpressionAttributeNames({{"#dynobase_year", YEAR_KEY}});

       int yearToMatch = askQuestionForIntRange(
               "\nLet's get a list of movies released in"
               " a given year. Enter a year between 1972 and 2018 ",
               1972, 2018);
       Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
       attributeValues.emplace(":valueToMatch",
                               Aws::DynamoDB::Model::AttributeValue().SetN(
                                       yearToMatch));
       req.SetExpressionAttributeValues(attributeValues);

       const Aws::DynamoDB::Model::QueryOutcome &result =
dynamoClient.Query(req);
       if (result.IsSuccess()) {
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "\nThere were " << items.size()
                         << " movies in the database from "
                         << yearToMatch << "." << std::endl;
               for (const auto &item: items) {
                   printMovieInfo(item);
               }
               doAgain = "n";
           }
           else {
               std::cout << "\nNo movies from " << yearToMatch
                         << " were found in the database"
                         << std::endl;
               doAgain = askQuestion(Aws::String("Try another year? (y/n) "));
           }
       }
       else {
           std::cerr << "Failed to Query items: " <<
result.GetError().GetMessage()
                     << std::endl;
       }

   } while (doAgain == "y");

   //  7. Use Scan to return movies released within a range of years.
   //     Show how to paginate data using ExclusiveStartKey. (Scan +
FilterExpression)
   {
       int startYear = askQuestionForIntRange("\nNow let's scan a range of years
"
                                              "for movies in the database. Enter
a start year: ",
                                              1972, 2018);
       int endYear = askQuestionForIntRange("\nEnter an end year: ",
                                            startYear, 2018);
       Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
       do {
           Aws::DynamoDB::Model::ScanRequest scanRequest;
           scanRequest.SetTableName(MOVIE_TABLE_NAME);
           scanRequest.SetFilterExpression(
                   "#dynobase_year >= :startYear AND #dynobase_year
<= :endYear");
           scanRequest.SetExpressionAttributeNames({{"#dynobase_year",
YEAR_KEY}});

           Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
           attributeValues.emplace(":startYear",
                                   Aws::DynamoDB::Model::AttributeValue().SetN(
                                           startYear));
           attributeValues.emplace(":endYear",
                                   Aws::DynamoDB::Model::AttributeValue().SetN(
                                           endYear));
           scanRequest.SetExpressionAttributeValues(attributeValues);

           if (!exclusiveStartKey.empty()) {
               scanRequest.SetExclusiveStartKey(exclusiveStartKey);
           }

           const Aws::DynamoDB::Model::ScanOutcome &result = dynamoClient.Scan(
                   scanRequest);
           if (result.IsSuccess()) {
               const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
               if (!items.empty()) {
                   std::stringstream stringStream;
                   stringStream << "\nFound " << items.size() << " movies in one
scan."
                                << " How many would you like to see? ";
                   size_t count = askQuestionForInt(stringStream.str());
                   for (size_t i = 0; i < count && i < items.size(); ++i) {
                       printMovieInfo(items[i]);
                   }
               }
               else {
                   std::cout << "\nNo movies in the database between " <<
startYear <<
                             " and " << endYear << "." << std::endl;
               }

               exclusiveStartKey = result.GetResult().GetLastEvaluatedKey();
               if (!exclusiveStartKey.empty()) {
                   std::cout << "Not all movies were retrieved. Scanning for
more."
                             << std::endl;
               }
               else {
                   std::cout << "All movies were retrieved with this scan."
                             << std::endl;
               }
           }
           else {
               std::cerr << "Failed to Scan movies: "
                         << result.GetError().GetMessage() << std::endl;
           }
```

```
        } while (!exclusiveStartKey.empty());
    }

    // 8. Delete a movie. (DeleteItem)
    {
        std::stringstream stringStream;
        stringStream << "\nWould you like to delete the movie " << title
                     << " from the database? (y/n) ";
        Aws::String answer = askQuestion(stringStream.str());
        if (answer == "y") {
            Aws::DynamoDB::Model::DeleteItemRequest request;
            request.AddKey(YEAR_KEY,
 Aws::DynamoDB::Model::AttributeValue().SetN(year));
            request.AddKey(TITLE_KEY,
                           Aws::DynamoDB::Model::AttributeValue().SetS(title));
            request.SetTableName(MOVIE_TABLE_NAME);

            const Aws::DynamoDB::Model::DeleteItemOutcome &result =
 dynamoClient.DeleteItem(
                    request);
            if (result.IsSuccess()) {
                std::cout << "\nRemoved \"" << title << "\" from the database."
                          << std::endl;
            }
            else {
                std::cerr << "Failed to delete the movie: "
                          << result.GetError().GetMessage()
                          << std::endl;
            }
        }
    }

    return true;
}

//! Routine to convert a JsonView object to an attribute map.
/*!
  \sa movieJsonViewToAttributeMap()
  \param jsonView: Json view object.
  \return map: Map that can be used in a DynamoDB request.
 */
Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
AwsDoc::DynamoDB::movieJsonViewToAttributeMap(
        const Aws::Utils::Json::JsonView &jsonView) {
```

```
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> result;

    if (jsonView.KeyExists(YEAR_KEY)) {
        result[YEAR_KEY].SetN(jsonView.GetInteger(YEAR_KEY));
    }
    if (jsonView.KeyExists(TITLE_KEY)) {
        result[TITLE_KEY].SetS(jsonView.GetString(TITLE_KEY));
    }
    if (jsonView.KeyExists(INFO_KEY)) {
        Aws::Map<Aws::String, const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue>> infoMap;
        Aws::Utils::Json::JsonView infoView = jsonView.GetObject(INFO_KEY);
        if (infoView.KeyExists(RATING_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetN(infoView.GetDouble(RATING_KEY));
            infoMap.emplace(std::make_pair(RATING_KEY, attributeValue));
        }
        if (infoView.KeyExists(PLOT_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetS(infoView.GetString(PLOT_KEY));
            infoMap.emplace(std::make_pair(PLOT_KEY, attributeValue));
        }

        result[INFO_KEY].SetM(infoMap);
    }

    return result;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
```

```
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
```

```
                   << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Go

SDK for Go V2

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

対話型シナリオを実行してテーブルを作成し、そのテーブルに対してアクションを実行します。

```
// RunMovieScenario is an interactive example that shows you how to use the AWS
 SDK for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//   1. Create a table that can hold movie data.
//   2. Put, get, and update a single movie in the table.
//   3. Write movie data to the table from a sample JSON file.
//   4. Query for movies that were released in a given year.
//   5. Scan for movies that were released in a range of years.
//   6. Delete a movie from the table.
//   7. Delete the table.
//
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is
 loaded
// into the named table.
func RunMovieScenario(
 sdkConfig aws.Config, questioner demotools.IQuestioner, tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB getting started demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{TableName: tableName,
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

 exists, err := tableBasics.TableExists()
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable()
  if err != nil {
   panic(err)
  } else {
```

```
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}

var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 []demotools.IAnswerValidator{demotools.NotEmpty{}})
customMovie.Year = questioner.AskInt("What year was it released?",
 []demotools.IAnswerValidator{demotools.NotEmpty{}, demotools.InIntRange{
  Lower: 1900, Upper: 2030}})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:", []demotools.IAnswerValidator{
  demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10}})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 []demotools.IAnswerValidator{demotools.NotEmpty{}})
err = tableBasics.AddMovie(customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
 "What new rating would you give it?", []demotools.IAnswerValidator{
  demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10}})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 []demotools.IAnswerValidator{demotools.NotEmpty{}})
attributes, err := tableBasics.UpdateMovie(customMovie)
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
   log.Printf("\t%v: %v\n", valKey, val)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
[]demotools.IAnswerValidator{
  demotools.InIntRange{Lower: 1, Upper: show}},
)
movie, err := tableBasics.GetMovie(movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
 []demotools.IAnswerValidator{demotools.InIntRange{Lower: 1972, Upper: 2018}},
)
releases, err := tableBasics.Query(releaseYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))
log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ", []demotools.IAnswerValidator{
 demotools.InIntRange{Lower: 1972, Upper: 2018}})
endYear := questioner.AskInt("Enter another year: ",
[]demotools.IAnswerValidator{
 demotools.InIntRange{Lower: 1972, Upper: 2018}})
releases, err = tableBasics.Scan(startYear, endYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n",
startYear, endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables()
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
   log.Printf("\t%v", table)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
 err = tableBasics.DeleteMovie(customMovie)
}
if err == nil {
 log.Printf("Deleted %v.\n", customMovie.Title)
}

if questioner.AskBool("Delete the table, too? (y/n)", "y") {
  err = tableBasics.DeleteTable()
 } else {
  log.Println("Don't forget to delete the table when you're done or you might " +
   "incur charges on your account.")
 }
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

この例で使用している Movie struct を定義します。

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

DynamoDB アクションを呼び出す構造体およびメソッドを作成します。

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists() (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  context.TODO(), &dynamodb.DescribeTableInput{TableName:
 aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
```

```
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable() (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(context.TODO(),
 &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName: aws.String(basics.TableName),
  ProvisionedThroughput: &types.ProvisionedThroughput{
   ReadCapacityUnits:  aws.Int64(10),
   WriteCapacityUnits: aws.Int64(10),
  },
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(context.TODO(), &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables() ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(context.TODO(), &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(context.TODO(),
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}




// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(movies []Movie, maxMovies int) (int,
 error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(context.TODO(),
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(title string, year int) (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(context.TODO(),
 &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(releaseYear int) ([]Movie, error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(context.TODO())
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
```

```
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(startYear int, endYear int) ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(context.TODO())
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(context.TODO(),
 &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable() error {
 _, err := basics.DynamoDbClient.DeleteTable(context.TODO(),
 &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Java

SDK for Java 2.x

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDB テーブルを作成します。

```
    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("year")
                .attributeType("N")
                .build());

        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("title")
                .attributeType("S")
                .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
                .attributeName("year")
                .keyType(KeyType.HASH)
                .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
                .attributeName("title")
                .keyType(KeyType.RANGE)
                .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
                .keySchema(tableKey)
                .provisionedThroughput(ProvisionedThroughput.builder()
                        .readCapacityUnits(10L)
                        .writeCapacityUnits(10L)
                        .build())
                .attributeDefinitions(attributeDefinitions)
                .tableName(tableName)
                .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();
```

```
           // Wait until the Amazon DynamoDB table is created.
           WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
           waiterResponse.matched().response().ifPresent(System.out::println);
           String newTable = response.tableDescription().tableName();
           System.out.println("The " + newTable + " was successfully created.");

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

サンプルの JSON ファイルをダウンロードして抽出するヘルパー関数を作成します。

```
   // Load data into the table.
   public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
       DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
               .dynamoDbClient(ddb)
               .build();

       DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
       JsonParser parser = new JsonFactory().createParser(new File(fileName));
       com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
       Iterator<JsonNode> iter = rootNode.iterator();
       ObjectNode currentNode;
       int t = 0;
       while (iter.hasNext()) {
           // Only add 200 Movies to the table.
           if (t == 200)
               break;
           currentNode = (ObjectNode) iter.next();

           int year = currentNode.path("year").asInt();
           String title = currentNode.path("title").asText();
           String info = currentNode.path("info").toString();

           Movies movies = new Movies();
           movies.setYear(year);
```

```
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }
```

テーブルから項目を取得します。

```
    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
                .n("1933")
                .build());

        keyToGet.put("title", AttributeValue.builder()
                .s("King Kong")
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName("Movies")
                .build();

        try {
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

完全な例です。

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This Java example performs these tasks:
 *
 * 1. Creates the Amazon DynamoDB Movie table with partition and sort key.
 * 2. Puts data into the Amazon DynamoDB table from a JSON document using the
 * Enhanced client.
 * 3. Gets data from the Movie table.
 * 4. Adds a new item.
 * 5. Updates an item.
 * 6. Uses a Scan to query items using the Enhanced client.
 * 7. Queries all items where the year is 2013 using the Enhanced Client.
 * 8. Deletes the table.
 */

public class Scenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        final String usage = """

                Usage:
                    <fileName>

                Where:
                    fileName - The path to the moviedata.json file that you can
 download from the Amazon DynamoDB Developer Guide.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = "Movies";
        String fileName = args[0];
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon DynamoDB example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
                "1. Creating an Amazon DynamoDB table named Movies with a key
 named year and a sort key named title.");
        createTable(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Loading data into the Amazon DynamoDB table.");
        loadData(ddb, tableName, fileName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Getting data from the Movie table.");
        getItem(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Putting a record into the Amazon DynamoDB
 table.");
        putRecord(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("5. Updating a record.");
        updateTableItem(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Scanning the Amazon DynamoDB table.");
        scanMovies(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Querying the Movies released in 2013.");
        queryTable(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        System.out.println(DASHES);

        ddb.close();
    }

    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("year")
                .attributeType("N")
                .build());

        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("title")
                .attributeType("S")
                .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
                .attributeName("year")
                .keyType(KeyType.HASH)
                .build();
        KeySchemaElement key2 = KeySchemaElement.builder()
                .attributeName("title")
                .keyType(KeyType.RANGE)
                .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
                .keySchema(tableKey)
                .provisionedThroughput(ProvisionedThroughput.builder()
                        .readCapacityUnits(10L)
                        .writeCapacityUnits(10L)
                        .build())
                .attributeDefinitions(attributeDefinitions)
                .tableName(tableName)
                .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Query the table.
    public static void queryTable(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
 DynamoDbEnhancedClient.builder()
                    .dynamoDbClient(ddb)
```

```
                   .build();

           DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
           QueryConditional queryConditional = QueryConditional
                   .keyEqualTo(Key.builder()
                           .partitionValue(2013)
                           .build());

           // Get items in the table and write out the ID value.
           Iterator<Movies> results =
custTable.query(queryConditional).items().iterator();
           String result = "";

           while (results.hasNext()) {
               Movies rec = results.next();
               System.out.println("The title of the movie is " +
rec.getTitle());
               System.out.println("The movie information  is " + rec.getInfo());
           }

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   // Scan the table.
   public static void scanMovies(DynamoDbClient ddb, String tableName) {
       System.out.println("******* Scanning all movies.\n");
       try {
           DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                   .dynamoDbClient(ddb)
                   .build();

           DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
           Iterator<Movies> results = custTable.scan().items().iterator();
           while (results.hasNext()) {
               Movies rec = results.next();
               System.out.println("The movie title is " + rec.getTitle());
               System.out.println("The movie year is " + rec.getYear());
           }
       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   // Load data into the table.
   public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
       DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
               .dynamoDbClient(ddb)
               .build();

       DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
       JsonParser parser = new JsonFactory().createParser(new File(fileName));
       com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
       Iterator<JsonNode> iter = rootNode.iterator();
       ObjectNode currentNode;
       int t = 0;
       while (iter.hasNext()) {
           // Only add 200 Movies to the table.
           if (t == 200)
               break;
           currentNode = (ObjectNode) iter.next();

           int year = currentNode.path("year").asInt();
           String title = currentNode.path("title").asText();
           String info = currentNode.path("info").toString();

           Movies movies = new Movies();
           movies.setYear(year);
           movies.setTitle(title);
           movies.setInfo(info);

           // Put the data into the Amazon DynamoDB Movie table.
           mappedTable.putItem(movies);
           t++;
       }
   }

   // Update the record to include show only directors.
   public static void updateTableItem(DynamoDbClient ddb, String tableName) {
       HashMap<String, AttributeValue> itemKey = new HashMap<>();
       itemKey.put("year", AttributeValue.builder().n("1933").build());
       itemKey.put("title", AttributeValue.builder().s("King Kong").build());

       HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
       updatedValues.put("info", AttributeValueUpdate.builder()
               .value(AttributeValue.builder().s("{\"directors\":[\"Merian C.
Cooper\",\"Ernest B. Schoedsack\"]")
                       .build())
               .action(AttributeAction.PUT)
               .build());

       UpdateItemRequest request = UpdateItemRequest.builder()
               .tableName(tableName)
               .key(itemKey)
               .attributeUpdates(updatedValues)
               .build();

       try {
           ddb.updateItem(request);
       } catch (ResourceNotFoundException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }

       System.out.println("Item was updated!");
   }

   public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
       DeleteTableRequest request = DeleteTableRequest.builder()
               .tableName(tableName)
               .build();

       try {
           ddb.deleteTable(request);

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
       System.out.println(tableName + " was successfully deleted!");
   }

   public static void putRecord(DynamoDbClient ddb) {
       try {
           DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                   .dynamoDbClient(ddb)
                   .build();

           DynamoDbTable<Movies> table = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));

           // Populate the Table.
           Movies record = new Movies();
           record.setYear(2020);
           record.setTitle("My Movie2");
           record.setInfo("no info");
           table.putItem(record);

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
       System.out.println("Added a new movie to the table.");
   }

   public static void getItem(DynamoDbClient ddb) {

       HashMap<String, AttributeValue> keyToGet = new HashMap<>();
       keyToGet.put("year", AttributeValue.builder()
               .n("1933")
               .build());

       keyToGet.put("title", AttributeValue.builder()
               .s("King Kong")
               .build());

       GetItemRequest request = GetItemRequest.builder()
               .key(keyToGet)
               .tableName("Movies")
               .build();

```

```
        try {
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import { readFileSync } from "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";
```

```
const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });
  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so
 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/
latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });
  const updateResponse = await docClient.send(updateCommand);
  log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

  /**
   * Delete a movie from the table.
   */

  log("Deleting a single movie from the table.");
  const deleteCommand = new DeleteCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
  });
  await client.send(deleteCommand);
  log("Movie deleted.");

  /**
   * Upload a batch of movies.
   */

  log("Adding movies from local JSON file.");
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );
  const movies = JSON.parse(file.toString());
  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);
  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        [tableName]: putRequests,
      },
    });

    await docClient.send(command);
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log(`Scan for movies released between 1980 and 1990`);
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression.
 Scan will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDB テーブルを作成します。

```
suspend fun createScenarioTable(tableNameVal: String, key: String) {
    val attDef = AttributeDefinition {
        attributeName = key
        attributeType = ScalarAttributeType.N
    }

    val attDef1 = AttributeDefinition {
        attributeName = "title"
        attributeType = ScalarAttributeType.S
    }

    val keySchemaVal = KeySchemaElement {
        attributeName = key
        keyType = KeyType.Hash
    }

    val keySchemaVal1 = KeySchemaElement {
        attributeName = "title"
        keyType = KeyType.Range
    }

    val provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef, attDef1)
        keySchema = listOf(keySchemaVal, keySchemaVal1)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->

        val response = ddb.createTable(request)
        ddb.waitUntilTableExists { // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

サンプルの JSON ファイルをダウンロードして抽出するヘルパー関数を作成します。

```
// Load data into the table.
suspend fun loadData(tableName: String, fileName: String) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request = PutItemRequest {
        tableName = tableNameVal
        item = itemValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

テーブルから項目を取得します。

```
suspend fun getMovie(tableNameVal: String, keyName: String, keyVal: String) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request = GetItemRequest {
        key = keyToGet
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

完全な例です。

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
          <fileName>

        Where:
           fileName - The path to the moviedata.json you can download from the
 Amazon DynamoDB Developer Guide.
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(1)
    }

    // Get the moviedata.json from the Amazon DynamoDB Developer Guide.
    val tableName = "Movies"
    val fileName = args[0]
    val partitionAlias = "#a"

    println("Creating an Amazon DynamoDB table named Movies with a key named id
 and a sort key named title.")
    createScenarioTable(tableName, "year")
    loadData(tableName, fileName)
    getMovie(tableName, "year", "1933")
    scanMovies(tableName)
    val count = queryMovieTable(tableName, "year", partitionAlias)
    println("There are $count Movies released in 2013.")
    deletIssuesTable(tableName)
}

suspend fun createScenarioTable(tableNameVal: String, key: String) {
    val attDef = AttributeDefinition {
        attributeName = key
        attributeType = ScalarAttributeType.N
    }

    val attDef1 = AttributeDefinition {
        attributeName = "title"
        attributeType = ScalarAttributeType.S
    }
    val keySchemaVal = KeySchemaElement {
        attributeName = key
        keyType = KeyType.Hash
    }

    val keySchemaVal1 = KeySchemaElement {
        attributeName = "title"
        keyType = KeyType.Range
    }

    val provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef, attDef1)
        keySchema = listOf(keySchemaVal, keySchemaVal1)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->

        val response = ddb.createTable(request)
        ddb.waitUntilTableExists { // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}

// Load data into the table.
suspend fun loadData(tableName: String, fileName: String) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request = PutItemRequest {
        tableName = tableNameVal
        item = itemValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}

suspend fun getMovie(tableNameVal: String, keyName: String, keyVal: String) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request = GetItemRequest {
        key = keyToGet
        tableName = tableNameVal
    }
    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}

suspend fun deletIssuesTable(tableNameVal: String) {
    val request = DeleteTableRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun queryMovieTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionAlias: String
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = "year"

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.N("2013")

    val request = QueryRequest {
        tableName = tableNameVal
        keyConditionExpression = "$partitionAlias = :$partitionKeyName"
        expressionAttributeNames = attrNameAlias
        this.expressionAttributeValues = attrValues
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}

suspend fun scanMovies(tableNameVal: String) {
    val request = ScanRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB getting started demo using PHP!
\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDBService();

        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );
        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
        $attributes = ["rating" =>
            [
                'AttributeName' => 'rating',
                'AttributeType' => 'N',
                'Value' => $rating,
            ],
            'plot' => [
                'AttributeName' => 'plot',
                'AttributeType' => 'S',
                'Value' => $plot,
            ]
        ];
        $service->updateItemAttributesByKey($tableName, $key, $attributes);
        echo "Movie added and updated.";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);


        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
 {$movie['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

        $movie = $service->getItemByKey($tableName, $key);
        echo "Ok, you have rated {$movie['Item']['title']['S']} as a
 {$movie['Item']['rating']['N']}\n";

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";
```

```
        echo "That's okay though. The book was better. Now, for something
lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }
        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDB テーブルをカプセル化するクラスを作成します。

```
from decimal import Decimal
from io import BytesIO
import json
import logging
import os
from pprint import pprint
```

```
import requests
from zipfile import ZipFile
import boto3
from boto3.dynamodb.conditions import Key
from botocore.exceptions import ClientError
from question import Question

logger = logging.getLogger(__name__)

class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data."""

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            self.table = table
        return exists


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                ProvisionedThroughput={
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10,
                },
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

サンプルの JSON ファイルをダウンロードして抽出するヘルパー関数を作成します。

```
def get_sample_movie_data(movie_file_name):
    """
    Gets sample movie data, either from a local file or by first downloading it
 from
    the Amazon DynamoDB developer guide.

    :param movie_file_name: The local file name where the movie data is stored in
 JSON format.
    :return: The movie data as a dict.
    """
    if not os.path.isfile(movie_file_name):
        print(f"Downloading {movie_file_name}...")
        movie_content = requests.get(
            "https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip"
        )
        movie_zip = ZipFile(BytesIO(movie_content.content))
        movie_zip.extractall()

    try:
        with open(movie_file_name) as movie_file:
            movie_data = json.load(movie_file, parse_float=Decimal)
    except FileNotFoundError:
        print(
            f"File {movie_file_name} not found. You must first download the file
 to "
            "run this demo. See the README for instructions."
        )
        raise
    else:
        # The sample file lists over 4000 movies, return only the first 250.
        return movie_data[:250]
```

対話型シナリオを実行してテーブルを作成し、そのテーブルに対してアクションを実行します。

```
def run_scenario(table_name, movie_file_name, dyn_resource):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB getting started demo.")
    print("-" * 88)

    movies = Movies(dyn_resource)
    movies_exists = movies.exists(table_name)
    if not movies_exists:
        print(f"\nCreating table {table_name}...")
        movies.create_table(table_name)
        print(f"\nCreated table {movies.table.name}.")

    my_movie = Question.ask_questions(
        [
            Question(
                "title", "Enter the title of a movie you want to add to the
 table: "
            ),
            Question("year", "What year was it released? ", Question.is_int),
            Question(
                "rating",
                "On a scale of 1 - 10, how do you rate it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question("plot", "Summarize the plot for me: "),
        ]
    )
    movies.add_movie(**my_movie)
    print(f"\nAdded '{my_movie['title']}' to '{movies.table.name}'.")
    print("-" * 88)
    movie_update = Question.ask_questions(
        [
            Question(
                "rating",
                f"\nLet's update your movie.\nYou rated it {my_movie['rating']},
what new "
                f"rating would you give it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question(
                "plot",
                f"You summarized the plot as '{my_movie['plot']}'.\nWhat would
you say now? ",
            ),
        ]
    )
    my_movie.update(movie_update)
    updated = movies.update_movie(**my_movie)
    print(f"\nUpdated '{my_movie['title']}' with new attributes:")
    pprint(updated)
    print("-" * 88)

    if not movies_exists:
        movie_data = get_sample_movie_data(movie_file_name)
        print(f"\nReading data from '{movie_file_name}' into your table.")
        movies.write_batch(movie_data)
        print(f"\nWrote {len(movie_data)} movies into {movies.table.name}.")
    print("-" * 88)

    title = "The Lord of the Rings: The Fellowship of the Ring"
    if Question.ask_question(
        f"Let's move on...do you want to get info about '{title}'? (y/n) ",
        Question.is_yesno,
    ):
        movie = movies.get_movie(title, 2001)
        print("\nHere's what I found:")
        pprint(movie)
    print("-" * 88)

    ask_for_year = True
    while ask_for_year:
        release_year = Question.ask_question(
            f"\nLet's get a list of movies released in a given year. Enter a year 
between "
            f"1972 and 2018: ",
            Question.is_int,
            Question.in_range(1972, 2018),
        )
        releases = movies.query_movies(release_year)
        if releases:
            print(f"There were {len(releases)} movies released in 
{release_year}:")
            for release in releases:
                print(f"\t{release['title']}")
            ask_for_year = False
        else:
            print(f"I don't know about any movies released in {release_year}!")
            ask_for_year = Question.ask_question(
                "Try another year? (y/n) ", Question.is_yesno
            )
    print("-" * 88)

    years = Question.ask_questions(
        [
            Question(
                "first",
                f"\nNow let's scan for movies released in a range of years. Enter 
a year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
            Question(
                "second",
                "Now enter another year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
        ]
    )
    releases = movies.scan_movies(years)
    if releases:
        count = Question.ask_question(
            f"\nFound {len(releases)} movies. How many do you want to see? ",
            Question.is_int,
            Question.in_range(1, len(releases)),
        )
        print(f"\nHere are your {count} movies:\n")
        pprint(releases[:count])
    else:
        print(
            f"I don't know about any movies released between {years['first']} "
            f"and {years['second']}."
        )
    print("-" * 88)

    if Question.ask_question(
        f"\nLet's remove your movie from the table. Do you want to remove "
        f"'{my_movie['title']}'? (y/n)",
        Question.is_yesno,
    ):
        movies.delete_movie(my_movie["title"], my_movie["year"])
        print(f"\nRemoved '{my_movie['title']}' from the table.")
    print("-" * 88)

    if Question.ask_question(f"\nDelete the table? (y/n) ", Question.is_yesno):
        movies.delete_table()
        print(f"Deleted {table_name}.")
    else:
        print(
            "Don't forget to delete the table when you're done or you might incur
 "
            "charges on your account."
        )

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        run_scenario(
            "doc-example-table-movies", "moviedata.json",
 boto3.resource("dynamodb")
        )
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

このシナリオでは、次のヘルパークラスを使用してコマンドプロンプトで質問します。

```
class Question:
    """
    A helper class to ask questions at a command prompt and validate and convert
    the answers.
    """

    def __init__(self, key, question, *validators):
        """
        :param key: The key that is used for storing the answer in a dict, when
                    multiple questions are asked in a set.
        :param question: The question to ask.
        :param validators: The answer is passed through the list of validators
 until
                           one fails or they all pass. Validators may also
 convert the
                           answer to another form, such as from a str to an int.
        """
        self.key = key
        self.question = question
        self.validators = Question.non_empty, *validators

    @staticmethod
    def ask_questions(questions):
        """
        Asks a set of questions and stores the answers in a dict.

        :param questions: The list of questions to ask.
        :return: A dict of answers.
        """
        answers = {}
        for question in questions:
            answers[question.key] = Question.ask_question(
                question.question, *question.validators
            )
        return answers

    @staticmethod
    def ask_question(question, *validators):
        """
        Asks a single question and validates it against a list of validators.
        When an answer fails validation, the complaint is printed and the
 question
        is asked again.
        :param question: The question to ask.
        :param validators: The list of validators that the answer must pass.
        :return: The answer, converted to its final form by the validators.
        """
        answer = None
        while answer is None:
            answer = input(question)
            for validator in validators:
                answer, complaint = validator(answer)
                if answer is None:
                    print(complaint)
                    break
        return answer

    @staticmethod
    def non_empty(answer):
        """
        Validates that the answer is not empty.
        :return: The non-empty answer, or None.
        """
        return answer if answer != "" else None, "I need an answer. Please?"

    @staticmethod
    def is_yesno(answer):
        """
        Validates a yes/no answer.
        :return: True when the answer is 'y'; otherwise, False.
        """
        return answer.lower() == "y", ""

    @staticmethod
    def is_int(answer):
        """
        Validates that the answer can be converted to an int.
        :return: The int answer; otherwise, None.
        """
        try:
            int_answer = int(answer)
        except ValueError:
            int_answer = None
        return int_answer, f"{answer} must be a valid integer."

    @staticmethod
    def is_letter(answer):
        """
        Validates that the answer is a letter.
        :return The letter answer, converted to uppercase; otherwise, None.
        """
        return (
            answer.upper() if answer.isalpha() else None,
            f"{answer} must be a single letter.",
        )

    @staticmethod
    def is_float(answer):
        """
        Validate that the answer can be converted to a float.
        :return The float answer; otherwise, None.
        """
        try:
            float_answer = float(answer)
        except ValueError:
            float_answer = None
        return float_answer, f"{answer} must be a valid float."

    @staticmethod
    def in_range(lower, upper):
        """
        Validate that the answer is within a range. The answer must be of a type
that can
        be compared to the lower and upper bounds.
        :return: The answer, if it is within the range; otherwise, None.
        """

        def _validate(answer):
            return (
                answer if lower <= answer <= upper else None,
                f"{answer} must be between {lower} and {upper}.",
            )

        return _validate
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

DynamoDB テーブルをカプセル化するクラスを作成します。

```
# Creates an Amazon DynamoDB table that can be used to store movie data.
# The table uses the release year of the movie as the partition key and the
# title as the sort key.
#
# @param table_name [String] The name of the table to create.
# @return [Aws::DynamoDB::Table] The newly created table.
def create_table(table_name)
  @table = @dynamo_resource.create_table(
    table_name: table_name,
    key_schema: [
      {attribute_name: "year", key_type: "HASH"},  # Partition key
      {attribute_name: "title", key_type: "RANGE"}  # Sort key
      ],
      attribute_definitions: [
        {attribute_name: "year", attribute_type: "N"},
        {attribute_name: "title", attribute_type: "S"}
      ],
      provisioned_throughput: {read_capacity_units: 10, write_capacity_units:
 10})
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

サンプルの JSON ファイルをダウンロードして抽出するヘルパー関数を作成します。

```
  # Gets sample movie data, either from a local file or by first downloading it
 from
  # the Amazon DynamoDB Developer Guide.
  #
  # @param movie_file_name [String] The local file name where the movie data is
 stored in JSON format.
  # @return [Hash] The movie data as a Hash.
  def fetch_movie_data(movie_file_name)
    if !File.file?(movie_file_name)
      @logger.debug("Downloading #{movie_file_name}...")
      movie_content = URI.open(
        "https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip"
      )
      movie_json = ""
      Zip::File.open_buffer(movie_content) do |zip|
        zip.each do |entry|
          movie_json = entry.get_input_stream.read
        end
      end
    else
      movie_json = File.read(movie_file_name)
    end
    movie_data = JSON.parse(movie_json)
    # The sample file lists over 4000 movies. This returns only the first 250.
    movie_data.slice(0, 250)
  rescue StandardError => e
    puts("Failure downloading movie data:\n#{e}")
    raise
  end
```

対話型シナリオを実行してテーブルを作成し、そのテーブルに対してアクションを実行します。

```
 table_name = "doc-example-table-movies-#{rand(10**4)}"
 scaffold = Scaffold.new(table_name)
 dynamodb_wrapper = DynamoDBBasics.new(table_name)

 new_step(1, "Create a new DynamoDB table if none already exists.")
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, "Add a new record to the DynamoDB table.")
 my_movie = {}
 my_movie[:title] = CLI::UI::Prompt.ask("Enter the title of a movie to add to
the table. E.g. The Matrix")
 my_movie[:year] = CLI::UI::Prompt.ask("What year was it released? E.g.
1989").to_i
 my_movie[:rating] = CLI::UI::Prompt.ask("On a scale of 1 - 10, how do you rate
it? E.g. 7").to_i
 my_movie[:plot] = CLI::UI::Prompt.ask("Enter a brief summary of the plot. E.g.
A man awakens to a new reality.")
 dynamodb_wrapper.add_item(my_movie)
 puts("\nNew record added:")
 puts JSON.pretty_generate(my_movie).green
 print "Done!\n".green

 new_step(3, "Update a record in the DynamoDB table.")
 my_movie[:rating] = CLI::UI::Prompt.ask("Let's update the movie you added with
a new rating, e.g. 3:").to_i
 response = dynamodb_wrapper.update_item(my_movie)
 puts("Updated '#{my_movie[:title]}' with new attributes:")
 puts JSON.pretty_generate(response).green
 print "Done!\n".green
```

```
  new_step(4, "Get a record from the DynamoDB table.")
  puts("Searching for #{my_movie[:title]} (#{my_movie[:year]})...")
  response = dynamodb_wrapper.get_item(my_movie[:title], my_movie[:year])
  puts JSON.pretty_generate(response).green
  print "Done!\n".green

  new_step(5, "Write a batch of items into the DynamoDB table.")
  download_file = "moviedata.json"
  puts("Downloading movie database to #{download_file}...")
  movie_data = scaffold.fetch_movie_data(download_file)
  puts("Writing movie data from #{download_file} into your table...")
  scaffold.write_batch(movie_data)
  puts("Records added: #{movie_data.length}.")
  print "Done!\n".green

  new_step(5, "Query for a batch of items by key.")
  loop do
    release_year = CLI::UI::Prompt.ask("Enter a year between 1972 and 2018, e.g.
1999:").to_i
    results = dynamodb_wrapper.query_items(release_year)
    if results.any?
      puts("There were #{results.length} movies released in #{release_year}:")
      results.each do |movie|
        print "\t #{movie["title"]}".green
      end
      break
    else
      continue = CLI::UI::Prompt.ask("Found no movies released in
#{release_year}! Try another year? (y/n)")
      break if !continue.eql?("y")
    end
  end
  print "\nDone!\n".green

  new_step(6, "Scan for a batch of items using a filter expression.")
  years = {}
  years[:start] = CLI::UI::Prompt.ask("Enter a starting year between 1972 and
2018:")
  years[:end] = CLI::UI::Prompt.ask("Enter an ending year between 1972 and
2018:")
  releases = dynamodb_wrapper.scan_items(years)
  if !releases.empty?
    puts("Found #{releases.length} movies.")
    count = Question.ask(
      "How many do you want to see? ", method(:is_int), in_range(1,
 releases.length))
    puts("Here are your #{count} movies:")
    releases.take(count).each do |release|
      puts("\t#{release["title"]}")
    end
  else
    puts("I don't know about any movies released between #{years[:start]} "\
         "and #{years[:end]}.")
  end
  print "\nDone!\n".green

  new_step(7, "Delete an item from the DynamoDB table.")
  answer = CLI::UI::Prompt.ask("Do you want to remove '#{my_movie[:title]}'? (y/
n) ")
  if answer.eql?("y")
    dynamodb_wrapper.delete_item(my_movie[:title], my_movie[:year])
    puts("Removed '#{my_movie[:title]}' from the table.")
    print "\nDone!\n".green
  end

  new_step(8, "Delete the DynamoDB table.")
  answer = CLI::UI::Prompt.ask("Delete the table? (y/n)")
  if answer.eql?("y")
    scaffold.delete_table
    puts("Deleted #{table_name}.")
  else
    puts("Don't forget to delete the table when you're done!")
  end
  print "\nThanks for watching!\n".green
rescue Aws::Errors::ServiceError
  puts("Something went wrong with the demo.")
rescue Errno::ENOENT
  true
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem

- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    " Create an Amazon Dynamo DB table.

    TRY.
        DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
        DATA(lo_dyn) = /aws1/cl_dyn_factory=>create( lo_session ).
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        DATA(oo_result) = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
      " It throws exception if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.

    " Describe table
    TRY.
        DATA(lo_table) = lo_dyn->describetable( iv_tablename = iv_table_name ).
        DATA(lv_tablename) = lo_table->get_table( )->ask_tablename( ).
        MESSAGE 'The table name is ' && lv_tablename TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table does not exist' TYPE 'E'.
    ENDTRY.

    " Put items into the table.
    TRY.
        DATA(lo_resp_putitem) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1975' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.5' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s = 'Star
 Wars' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1978' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '8.1' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Speed' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1994' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.9' }| ) ) )
          ) ).
        " TYPE REF TO ZCL_AWS1_dyn_PUT_ITEM_OUTPUT
        MESSAGE '3 rows inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Get item from table.
    TRY.
        DATA(lo_resp_getitem) = lo_dyn->getitem(
          iv_tablename                = iv_table_name
          it_key                      = VALUE /aws1/cl_dynattributevalue=>tt_key(
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ) ).
        DATA(lt_attr) = lo_resp_getitem->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'year' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Query item from table.
    TRY.
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
              ( NEW /aws1/cl_dynattributevalue( iv_n = '1975' ) ) ).
        DATA(lt_keyconditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
            it_attributevaluelist = lt_attributelist
            iv_comparisonoperator = |EQ|
          ) ) ) ).
        DATA(lo_query_result) = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_keyconditions ).
        DATA(lt_items) = lo_query_result->get_items( ).
        READ TABLE lo_query_result->get_items( ) INTO DATA(lt_item) INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
    " Scan items from table.
    TRY.
        DATA(lo_scan_result) = lo_dyn->scan( iv_tablename = iv_table_name ).
        lt_items = lo_scan_result->get_items( ).
        " Read the first item and display the attributes.
        READ TABLE lo_query_result->get_items( ) INTO lt_item INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Update items from table.
    TRY.
        DATA(lt_attributeupdates) = VALUE /aws1/
cl_dynattrvalueupdate=>tt_attributeupdates(
          ( VALUE /aws1/cl_dynattrvalueupdate=>ts_attributeupdates_maprow(
          key = 'rating' value = NEW /aws1/cl_dynattrvalueupdate(
            io_value  = NEW /aws1/cl_dynattributevalue( iv_n = '7.6' )
            iv_action = |PUT| ) ) ) ).
        DATA(lt_key) = VALUE /aws1/cl_dynattributevalue=>tt_key(
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 '1980' ) ) ) ).
        DATA(lo_resp) = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = lt_key
          it_attributeupdates = lt_attributeupdates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
    " Delete table.
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファァレンス」の以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Swift

SDK for Swift

> Note
>
> これはプレビューリリースの SDK に関するプレリリースドキュメントです。このドキュメントは変更される可能性があります。

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

SDK for Swift への DynamoDB 呼び出しを処理する Swift クラス。

```
import Foundation
import AWSDynamoDB

/// An enumeration of error codes representing issues that can arise when using
/// the `MovieTable` class.
enum MoviesError: Error {
    /// The specified table wasn't found or couldn't be created.
    case TableNotFound
    /// The specified item wasn't found or couldn't be created.
    case ItemNotFound
    /// The Amazon DynamoDB client is not properly initialized.
    case UninitializedClient
    /// The table status reported by Amazon DynamoDB is not recognized.
    case StatusUnknown
    /// One or more specified attribute values are invalid or missing.
    case InvalidAttributes
}



/// A class representing an Amazon DynamoDB table containing movie
/// information.
public class MovieTable {
    var ddbClient: DynamoDBClient? = nil
    let tableName: String

    /// Create an object representing a movie table in an Amazon DynamoDB
    /// database.
    ///
    /// - Parameters:
    ///   - region: The Amazon Region to create the database in.
    ///   - tableName: The name to assign to the table. If not specified, a
    ///     random table name is generated automatically.
    ///
    /// > Note: The table is not necessarily available when this function
    /// returns. Use `tableExists()` to check for its availability, or
```

```
   /// `awaitTableActive()` to wait until the table's status is reported as
   /// ready to use by Amazon DynamoDB.
   ///
   init(region: String = "us-east-2", tableName: String) async throws {
       ddbClient = try DynamoDBClient(region: region)
       self.tableName = tableName

       try await self.createTable()
   }

   ///
   /// Create a movie table in the Amazon DynamoDB data store.
   ///
   private func createTable() async throws {
       guard let client = self.ddbClient else {
           throw MoviesError.UninitializedClient
       }

       let input = CreateTableInput(
           attributeDefinitions: [
               DynamoDBClientTypes.AttributeDefinition(attributeName: "year",
attributeType: .n),
               DynamoDBClientTypes.AttributeDefinition(attributeName: "title",
attributeType: .s),
           ],
           keySchema: [
               DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
               DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
           ],
           provisionedThroughput: DynamoDBClientTypes.ProvisionedThroughput(
               readCapacityUnits: 10,
               writeCapacityUnits: 10
           ),
           tableName: self.tableName
       )
       let output = try await client.createTable(input: input)
       if output.tableDescription == nil {
           throw MoviesError.TableNotFound
       }
   }

   /// Check to see if the table exists online yet.
```

```
    ///
    /// - Returns: `true` if the table exists, or `false` if not.
    ///
    func tableExists() async throws -> Bool {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = DescribeTableInput(
            tableName: tableName
        )
        let output = try await client.describeTable(input: input)
        guard let description = output.table else {
            throw MoviesError.TableNotFound
        }

        return (description.tableName == self.tableName)
    }

    ///
    /// Waits for the table to exist and for its status to be active.
    ///
    func awaitTableActive() async throws {
        while (try await tableExists() == false) {
            Thread.sleep(forTimeInterval: 0.25)
        }

        while (try await getTableStatus() != .active) {
            Thread.sleep(forTimeInterval: 0.25)
        }
    }

    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = DeleteTableInput(
            tableName: self.tableName
        )
        _ = try await client.deleteTable(input: input)
    }

    /// Get the table's status.
    ///
    /// - Returns: The table status, as defined by the
    ///   `DynamoDBClientTypes.TableStatus` enum.
    ///
    func getTableStatus() async throws -> DynamoDBClientTypes.TableStatus {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = DescribeTableInput(
            tableName: self.tableName
        )
        let output = try await client.describeTable(input: input)
        guard let description = output.table else {
            throw MoviesError.TableNotFound
        }
        guard let status = description.tableStatus else {
            throw MoviesError.StatusUnknown
        }
        return status
    }

    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        // Create a Swift `URL` and use it to load the file into a `Data`
        // object. Then decode the JSON into an array of `Movie` objects.

        let fileUrl = URL(fileURLWithPath: jsonPath)
        let jsonData = try Data(contentsOf: fileUrl)

        var movieList = try JSONDecoder().decode([Movie].self, from: jsonData)

        // Truncate the list to the first 200 entries or so for this example.
        if movieList.count > 200 {
            movieList = Array(movieList[...199])
        }

        // Before sending records to the database, break the movie list into
        // 25-entry chunks, which is the maximum size of a batch item request.

        let count = movieList.count
        let chunks = stride(from: 0, to: count, by: 25).map {
            Array(movieList[$0 ..< Swift.min($0 + 25, count)])
        }

        // For each chunk, create a list of write request records and populate
        // them with `PutRequest` requests, each specifying one movie from the
        // chunk. Once the chunk's items are all in the `PutRequest` list,
        // send them to Amazon DynamoDB using the
        // `DynamoDBClient.batchWriteItem()` function.

        for chunk in chunks {
            var requestList: [DynamoDBClientTypes.WriteRequest] = []

            for movie in chunk {
                let item = try await movie.getAsItem()
                let request = DynamoDBClientTypes.WriteRequest(
                    putRequest: .init(
                        item: item
                    )
                )
                requestList.append(request)
            }

            let input = BatchWriteItemInput(requestItems: [tableName:
 requestList])
            _ = try await client.batchWriteItem(input: input)
        }
    }

    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        // Get a DynamoDB item containing the movie data.
        let item = try await movie.getAsItem()

        // Send the `PutItem` request to Amazon DynamoDB.

        let input = PutItemInput(
            item: item,
            tableName: self.tableName
        )
        _ = try await client.putItem(input: input)
    }

    /// Given a movie's details, add a movie to the Amazon DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title as a `String`.
    ///   - year: The release year of the movie (`Int`).
    ///   - rating: The movie's rating if available (`Double`; default is
    ///     `nil`).
    ///   - plot: A summary of the movie's plot (`String`; default is `nil`,
    ///     indicating no plot summary is available).
    ///
    func add(title: String, year: Int, rating: Double? = nil,
             plot: String? = nil) async throws {
        let movie = Movie(title: title, year: year, rating: rating, plot: plot)
        try await self.add(movie: movie)
    }

    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = GetItemInput(
            key: [
                "year": .n(String(year)),
                "title": .s(title)
            ],
            tableName: self.tableName
        )
        let output = try await client.getItem(input: input)
        guard let item = output.item else {
            throw MoviesError.ItemNotFound
        }

        let movie = try Movie(withItem: item)
        return movie
    }

    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = QueryInput(
            expressionAttributeNames: [
                "#y": "year"
            ],
            expressionAttributeValues: [
                ":y": .n(String(year))
            ],
            keyConditionExpression: "#y = :y",
            tableName: self.tableName
        )
        let output = try await client.query(input: input)

        guard let items = output.items else {
            throw MoviesError.ItemNotFound
        }
```

```
       // Convert the found movies into `Movie` objects and return an array
       // of them.

       var movieList: [Movie] = []
       for item in items {
           let movie = try Movie(withItem: item)
           movieList.append(movie)
       }
       return movieList
   }

   /// Return an array of `Movie` objects released in the specified range of
   /// years.
   ///
   /// - Parameters:
   ///   - firstYear: The first year of movies to return.
   ///   - lastYear: The last year of movies to return.
   ///   - startKey: A starting point to resume processing; always use `nil`.
   ///
   /// - Returns: An array of `Movie` objects describing the matching movies.
   ///
   /// > Note: The `startKey` parameter is used by this function when
   ///   recursively calling itself, and should always be `nil` when calling
   ///   directly.
   ///
   func getMovies(firstYear: Int, lastYear: Int,
                  startKey: [Swift.String:DynamoDBClientTypes.AttributeValue]? =
nil)
                  async throws -> [Movie] {
       var movieList: [Movie] = []

       guard let client = self.ddbClient else {
           throw MoviesError.UninitializedClient
       }

       let input = ScanInput(
           consistentRead: true,
           exclusiveStartKey: startKey,
           expressionAttributeNames: [
               "#y": "year"            // `year` is a reserved word, so use `#y`
instead.
           ],
           expressionAttributeValues: [
               ":y1": .n(String(firstYear)),
               ":y2": .n(String(lastYear))
           ],
           filterExpression: "#y BETWEEN :y1 AND :y2",
           tableName: self.tableName
       )

       let output = try await client.scan(input: input)

       guard let items = output.items else {
           return movieList
       }

       // Build an array of `Movie` objects for the returned items.

       for item in items {
           let movie = try Movie(withItem: item)
           movieList.append(movie)
       }

       // Call this function recursively to continue collecting matching
       // movies, if necessary.

       if output.lastEvaluatedKey != nil {
           let movies = try await self.getMovies(firstYear: firstYear, lastYear: 
lastYear,
                        startKey: output.lastEvaluatedKey)
           movieList += movies
       }
       return movieList
   }

   /// Update the specified movie with new `rating` and `plot` information.
   ///
   /// - Parameters:
   ///   - title: The title of the movie to update.
   ///   - year: The release year of the movie to update.
   ///   - rating: The new rating for the movie.
   ///   - plot: The new plot summary string for the movie.
   ///
   /// - Returns: An array of mappings of attribute names to their new
   ///   listing each item actually changed. Items that didn't need to change
   ///   aren't included in this list. `nil` if no changes were made.
   ///
```

```
   func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
               -> [Swift.String:DynamoDBClientTypes.AttributeValue]? {
       guard let client = self.ddbClient else {
           throw MoviesError.UninitializedClient
       }

       // Build the update expression and the list of expression attribute
       // values. Include only the information that's changed.

       var expressionParts: [String] = []
       var attrValues: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]

       if rating != nil {
           expressionParts.append("info.rating=:r")
           attrValues[":r"] = .n(String(rating!))
       }
       if plot != nil {
           expressionParts.append("info.plot=:p")
           attrValues[":p"] = .s(plot!)
       }
       let expression: String = "set \(expressionParts.joined(separator: ", "))"

       let input = UpdateItemInput(
           // Create substitution tokens for the attribute values, to ensure
           // no conflicts in expression syntax.
           expressionAttributeValues: attrValues,
           // The key identifying the movie to update consists of the release
           // year and title.
           key: [
               "year": .n(String(year)),
               "title": .s(title)
           ],
           returnValues: .updatedNew,
           tableName: self.tableName,
           updateExpression: expression
       )
       let output = try await client.updateItem(input: input)

       guard let attributes: [Swift.String:DynamoDBClientTypes.AttributeValue] =
output.attributes else {
           throw MoviesError.InvalidAttributes
       }
       return attributes
```

```
    }

    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        guard let client = self.ddbClient else {
            throw MoviesError.UninitializedClient
        }

        let input = DeleteItemInput(
            key: [
                "year": .n(String(year)),
                "title": .s(title)
            ],
            tableName: self.tableName
        )
        _ = try await client.deleteItem(input: input)
    }
}
```

MovieTable クラスがムービーを表現するために使用する構造。

```
import Foundation
import AWSDynamoDB

/// The optional details about a movie.
public struct Details: Codable {
    /// The movie's rating, if available.
    var rating: Double?
    /// The movie's plot, if available.
    var plot: String?
}

/// A structure describing a movie. The `year` and `title` properties are
/// required and are used as the key for Amazon DynamoDB operations. The
/// `info` sub-structure's two properties, `rating` and `plot`, are optional.
public struct Movie: Codable {
    /// The year in which the movie was released.
    var year: Int
    /// The movie's title.
    var title: String
    /// A `Details` object providing the optional movie rating and plot
    /// information.
    var info: Details

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - rating: The movie's rating (optional `Double`).
    ///   - plot: The movie's plot (optional `String`)
    init(title: String, year: Int, rating: Double? = nil, plot: String? = nil) {
        self.title = title
        self.year = year

        self.info = Details(rating: rating, plot: plot)
    }

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - info: The optional rating and plot information for the movie in a
    ///     `Details` object.
    init(title: String, year: Int, info: Details?){
        self.title = title
        self.year = year

        if info != nil {
            self.info = info!
        } else {
            self.info = Details(rating: nil, plot: nil)
        }
    }

    ///
    /// Return a new `MovieTable` object, given an array mapping string to Amazon
    /// DynamoDB attribute values.
    ///
    /// - Parameter item: The item information provided to the form used by
    ///   DynamoDB. This is an array of strings mapped to
    ///   `DynamoDBClientTypes.AttributeValue` values.
    init(withItem item: [Swift.String:DynamoDBClientTypes.AttributeValue]) throws
  {
        // Read the attributes.

        guard let titleAttr = item["title"],
              let yearAttr = item["year"] else {
            throw MoviesError.ItemNotFound
        }
        let infoAttr = item["info"] ?? nil

        // Extract the values of the title and year attributes.

        if case .s(let titleVal) = titleAttr {
            self.title = titleVal
        } else {
            throw MoviesError.InvalidAttributes
        }

        if case .n(let yearVal) = yearAttr {
            self.year = Int(yearVal)!
        } else {
            throw MoviesError.InvalidAttributes
        }

        // Extract the rating and/or plot from the `info` attribute, if
        // they're present.

        var rating: Double? = nil
        var plot: String? = nil

        if infoAttr != nil, case .m(let infoVal) = infoAttr {
            let ratingAttr = infoVal["rating"] ?? nil
            let plotAttr = infoVal["plot"] ?? nil

            if ratingAttr != nil, case .n(let ratingVal) = ratingAttr {
                rating = Double(ratingVal) ?? nil
            }
            if plotAttr != nil, case .s(let plotVal) = plotAttr {
                plot = plotVal
            }
        }

        self.info = Details(rating: rating, plot: plot)
    }

    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
}
```

MovieTable クラスを使用して DynamoDB データベースにアクセスするプログラム。

```
import Foundation
import ArgumentParser
import AWSDynamoDB
import ClientRuntime

@testable import MovieList

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The path of the sample movie data JSON file.")
    var jsonPath: String = "../../../../resources/sample_files/movies.json"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion = "us-east-2"

    @Option(
        help: ArgumentHelp("The level of logging for the Swift SDK to perform."),
        completion: .list([
            "critical",
            "debug",
            "error",
            "info",
            "notice",
            "trace",
            "warning"
        ])
    )
    var logLevel: String = "error"

    /// Configuration details for the command.
    static var configuration = CommandConfiguration(
        commandName: "basics",
        abstract: "A basic scenario demonstrating the usage of Amazon DynamoDB.",
        discussion: """
        An example showing how to use Amazon DynamoDB to perform a series of
        common database activities on a simple movie database.
        """
    )

    /// Called by ``main()`` to asynchronously run the AWS example.
    func runAsync() async throws {
        print("Welcome to the AWS SDK for Swift basic scenario for Amazon
 DynamoDB!")
        SDKLoggingSystem.initialize(logLevel: .error)
```

```
       //======================================================================
       // 1. Create the table. The Amazon DynamoDB table is represented by
       //    the `MovieTable` class.
       //======================================================================

       let tableName = "ddb-movies-sample-\(Int.random(in: 1...Int.max))"
       //let tableName = String.uniqueName(withPrefix: "ddb-movies-sample", 
maxDigits: 8)

       print("Creating table \"\(tableName)\"...")

       let movieDatabase = try await MovieTable(region: awsRegion,
                           tableName: tableName)

       print("\nWaiting for table to be ready to use...")
       try await movieDatabase.awaitTableActive()

       //======================================================================
       // 2. Add a movie to the table.
       //======================================================================

       print("\nAdding a movie...")
       try await movieDatabase.add(title: "Avatar: The Way of Water", year: 
2022)
       try await movieDatabase.add(title: "Not a Real Movie", year: 2023)

       //======================================================================
       // 3. Update the plot and rating of the movie using an update
       //    expression.
       //======================================================================

       print("\nAdding details to the added movie...")
       _ = try await movieDatabase.update(title: "Avatar: The Way of Water", 
year: 2022,
                   rating: 9.2, plot: "It's a sequel.")

       //======================================================================
       // 4. Populate the table from the JSON file.
       //======================================================================

       print("\nPopulating the movie database from JSON...")
       try await movieDatabase.populate(jsonPath: jsonPath)
```

```
        //=====================================================================
        // 5. Get a specific movie by key. In this example, the key is a
        //    combination of `title` and `year`.
        //=====================================================================

        print("\nLooking for a movie in the table...")
        let gotMovie = try await movieDatabase.get(title: "This Is the End",
year: 2013)

        print("Found the movie \"\(gotMovie.title)\", released in
\(gotMovie.year).")
        print("Rating: \(gotMovie.info.rating ?? 0.0).")
        print("Plot summary: \(gotMovie.info.plot ?? "None.")")

        //=====================================================================
        // 6. Delete a movie.
        //=====================================================================

        print("\nDeleting the added movie...")
        try await movieDatabase.delete(title: "Avatar: The Way of Water", year:
2022)

        //=====================================================================
        // 7. Use a query with a key condition expression to return all movies
        //    released in a given year.
        //=====================================================================

        print("\nGetting movies released in 1994...")
        let movieList = try await movieDatabase.getMovies(fromYear: 1994)
        for movie in movieList {
            print("    \(movie.title)")
        }

        //=====================================================================
        // 8. Use `scan()` to return movies released in a range of years.
        //=====================================================================

        print("\nGetting movies released between 1993 and 1997...")
        let scannedMovies = try await movieDatabase.getMovies(firstYear: 1993,
lastYear: 1997)
        for movie in scannedMovies {
            print("    \(movie.title) (\(movie.year))")
        }
        //======================================================================
        // 9. Delete the table.
        //======================================================================

        print("\nDeleting the table...")
        try await movieDatabase.deleteTable()
    }
}

@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API の詳細については、AWS SDK for Swift API リファレンスの以下のトピックを参照してください。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

PartiQL ステートメントのバッチと AWS SDK を使用して DynamoDB テーブルにクエリを実行する

次のコード例は、以下を実行する方法を示しています。

- 複数の SELECT ステートメントを実行して、項目のバッチを取得する。
- 複数の INSERT ステートメントを実行して、項目のバッチを追加する。
- 複数の UPDATE ステートメントを実行して、項目のバッチを更新する。
- 複数の DELETE ステートメントを実行して、項目のバッチを削除する。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = "moviedata.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");
```

```
var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();
```

```
// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1,
 year1, producer2, title2, year2);
if (success)
{
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
```

```
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT
 statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch
 method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting
 the table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}


        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT FROM {tableName} WHERE title = ? AND year
 = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
 movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
 {{'title': ?, 'year': ?}}";
```

```
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }
            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }

        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「BatchExecuteStatement」を参照してください。

C++

SDK for C++

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlBatchExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using PartiQL batch statements.
```

```
/*!
  \sa partiqlBatchExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::partiqlBatchExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {

    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
```

```
                 << INFO_KEY << "': ?}";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);

           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

           // Create attribute for the info map.
           Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

           std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                   ALLOCATION_TAG.c_str());
           ratingAttribute->SetN(ratings[i]);
           infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

           std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                   ALLOCATION_TAG.c_str());
           plotAttribute->SetS(plots[i]);
           infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
           attributes.push_back(infoMapAttribute);
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);

       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }
```

```
   }

   std::cout << "Retrieving the movie data with a batch \"SELECT\" statement."
             << std::endl;

   // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);
           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);

       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (outcome.IsSuccess()) {
           const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

           for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
               const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();
```

```
               printMovieInfo(item);
           }
       }
       else {
           std::cerr << "Failed to retrieve the movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
   }

   // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

   for (size_t i = 0; i < titles.size(); ++i) {
       ratings[i] = askQuestionForFloatRange(
               Aws::String("\nLet's update your the movie, \"") + titles[i] +
               ".\nYou rated it  " + std::to_string(ratings[i])
               + ", what new rating would you give it? ", 1, 10);
   }

   std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());

       std::stringstream sqlStream;
       sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);

           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));
```

```
attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

       request.SetStatements(statements);
       Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
               request);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to update movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
   }

   std::cout << "Retrieving the updated movie data with a batch \"SELECT\"
statement."
             << std::endl;

   // 5. Get the updated data for multiple movies using "Select" statements.
(BatchExecuteStatement)
   {
       Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
               titles.size());
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       std::string sql(sqlStream.str());

       for (size_t i = 0; i < statements.size(); ++i) {
           statements[i].SetStatement(sql);
           Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
           attributes.push_back(
                   Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
           statements[i].SetParameters(attributes);
       }

       Aws::DynamoDB::Model::BatchExecuteStatementRequest request;
```

```
        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movies information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Deleting the movie data with a batch \"DELETE\" statement."
              << std::endl;

    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

 attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
 std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
 dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }
```

```
    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「BatchExecuteStatement」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルを作成し、PartiQL クエリのバッチを実行するシナリオを実行します。

```
// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(sdkConfig aws.Config, tableName string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists()
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable()
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
 log.Println(strings.Repeat("-", 88))

 currentYear, _, _ := time.Now().Date()
 customMovies := []actions.Movie{{
  Title: "House PartiQL",
  Year:  currentYear - 5,
  Info: map[string]interface{}{
   "plot":   "Wacky high jinks result from querying a mysterious database.",
   "rating": 8.5}}, {
  Title: "House PartiQL 2",
  Year:  currentYear - 3,
  Info: map[string]interface{}{
   "plot":   "Moderate high jinks result from querying another mysterious
 database.",
   "rating": 6.5}}, {
  Title: "House PartiQL 3",
  Year:  currentYear - 1,
  Info: map[string]interface{}{
   "plot":   "Tepid high jinks result from querying yet another mysterious
 database.",
   "rating": 2.5},
 },
 }
```

```
log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))

newRatings := []float64{7.7, 4.4, 1.1}
log.Println("Updating a batch of movies with new ratings.")
err = runner.UpdateMovieBatch(customMovies, newRatings)
if err == nil {
 log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting projected data from the table to verify our update.")
log.Println("Using a page size of 2 to demonstrate paging.")
projections, err := runner.GetAllMovies(2)
if err == nil {
 log.Println("All movies:")
 for _, projection := range projections {
  log.Println(projection)
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Deleting a batch of movies.")
err = runner.DeleteMovieBatch(customMovies)
if err == nil {
 log.Printf("Deleted %v movies.\n", len(customMovies))
}

err = tableBasics.DeleteTable()
if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

この例で使用している Movie struct を定義します。

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

PartiQL ステートメントを実行する struct およびメソッドを作成します。

```
// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(movies []Movie) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(movies []Movie) ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(movies []Movie, ratings []float64)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(movies []Movie) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(context.TODO(),
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「BatchExecuteStatement」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
public class ScenarioPartiQLBatch {
        public static void main(String[] args) throws IOException {
                String tableName = "MoviesPartiQBatch";
                Region region = Region.US_EAST_1;
                DynamoDbClient ddb = DynamoDbClient.builder()
                                .region(region)
                                .build();
                System.out.println("******* Creating an Amazon DynamoDB table 
named " + tableName
                                + " with a key named year and a sort key named 
title.");
                createTable(ddb, tableName);

                System.out.println("******* Adding multiple records into the " + 
tableName
                                + " table using a batch command.");
                putRecordBatch(ddb);

                System.out.println("******* Updating multiple records using a 
batch command.");
                updateTableItemBatch(ddb);

                System.out.println("******* Deleting multiple records using a 
batch command.");
                deleteItemBatch(ddb);

                System.out.println("******* Deleting the Amazon DynamoDB 
table.");
                deleteDynamoDBTable(ddb, tableName);
                ddb.close();
        }

        public static void createTable(DynamoDbClient ddb, String tableName) {
                DynamoDbWaiter dbWaiter = ddb.waiter();
                ArrayList<AttributeDefinition> attributeDefinitions = new 
ArrayList<>();

                // Define attributes.
                attributeDefinitions.add(AttributeDefinition.builder()
                                .attributeName("year")
                                .attributeType("N")
                                .build());

                attributeDefinitions.add(AttributeDefinition.builder()
                                .attributeName("title")
                                .attributeType("S")
                                .build());

                ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
                KeySchemaElement key = KeySchemaElement.builder()
```

```
                               .attributeName("year")
                               .keyType(KeyType.HASH)
                               .build();

               KeySchemaElement key2 = KeySchemaElement.builder()
                               .attributeName("title")
                               .keyType(KeyType.RANGE) // Sort
                               .build();

               // Add KeySchemaElement objects to the list.
               tableKey.add(key);
               tableKey.add(key2);

               CreateTableRequest request = CreateTableRequest.builder()
                               .keySchema(tableKey)

.provisionedThroughput(ProvisionedThroughput.builder()
                                               .readCapacityUnits(new Long(10))
                                               .writeCapacityUnits(new Long(10))
                                               .build())
                               .attributeDefinitions(attributeDefinitions)
                               .tableName(tableName)
                               .build();

               try {
                       CreateTableResponse response = ddb.createTable(request);
                       DescribeTableRequest tableRequest =
DescribeTableRequest.builder()
                                       .tableName(tableName)
                                       .build();

                       // Wait until the Amazon DynamoDB table is created.
                       WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter
                                       .waitUntilTableExists(tableRequest);

waiterResponse.matched().response().ifPresent(System.out::println);
                       String newTable =
response.tableDescription().tableName();
                       System.out.println("The " + newTable + " was successfully
created.");

               } catch (DynamoDbException e) {
                       System.err.println(e.getMessage());
```

```
                        System.exit(1);
                }
        }

        public static void putRecordBatch(DynamoDbClient ddb) {
                String sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE
{'year':?, 'title' : ?, 'info' : ?}";
                try {
                        // Create three movies to add to the Amazon DynamoDB
table.
                        // Set data for Movie 1.
                        List<AttributeValue> parameters = new ArrayList<>();

                        AttributeValue att1 = AttributeValue.builder()
                                        .n(String.valueOf("2022"))
                                        .build();

                        AttributeValue att2 = AttributeValue.builder()
                                        .s("My Movie 1")
                                        .build();

                        AttributeValue att3 = AttributeValue.builder()
                                        .s("No Information")
                                        .build();

                        parameters.add(att1);
                        parameters.add(att2);
                        parameters.add(att3);

                        BatchStatementRequest statementRequestMovie1 =
BatchStatementRequest.builder()
                                        .statement(sqlStatement)
                                        .parameters(parameters)
                                        .build();

                        // Set data for Movie 2.
                        List<AttributeValue> parametersMovie2 = new
ArrayList<>();
                        AttributeValue attMovie2 = AttributeValue.builder()
                                        .n(String.valueOf("2022"))
                                        .build();

                        AttributeValue attMovie2A = AttributeValue.builder()
                                        .s("My Movie 2")
```

```
                                       .build();

                       AttributeValue attMovie2B = AttributeValue.builder()
                                       .s("No Information")
                                       .build();

                       parametersMovie2.add(attMovie2);
                       parametersMovie2.add(attMovie2A);
                       parametersMovie2.add(attMovie2B);

                       BatchStatementRequest statementRequestMovie2 = 
BatchStatementRequest.builder()
                                       .statement(sqlStatement)
                                       .parameters(parametersMovie2)
                                       .build();

                       // Set data for Movie 3.
                       List<AttributeValue> parametersMovie3 = new 
ArrayList<>();
                       AttributeValue attMovie3 = AttributeValue.builder()
                                       .n(String.valueOf("2022"))
                                       .build();

                       AttributeValue attMovie3A = AttributeValue.builder()
                                       .s("My Movie 3")
                                       .build();

                       AttributeValue attMovie3B = AttributeValue.builder()
                                       .s("No Information")
                                       .build();

                       parametersMovie3.add(attMovie3);
                       parametersMovie3.add(attMovie3A);
                       parametersMovie3.add(attMovie3B);

                       BatchStatementRequest statementRequestMovie3 = 
BatchStatementRequest.builder()
                                       .statement(sqlStatement)
                                       .parameters(parametersMovie3)
                                       .build();

                       // Add all three movies to the list.
                       List<BatchStatementRequest> myBatchStatementList = new 
ArrayList<>();
```

```
                       myBatchStatementList.add(statementRequestMovie1);
                       myBatchStatementList.add(statementRequestMovie2);
                       myBatchStatementList.add(statementRequestMovie3);

                       BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
                                       .statements(myBatchStatementList)
                                       .build();

                       BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
                       System.out.println("ExecuteStatement successful: " +
response.toString());
                       System.out.println("Added new movies using a batch
command.");

               } catch (DynamoDbException e) {
                       System.err.println(e.getMessage());
                       System.exit(1);
               }
       }

       public static void updateTableItemBatch(DynamoDbClient ddb) {
               String sqlStatement = "UPDATE MoviesPartiQBatch SET info =
'directors\":[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and
title=?";
               List<AttributeValue> parametersRec1 = new ArrayList<>();

               // Update three records.
               AttributeValue att1 = AttributeValue.builder()
                               .n(String.valueOf("2022"))
                               .build();

               AttributeValue att2 = AttributeValue.builder()
                               .s("My Movie 1")
                               .build();

               parametersRec1.add(att1);
               parametersRec1.add(att2);

               BatchStatementRequest statementRequestRec1 =
BatchStatementRequest.builder()
                               .statement(sqlStatement)
                               .parameters(parametersRec1)
                               .build();

               // Update record 2.
               List<AttributeValue> parametersRec2 = new ArrayList<>();
               AttributeValue attRec2 = AttributeValue.builder()
                               .n(String.valueOf("2022"))
                               .build();

               AttributeValue attRec2a = AttributeValue.builder()
                               .s("My Movie 2")
                               .build();

               parametersRec2.add(attRec2);
               parametersRec2.add(attRec2a);
               BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
                               .statement(sqlStatement)
                               .parameters(parametersRec2)
                               .build();

               // Update record 3.
               List<AttributeValue> parametersRec3 = new ArrayList<>();
               AttributeValue attRec3 = AttributeValue.builder()
                               .n(String.valueOf("2022"))
                               .build();

               AttributeValue attRec3a = AttributeValue.builder()
                               .s("My Movie 3")
                               .build();

               parametersRec3.add(attRec3);
               parametersRec3.add(attRec3a);
               BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
                               .statement(sqlStatement)
                               .parameters(parametersRec3)
                               .build();

               // Add all three movies to the list.
               List<BatchStatementRequest> myBatchStatementList = new
ArrayList<>();
               myBatchStatementList.add(statementRequestRec1);
               myBatchStatementList.add(statementRequestRec2);
               myBatchStatementList.add(statementRequestRec3);
               BatchExecuteStatementRequest batchRequest = 
BatchExecuteStatementRequest.builder()
                               .statements(myBatchStatementList)
                               .build();

               try {
                       BatchExecuteStatementResponse response = 
ddb.batchExecuteStatement(batchRequest);
                       System.out.println("ExecuteStatement successful: " + 
response.toString());
                       System.out.println("Updated three movies using a batch 
command.");

               } catch (DynamoDbException e) {
                       System.err.println(e.getMessage());
                       System.exit(1);
               }
               System.out.println("Item was updated!");
       }

       public static void deleteItemBatch(DynamoDbClient ddb) {
               String sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year 
= ? and title=?";
               List<AttributeValue> parametersRec1 = new ArrayList<>();

               // Specify three records to delete.
               AttributeValue att1 = AttributeValue.builder()
                               .n(String.valueOf("2022"))
                               .build();

               AttributeValue att2 = AttributeValue.builder()
                               .s("My Movie 1")
                               .build();

               parametersRec1.add(att1);
               parametersRec1.add(att2);

               BatchStatementRequest statementRequestRec1 = 
BatchStatementRequest.builder()
                               .statement(sqlStatement)
                               .parameters(parametersRec1)
                               .build();
               // Specify record 2.
               List<AttributeValue> parametersRec2 = new ArrayList<>();
               AttributeValue attRec2 = AttributeValue.builder()
                               .n(String.valueOf("2022"))
                               .build();

               AttributeValue attRec2a = AttributeValue.builder()
                               .s("My Movie 2")
                               .build();

               parametersRec2.add(attRec2);
               parametersRec2.add(attRec2a);
               BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
                               .statement(sqlStatement)
                               .parameters(parametersRec2)
                               .build();

               // Specify record 3.
               List<AttributeValue> parametersRec3 = new ArrayList<>();
               AttributeValue attRec3 = AttributeValue.builder()
                               .n(String.valueOf("2022"))
                               .build();

               AttributeValue attRec3a = AttributeValue.builder()
                               .s("My Movie 3")
                               .build();

               parametersRec3.add(attRec3);
               parametersRec3.add(attRec3a);

               BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
                               .statement(sqlStatement)
                               .parameters(parametersRec3)
                               .build();

               // Add all three movies to the list.
               List<BatchStatementRequest> myBatchStatementList = new
ArrayList<>();
               myBatchStatementList.add(statementRequestRec1);
               myBatchStatementList.add(statementRequestRec2);
               myBatchStatementList.add(statementRequestRec3);
```

```
                BatchExecuteStatementRequest batchRequest =
 BatchExecuteStatementRequest.builder()
                                .statements(myBatchStatementList)
                                .build();

                try {
                        ddb.batchExecuteStatement(batchRequest);
                        System.out.println("Deleted three movies using a batch
 command.");

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
        }

        public static void deleteDynamoDBTable(DynamoDbClient ddb, String
 tableName) {
                DeleteTableRequest request = DeleteTableRequest.builder()
                                .tableName(tableName)
                                .build();

                try {
                        ddb.deleteTable(request);

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
                System.out.println(tableName + " was successfully deleted!");
        }

        private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,
                        List<AttributeValue> parameters) {
                ExecuteStatementRequest request =
 ExecuteStatementRequest.builder()
                                .statement(statement)
                                .parameters(parameters)
                                .build();

                return ddb.executeStatement(request);
        }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「BatchExecuteStatement」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

PartiQL ステートメントのバッチを実行します。

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async () => {
  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
```

```
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log(`Cities inserted.`);

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */
  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log(`Updated cities.`);

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */
```

```
    log("Deleting the table.");
    const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
    await client.send(deleteTableCommand);
    log("Table deleted.");
  };
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「BatchExecuteStatement」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun main() {
    val ddb = DynamoDbClient { region = "us-east-1" }
    val tableName = "MoviesPartiQBatch"
    println("Creating an Amazon DynamoDB table named $tableName with a key named
 id and a sort key named title.")
    createTablePartiQLBatch(ddb, tableName, "year")
    putRecordBatch(ddb)
    updateTableItemBatchBatch(ddb)
    deleteItemsBatch(ddb)
    deleteTablePartiQLBatch(tableName)
}

suspend fun createTablePartiQLBatch(ddb: DynamoDbClient, tableNameVal: String,
 key: String) {
    val attDef = AttributeDefinition {
        attributeName = key
        attributeType = ScalarAttributeType.N
    }

    val attDef1 = AttributeDefinition {
        attributeName = "title"
```

```
            attributeType = ScalarAttributeType.S
        }

        val keySchemaVal = KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

        val keySchemaVal1 = KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

        val provisionedVal = ProvisionedThroughput {
            readCapacityUnits = 10
            writeCapacityUnits = 10
        }

        val request = CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            provisionedThroughput = provisionedVal
            tableName = tableNameVal
        }

        val response = ddb.createTable(request)
        ddb.waitUntilTableExists { // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
     ${response.tableDescription?.tableArn}")
    }

    suspend fun putRecordBatch(ddb: DynamoDbClient) {
        val sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
     'title' : ?, 'info' : ?}"

        // Create three movies to add to the Amazon DynamoDB table.
        val parametersMovie1 = mutableListOf<AttributeValue>()
        parametersMovie1.add(AttributeValue.N("2022"))
        parametersMovie1.add(AttributeValue.S("My Movie 1"))
        parametersMovie1.add(AttributeValue.S("No Information"))

        val statementRequestMovie1 = BatchStatementRequest {
```

```
        statement = sqlStatement
        parameters = parametersMovie1
    }

    // Set data for Movie 2.
    val parametersMovie2 = mutableListOf<AttributeValue>()
    parametersMovie2.add(AttributeValue.N("2022"))
    parametersMovie2.add(AttributeValue.S("My Movie 2"))
    parametersMovie2.add(AttributeValue.S("No Information"))

    val statementRequestMovie2 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersMovie2
    }

    // Set data for Movie 3.
    val parametersMovie3 = mutableListOf<AttributeValue>()
    parametersMovie3.add(AttributeValue.N("2022"))
    parametersMovie3.add(AttributeValue.S("My Movie 3"))
    parametersMovie3.add(AttributeValue.S("No Information"))

    val statementRequestMovie3 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersMovie3
    }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestMovie1)
    myBatchStatementList.add(statementRequestMovie2)
    myBatchStatementList.add(statementRequestMovie3)

    val batchRequest = BatchExecuteStatementRequest {
        statements = myBatchStatementList
    }
    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: " + response.toString())
    println("Added new movies using a batch command.")
}

suspend fun updateTableItemBatchBatch(ddb: DynamoDbClient) {
    val sqlStatement =
        "UPDATE MoviesPartiQBatch SET info = 'directors\":[\"Merian C. Cooper\",
\"Ernest B. Schoedsack' where year=? and title=?"
```

```
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))
    val statementRequestRec1 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersRec1
    }

    // Update record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersRec2
    }

    // Update record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersRec3
    }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest = BatchExecuteStatementRequest {
        statements = myBatchStatementList
    }

    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: $response")
    println("Updated three movies using a batch command.")
    println("Items were updated!")
}

suspend fun deleteItemsBatch(ddb: DynamoDbClient) {
    // Specify three records to delete.
    val sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))

    val statementRequestRec1 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersRec1
    }

    // Specify record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersRec2
    }

    // Specify record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 = BatchStatementRequest {
        statement = sqlStatement
        parameters = parametersRec3
    }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest = BatchExecuteStatementRequest {
        statements = myBatchStatementList
    }

    ddb.batchExecuteStatement(batchRequest)
    println("Deleted three movies using a batch command.")
}

suspend fun deleteTablePartiQLBatch(tableNameVal: String) {
    val request = DeleteTableRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「BatchExecuteStatement」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithPartiQLBatch
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");
        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQLBatch($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);
        echo "Movie added and updated.\n";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "\nThe movie {$movie['Responses'][0]['Item']['title']['S']}
        was released in {$movie['Responses'][0]['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Responses'][0]['Item']
['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "Okay, you have rated {$movie['Responses'][0]['Item']['title']
['S']}
        as a {$movie['Responses'][0]['Item']['rating']['N']}\n";
```

```
       $service->deleteItemByPartiQLBatch($statement, $parameters);
       echo "But, bad news, this was a trap. That movie has now been deleted 
because of your rating...harsh.\n";

       echo "That's okay though. The book was better. Now, for something 
lighter, in what year were you born?\n";
       $birthYear = "not a number";
       while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
           $birthYear = testable_readline("Birth year: ");
       }
       $birthKey = [
           'Key' => [
               'year' => [
                   'N' => "$birthYear",
               ],
           ],
       ];
       $result = $service->query($tableName, $birthKey);
       $marshal = new Marshaler();
       echo "Here are the movies in our collection released the year you were 
born:\n";
       $oops = "Oops! There were no movies released in that year (that we know 
of).\n";
       $display = "";
       foreach ($result['Items'] as $movie) {
           $movie = $marshal->unmarshalItem($movie);
           $display .= $movie['title'] . "\n";
       }
       echo ($display) ?: $oops;

       $yearsKey = [
           'Key' => [
               'year' => [
                   'N' => [
                       'minRange' => 1990,
                       'maxRange' => 1999,
                   ],
               ],
           ],
       ];
       $filter = "year between 1990 and 1999";
       echo "\nHere's a list of all the movies released in the 90s:\n";
       $result = $service->scan($tableName, $yearsKey, $filter);
```

```
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「BatchExecuteStatement」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

PartiQL ステートメントのバッチを実行できるクラスを作成します。

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint
```

```
import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
 with
                           each statement. This list must be in the same order as
 the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
 table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
 %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

テーブルを作成し、PartiQL クエリをバッチで実行するシナリオを実行します。

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL batch statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    movie_data = [
        {
            "title": f"House PartiQL",
            "year": datetime.now().year - 5,
            "info": {
                "plot": "Wacky high jinks result from querying a mysterious
 database.",
                "rating": Decimal("8.5"),
            },
        },
        {
            "title": f"House PartiQL 2",
            "year": datetime.now().year - 3,
            "info": {
                "plot": "Moderate high jinks result from querying another
mysterious database.",
                "rating": Decimal("6.5"),
            },
        },
        {
            "title": f"House PartiQL 3",
            "year": datetime.now().year - 1,
            "info": {
                "plot": "Tepid high jinks result from querying yet another
mysterious database.",
                "rating": Decimal("2.5"),
            },
        },
    ]

    print(f"Inserting a batch of movies into table '{table_name}.")
    statements = [
        f'INSERT INTO "{table_name}" ' f"VALUE {{'title': ?, 'year': ?,
'info': ?}}"
    ] * len(movie_data)
    params = [list(movie.values()) for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting data for a batch of movies.")
    statements = [f'SELECT * FROM "{table_name}" WHERE title=? AND year=?'] *
len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    output = wrapper.run_partiql(statements, params)
    for item in output["Responses"]:
        print(f"\n{item['Item']['title']}, {item['Item']['year']}")
        pprint(item["Item"])
    print("-" * 88)
    ratings = [Decimal("7.7"), Decimal("5.5"), Decimal("1.3")]
    print(f"Updating a batch of movies with new ratings.")
    statements = [
        f'UPDATE "{table_name}" SET info.rating=? ' f"WHERE title=? AND year=?"
    ] * len(movie_data)
    params = [
        [rating, movie["title"], movie["year"]]
        for rating, movie in zip(ratings, movie_data)
    ]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting projected data from the table to verify our update.")
    output = wrapper.dyn_resource.meta.client.execute_statement(
        Statement=f'SELECT title, info.rating FROM "{table_name}"'
    )
    pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting a batch of movies from the table.")
    statements = [f'DELETE FROM "{table_name}" WHERE title=? AND year=?'] * len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLBatchWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「BatchExecuteStatement」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルを作成し、PartiQL クエリのバッチを実行するシナリオを実行します。

```
table_name = "doc-example-table-movies-partiql-#{rand(10**4)}"
scaffold = Scaffold.new(table_name)
sdk = DynamoDBPartiQLBatch.new(table_name)

new_step(1, "Create a new DynamoDB table if none already exists.")
unless scaffold.exists?(table_name)
  puts("\nNo such table: #{table_name}. Creating it...")
  scaffold.create_table(table_name)
  print "Done!\n".green
end

new_step(2, "Populate DynamoDB table with movie data.")
download_file = "moviedata.json"
puts("Downloading movie database to #{download_file}...")
movie_data = scaffold.fetch_movie_data(download_file)
puts("Writing movie data from #{download_file} into your table...")
scaffold.write_batch(movie_data)
puts("Records added: #{movie_data.length}.")
print "Done!\n".green

new_step(3, "Select a batch of items from the movies table.")
puts "Let's select some popular movies for side-by-side comparison."
```

```
  response = sdk.batch_execute_select([["Mean Girls", 2004], ["Goodfellas",
 1977], ["The Prancing of the Lambs", 2005]])
  puts("Items selected: #{response['responses'].length}\n")
  print "\nDone!\n".green

  new_step(4, "Delete a batch of items from the movies table.")
  sdk.batch_execute_write([["Mean Girls", 2004], ["Goodfellas", 1977], ["The
 Prancing of the Lambs", 2005]])
  print "\nDone!\n".green

  new_step(5, "Delete the table.")
  if scaffold.exists?(table_name)
    scaffold.delete_table
  end
end
```

- API の詳細については、AWS SDK for Ruby API リファレンスの「BatchExecuteStatement」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

PartiQL と AWS SDK を使用して DynamoDB テーブルに対してクエリを実行する

次のコード例は、以下を実行する方法を示しています。

- SELECT ステートメントを実行して項目を取得する。
- INSERT 文を実行して項目を追加する。
- UPDATE ステートメントを使用して項目を更新する。
- DELETE ステートメントを実行して項目を削除する。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
        private static readonly AmazonDynamoDBClient Client = new
 AmazonDynamoDBClient();



        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
 movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
```

```
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }



        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string,
 AttributeValue>> items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
                items.ForEach(item =>
 Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}




        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
```

```
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }




        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
 movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
 'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
```

```
       /// <returns>A Boolean value that indicates the success of the
       /// DELETE operation.</returns>
       public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
       {
           var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

           var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
           {
               Statement = deleteSingle,
               Parameters = new List<AttributeValue>
               {
                   new AttributeValue { S = movieTitle },
                   new AttributeValue { N = year.ToString() },
               },
           });

           return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
       }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ExecuteStatement」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
       //  1. Create a table. (CreateTable)
       if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

           AwsDoc::DynamoDB::partiqlExecuteScenario(clientConfig);
```

```
            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using single PartiQL
 statements.
/*!
  \sa partiqlExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::partiqlExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
 it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        request.SetStatement(sqlStream.str());

        // Create the parameter attributes.
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
 Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        ratingAttribute->SetN(rating);
        infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

        std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
 Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                ALLOCATION_TAG.c_str());
        plotAttribute->SetS(plot);
        infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
        attributes.push_back(infoMapAttribute);
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    //  3. Get the data for the movie using a "Select" statement.
 (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
```

```
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to retrieve movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
       else {
           // Print the retrieved movie information.
           const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

           if (items.size() == 1) {
               printMovieInfo(items[0]);
           }
           else {
               std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                         << " There should be only one movie." << std::endl;
           }
       }
   }

   //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
   {
       rating = askQuestionForFloatRange(
               Aws::String("\nLet's update your movie.\nYou rated it  ") +
               std::to_string(rating)
               + ", what new rating would you give it? ", 1, 10);

       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                 << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                 << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";
       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to update a movie: "
                     << outcome.GetError().GetMessage();
           return false;
       }
   }

   std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

   //  5. Get the updated data for the movie using a "Select" statement.
(ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to retrieve the movie information: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
       else {
           const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

           if (items.size() == 1) {
               printMovieInfo(items[0]);
           }
           else {
               std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                         << " There should be only one movie." << std::endl;
           }
       }
   }

   std::cout << "Deleting the movie" << std::endl;

   // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                 << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

       request.SetStatement(sqlStream.str());

       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to delete the movie: "
                     << outcome.GetError().GetMessage() << std::endl;
           return false;
       }
```

```
    }

    std::cout << "Movie successfully deleted." << std::endl;
    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
 std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
 dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ExecuteStatement」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルを作成し、PartiQL クエリを実行するシナリオを実行します。

```
// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(sdkConfig aws.Config, tableName string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists()
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable()
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
 log.Println(strings.Repeat("-", 88))

 currentYear, _, _ := time.Now().Date()
 customMovie := actions.Movie{
  Title: "24 Hour PartiQL People",
  Year:  currentYear,
  Info: map[string]interface{}{
   "plot":   "A group of data developers discover a new query language they can't
 stop using.",
   "rating": 9.9,
  },
 }

 log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
 customMovie.Year)
 err = runner.AddMovie(customMovie)
 if err == nil {
  log.Printf("Added %v to the movie table.\n", customMovie.Title)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
 customMovie.Year)
 movie, err := runner.GetMovie(customMovie.Title, customMovie.Year)
 if err == nil {
  log.Println(movie)
 }
 log.Println(strings.Repeat("-", 88))

 newRating := 6.6
 log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
 newRating)
 err = runner.UpdateMovie(customMovie, newRating)
 if err == nil {
  log.Printf("Updated %v with a new rating.\n", customMovie.Title)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Getting data again to verify the update.")
 movie, err = runner.GetMovie(customMovie.Title, customMovie.Year)
 if err == nil {
  log.Println(movie)
 }
```

```
 log.Println(strings.Repeat("-", 88))

 log.Printf("Deleting movie '%v'.\n", customMovie.Title)
 err = runner.DeleteMovie(customMovie)
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 err = tableBasics.DeleteTable()
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

この例で使用している Movie struct を定義します。

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

PartiQL ステートメントを実行する struct およびメソッドを作成します。

```
// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(title string, year int) (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(movie Movie, rating float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(context.TODO(),
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「ExecuteStatement」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
public class ScenarioPartiQ {
    public static void main(String[] args) throws IOException {
        final String usage = """

                Usage:
                    <fileName>

                Where:
                    fileName - The path to the moviedata.json file that you can
 download from the Amazon DynamoDB Developer Guide.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String fileName = args[0];
        String tableName = "MoviesPartiQ";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        System.out.println(
                "******* Creating an Amazon DynamoDB table named MoviesPartiQ
 with a key named year and a sort key named title.");
        createTable(ddb, tableName);

        System.out.println("******* Loading data into the MoviesPartiQ table.");
        loadData(ddb, fileName);
```

```
       System.out.println("******* Getting data from the MoviesPartiQ table.");
       getItem(ddb);

       System.out.println("******* Putting a record into the MoviesPartiQ
table.");
       putRecord(ddb);

       System.out.println("******* Updating a record.");
       updateTableItem(ddb);

       System.out.println("******* Querying the movies released in 2013.");
       queryTable(ddb);

       System.out.println("******* Deleting the Amazon DynamoDB table.");
       deleteDynamoDBTable(ddb, tableName);
       ddb.close();
   }

   public static void createTable(DynamoDbClient ddb, String tableName) {
       DynamoDbWaiter dbWaiter = ddb.waiter();
       ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

       // Define attributes.
       attributeDefinitions.add(AttributeDefinition.builder()
               .attributeName("year")
               .attributeType("N")
               .build());

       attributeDefinitions.add(AttributeDefinition.builder()
               .attributeName("title")
               .attributeType("S")
               .build());

       ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
       KeySchemaElement key = KeySchemaElement.builder()
               .attributeName("year")
               .keyType(KeyType.HASH)
               .build();

       KeySchemaElement key2 = KeySchemaElement.builder()
               .attributeName("title")
               .keyType(KeyType.RANGE) // Sort
               .build();
       // Add KeySchemaElement objects to the list.
       tableKey.add(key);
       tableKey.add(key2);

       CreateTableRequest request = CreateTableRequest.builder()
               .keySchema(tableKey)
               .provisionedThroughput(ProvisionedThroughput.builder()
                       .readCapacityUnits(new Long(10))
                       .writeCapacityUnits(new Long(10))
                       .build())
               .attributeDefinitions(attributeDefinitions)
               .tableName(tableName)
               .build();

       try {
           CreateTableResponse response = ddb.createTable(request);
           DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                   .tableName(tableName)
                   .build();

           // Wait until the Amazon DynamoDB table is created.
           WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
           waiterResponse.matched().response().ifPresent(System.out::println);
           String newTable = response.tableDescription().tableName();
           System.out.println("The " + newTable + " was successfully created.");

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   // Load data into the table.
   public static void loadData(DynamoDbClient ddb, String fileName) throws
IOException {

       String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
       JsonParser parser = new JsonFactory().createParser(new File(fileName));
       com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
       Iterator<JsonNode> iter = rootNode.iterator();
       ObjectNode currentNode;
```

```
        int t = 0;
        List<AttributeValue> parameters = new ArrayList<>();
        while (iter.hasNext()) {

            // Add 200 movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            AttributeValue att1 = AttributeValue.builder()
                    .n(String.valueOf(year))
                    .build();

            AttributeValue att2 = AttributeValue.builder()
                    .s(title)
                    .build();

            AttributeValue att3 = AttributeValue.builder()
                    .s(info)
                    .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            // Insert the movie into the Amazon DynamoDB table.
            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added Movie " + title);

            parameters.remove(att1);
            parameters.remove(att2);
            parameters.remove(att3);
            t++;
        }
    }

    public static void getItem(DynamoDbClient ddb) {

        String sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and
title=?";
```

```
       List<AttributeValue> parameters = new ArrayList<>();
       AttributeValue att1 = AttributeValue.builder()
               .n("2012")
               .build();

       AttributeValue att2 = AttributeValue.builder()
               .s("The Perks of Being a Wallflower")
               .build();

       parameters.add(att1);
       parameters.add(att2);

       try {
           ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
           System.out.println("ExecuteStatement successful: " +
response.toString());

       } catch (DynamoDbException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   public static void putRecord(DynamoDbClient ddb) {

       String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
       try {
           List<AttributeValue> parameters = new ArrayList<>();

           AttributeValue att1 = AttributeValue.builder()
                   .n(String.valueOf("2020"))
                   .build();

           AttributeValue att2 = AttributeValue.builder()
                   .s("My Movie")
                   .build();

           AttributeValue att3 = AttributeValue.builder()
                   .s("No Information")
                   .build();

           parameters.add(att1);
```

```
            parameters.add(att2);
            parameters.add(att3);

            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added new movie.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItem(DynamoDbClient ddb) {

        String sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2013"))
                .build();

        AttributeValue att2 = AttributeValue.builder()
                .s("The East")
                .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            executeStatementRequest(ddb, sqlStatement, parameters);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    // Query the table where the year is 2013.
    public static void queryTable(DynamoDbClient ddb) {
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year = ? ORDER BY
 year";
        try {
            List<AttributeValue> parameters = new ArrayList<>();
            AttributeValue att1 = AttributeValue.builder()
                    .n(String.valueOf("2013"))
                    .build();
            parameters.add(att1);

            // Get items in the table and write out the ID value.
            ExecuteStatementResponse response = executeStatementRequest(ddb,
 sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
 response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
 {

        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,
            List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
                .statement(statement)
                .parameters(parameters)
                .build();

        return ddb.executeStatement(request);
```

```
    }

    private static void processResults(ExecuteStatementResponse
 executeStatementResult) {
        System.out.println("ExecuteStatement successful: " +
 executeStatementResult.toString());
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ExecuteStatement」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

単一の PartiQL ステートメントを実行します。

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";
```

```
export const main = async () => {
  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");
  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log(`Coffee inserted.`);

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log(`Updated coffee`);

  /**
```

```
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ExecuteStatement」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
          <fileName>
        Where:
           fileName - The path to the moviedata.json you can download from the
 Amazon DynamoDB Developer Guide.
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(1)
    }

    val ddb = DynamoDbClient { region = "us-east-1" }
    val tableName = "MoviesPartiQ"

    // Get the moviedata.json from the Amazon DynamoDB Developer Guide.
    val fileName = args[0]
    println("Creating an Amazon DynamoDB table named MoviesPartiQ with a key
 named id and a sort key named title.")
    createTablePartiQL(ddb, tableName, "year")
    loadDataPartiQL(ddb, fileName)

    println("******* Getting data from the MoviesPartiQ table.")
    getMoviePartiQL(ddb)

    println("******* Putting a record into the MoviesPartiQ table.")
    putRecordPartiQL(ddb)

    println("******* Updating a record.")
    updateTableItemPartiQL(ddb)

    println("******* Querying the movies released in 2013.")
    queryTablePartiQL(ddb)

    println("******* Deleting the MoviesPartiQ table.")
    deleteTablePartiQL(tableName)
}

suspend fun createTablePartiQL(ddb: DynamoDbClient, tableNameVal: String, key:
 String) {
    val attDef = AttributeDefinition {
        attributeName = key
        attributeType = ScalarAttributeType.N
    }
    val attDef1 = AttributeDefinition {
        attributeName = "title"
        attributeType = ScalarAttributeType.S
    }

    val keySchemaVal = KeySchemaElement {
        attributeName = key
        keyType = KeyType.Hash
    }

    val keySchemaVal1 = KeySchemaElement {
        attributeName = "title"
        keyType = KeyType.Range
    }

    val provisionedVal = ProvisionedThroughput {
        readCapacityUnits = 10
        writeCapacityUnits = 10
    }

    val request = CreateTableRequest {
        attributeDefinitions = listOf(attDef, attDef1)
        keySchema = listOf(keySchemaVal, keySchemaVal1)
        provisionedThroughput = provisionedVal
        tableName = tableNameVal
    }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists { // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun loadDataPartiQL(ddb: DynamoDbClient, fileName: String) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
    var t = 0
    while (iter.hasNext()) {
        if (t == 200) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
        parameters.add(AttributeValue.N(year.toString()))
        parameters.add(AttributeValue.S(title))
        parameters.add(AttributeValue.S(info))

        executeStatementPartiQL(ddb, sqlStatement, parameters)
        println("Added Movie $title")
        parameters.clear()
        t++
    }
}

suspend fun getMoviePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and title=?"
    val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
    parameters.add(AttributeValue.N("2012"))
    parameters.add(AttributeValue.S("The Perks of Being a Wallflower"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun putRecordPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2020"))
    parameters.add(AttributeValue.S("My Movie"))
    parameters.add(AttributeValue.S("No Info"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Added new movie.")
}

suspend fun updateTableItemPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":[\"Merian C.
 Cooper\",\"Ernest B. Schoedsack\' where year=? and title=?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    parameters.add(AttributeValue.S("The East"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Item was updated!")
}

// Query the table where the year is 2013.
suspend fun queryTablePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year = ?"

    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun deleteTablePartiQL(tableNameVal: String) {
    val request = DeleteTableRequest {
        tableName = tableNameVal
    }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun executeStatementPartiQL(
    ddb: DynamoDbClient,
    statementVal: String,
    parametersVal: List<AttributeValue>
): ExecuteStatementResponse {
    val request = ExecuteStatementRequest {
        statement = statementVal
        parameters = parametersVal
    }

    return ddb.executeStatement(request)
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「ExecuteStatement」を参照してください。

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\testable_readline;
use function AwsUtilities\loadMovieData;

class GettingStartedWithPartiQL
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQL($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);
        echo "Movie added and updated.\n";



        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "\nThe movie {$movie['Items'][0]['title']['S']} was released in
 {$movie['Items'][0]['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Items'][0]['title']
['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "Okay, you have rated {$movie['Items'][0]['title']['S']} as a
 {$movie['Items'][0]['rating']['N']}\n";

        $service->deleteItemByPartiQL($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

```
    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「ExecuteStatement」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

PartiQL ステートメントを実行できるクラスを作成します。

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
```

```
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

テーブルを作成し、PartiQL クエリを実行するシナリオを実行します。

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL single statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    title = "24 Hour PartiQL People"
    year = datetime.now().year
    plot = "A group of data developers discover a new query language they can't
stop using."
    rating = Decimal("9.9")

    print(f"Inserting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f"INSERT INTO \"{table_name}\" VALUE {{'title': ?, 'year': ?,
'info': ?}}",
        [title, year, {"plot": plot, "rating": rating}],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data for movie '{title}' released in {year}.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    rating = Decimal("2.4")
    print(f"Updating movie '{title}' with a rating of {float(rating)}.")
    wrapper.run_partiql(
        f'UPDATE "{table_name}" SET info.rating=? WHERE title=? AND year=?',
        [rating, title, year],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data again to verify our update.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)
    print(f"Deleting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f'DELETE FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ExecuteStatement」を参照してください。

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

テーブルを作成し、PartiQL クエリを実行するシナリオを実行します。

```
table_name = "doc-example-table-movies-partiql-#{rand(10**8)}"
```

```
  scaffold = Scaffold.new(table_name)
  sdk = DynamoDBPartiQLSingle.new(table_name)

  new_step(1, "Create a new DynamoDB table if none already exists.")
  unless scaffold.exists?(table_name)
    puts("\nNo such table: #{table_name}. Creating it...")
    scaffold.create_table(table_name)
    print "Done!\n".green
  end

  new_step(2, "Populate DynamoDB table with movie data.")
  download_file = "moviedata.json"
  puts("Downloading movie database to #{download_file}...")
  movie_data = scaffold.fetch_movie_data(download_file)
  puts("Writing movie data from #{download_file} into your table...")
  scaffold.write_batch(movie_data)
  puts("Records added: #{movie_data.length}.")
  print "Done!\n".green

  new_step(3, "Select a single item from the movies table.")
  response = sdk.select_item_by_title("Star Wars")
  puts("Items selected for title 'Star Wars': #{response.items.length}\n")
  print "#{response.items.first}".yellow
  print "\n\nDone!\n".green

  new_step(4, "Update a single item from the movies table.")
  puts "Let's correct the rating on The Big Lebowski to 10.0."
  sdk.update_rating_by_title("The Big Lebowski", 1998, 10.0)
  print "\nDone!\n".green

  new_step(5, "Delete a single item from the movies table.")
  puts "Let's delete The Silence of the Lambs because it's just too scary."
  sdk.delete_item_by_title("The Silence of the Lambs", 1991)
  print "\nDone!\n".green

  new_step(6, "Insert a new item into the movies table.")
  puts "Let's create a less-scary movie called The Prancing of the Lambs."
  sdk.insert_item("The Prancing of the Lambs", 2005, "A movie about happy
 livestock.", 5.0)
  print "\nDone!\n".green

  new_step(7, "Delete the table.")
  if scaffold.exists?(table_name)
    scaffold.delete_table
    end
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ExecuteStatement」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

    let pt = ProvisionedThroughput::builder()
        .read_capacity_units(10)
        .write_capacity_units(5)
        .build()
        .expect("creating ProvisionedThroughput");

    match client
```

```
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .provisioned_throughput(pt)
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn add_item(client: &Client, item: Item) -> Result<(),
 SdkError<ExecuteStatementError>> {
    match client
        .execute_statement()
        .statement(format!(
            r#"INSERT INTO "{}" VALUE {{
                "{}": ?,
                "acount_type": ?,
                "age": ?,
                "first_name": ?,
                "last_name": ?
        }} "#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![
            AttributeValue::S(item.utype),
            AttributeValue::S(item.age),
            AttributeValue::S(item.first_name),
            AttributeValue::S(item.last_name),
        ]))
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}

async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
 Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
```

```
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「ExecuteStatement」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して DynamoDB のドキュメントモデルを使用する

次のコード例は、DynamoDB のドキュメントモデルと AWS SDK を使用して、作成、読み込み、更新、削除 (CRUD) オペレーションとバッチオペレーションを実行する方法を示しています。

詳細については、「ドキュメントモデル」を参照してください。

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ドキュメントモデルを使用して CRUD オペレーションを実行します。

```
    /// <summary>
    /// Performs CRUD operations on an Amazon DynamoDB table.
    /// </summary>
    public class MidlevelItemCRUD
    {
        public static async Task Main()
        {
            var tableName = "ProductCatalog";
            var sampleBookId = 555;

            var client = new AmazonDynamoDBClient();
            var productCatalog = LoadTable(client, tableName);

            await CreateBookItem(productCatalog, sampleBookId);
            RetrieveBook(productCatalog, sampleBookId);

            // Couple of sample updates.
            UpdateMultipleAttributes(productCatalog, sampleBookId);
            UpdateBookPriceConditionally(productCatalog, sampleBookId);

            // Delete.
            await DeleteBook(productCatalog, sampleBookId);
        }

        /// <summary>
        /// Loads the contents of a DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table to load.</param>
        /// <returns>A DynamoDB table object.</returns>
        public static Table LoadTable(IAmazonDynamoDB client, string tableName)
        {
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
 ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
 sampleBookId)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
 3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };

            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
 ID.</param>
        public static async void RetrieveBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
 "Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
 config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
 ID.</param>
```

```
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
 config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
 updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
 ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally()
 ***");
```

```
           int partitionKey = sampleBookId;

           var book = new Document
           {
               ["Id"] = partitionKey,
               ["Price"] = 29.99,
           };

           // For conditional price update, creating a condition expression.
           var expr = new Expression
           {
               ExpressionStatement = "Price = :val",
           };
           expr.ExpressionAttributeValues[":val"] = 19.00;

           // Optional parameters.
           var config = new UpdateItemOperationConfig
           {
               ConditionalExpression = expr,
               ReturnValues = ReturnValues.AllNewAttributes,
           };

           Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
           Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
           PrintDocument(updatedBook);
       }

       /// <summary>
       /// Deletes the book with the supplied Id value from the DynamoDB table
       /// ProductCatalog.
       /// </summary>
       /// <param name="productCatalog">A DynamoDB table object.</param>
       /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
       public static async Task DeleteBook(
         Table productCatalog,
         int sampleBookId)
       {
           Console.WriteLine("\n*** Executing DeleteBook() ***");

           // Optional configuration.
```

```
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
            Document document = await
productCatalog.DeleteItemAsync(sampleBookId, config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
        {
            if (updatedDocument is null)
            {
                return;
            }

            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }
                Console.WriteLine($"{attribute} - {stringValue}", attribute, 
stringValue);
            }
        }
    }
```

ドキュメントモデルを使用してバッチ書き込みオペレーションを実行します。

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
    public class MidLevelBatchWriteItem
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown
at this time.
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in
SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }

        /// <summary>
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");

            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in 
 MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

ドキュメントモデルを使用してテーブルをスキャンします。

```
   /// <summary>
   /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
   /// table for values.
   /// </summary>
   public class MidLevelScanOnly
   {
       public static async Task Main()
       {
           IAmazonDynamoDB client = new AmazonDynamoDBClient();

           Table productCatalogTable = Table.LoadTable(client, 
"ProductCatalog");

           await FindProductsWithNegativePrice(productCatalogTable);
           await FindProductsWithNegativePriceWithConfig(productCatalogTable);
       }

       /// <summary>
       /// Retrieves any products that have a negative price in a DynamoDB 
table.
```

```
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePrice(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
 0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            Search search = productCatalogTable.Scan(scanFilter);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePrice:
 printing ...........");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
 0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig: 
printing ............");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
 primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

ドキュメントモデルを使用して、テーブルをクエリおよびスキャンする。

```
    /// <summary>
    /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class MidLevelQueryAndScan
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            // Query examples.
            Table replyTable = Table.LoadTable(client, "Reply");
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 2";

            await FindRepliesInLast15Days(replyTable);
            await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
 threadSubject);
            await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
 threadSubject);

            // Get Example.
            Table productCatalogTable = Table.LoadTable(client,
 "ProductCatalog");
            int productId = 101;

            await GetProduct(productCatalogTable, productId);
        }

        /// <summary>
        /// Retrieves information about a product from the DynamoDB table
        /// ProductCatalog based on the product ID and displays the information
        /// on the console.
        /// </summary>
        /// <param name="tableName">The name of the table from which to retrieve
        /// product information.</param>
        /// <param name="productId">The ID of the product to retrieve.</param>
        public static async Task GetProduct(Table tableName, int productId)
        {
            Console.WriteLine("*** Executing GetProduct() ***");
            Document productDocument = await tableName.GetItemAsync(productId);
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        public static async Task FindRepliesInLast15Days(
          Table table)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            // Use Query overloads that take the minimum required query
parameters.
            Search search = table.Query(filter);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15Days:
printing ............");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
```

```
           while (!search.IsDone);
       }

       /// <summary>
       /// Retrieve replies made during a specific time period.
       /// </summary>
       /// <param name="table">The table we want to query.</param>
       /// <param name="forumName">The name of the forum that we're interested
in.</param>
       /// <param name="threadSubject">The subject of the thread, which we are
       /// searching for replies.</param>
       public static async Task FindRepliesPostedWithinTimePeriod(
         Table table,
         string forumName,
         string threadSubject)
       {
           DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
           DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0,
0));

           var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadSubject);
           filter.AddCondition("ReplyDateTime", QueryOperator.Between,
startDate, endDate);

           var config = new QueryOperationConfig()
           {
               Limit = 2, // 2 items/page.
               Select = SelectValues.SpecificAttributes,
               AttributesToGet = new List<string>
       {
         "Message",
         "ReplyDateTime",
         "PostedBy",
       },
               ConsistentRead = true,
               Filter = filter,
           };

           Search search = table.Query(config);

           do
           {
               var documentList = await search.GetNextSetAsync();
               Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing 
replies posted within dates: {0} and {1} ............", startDate, endDate);

               foreach (var document in documentList)
               {
                   PrintDocument(document);
               }
           }
           while (!search.IsDone);
       }

       /// <summary>
       /// Perform a query for replies made in the last 15 days using a DynamoDB
       /// QueryOperationConfig object.
       /// </summary>
       /// <param name="table">The table we want to query.</param>
       /// <param name="forumName">The name of the forum that we're interested 
in.</param>
       /// <param name="threadName">The bane of the thread that we are searching
       /// for replies.</param>
       public static async Task FindRepliesInLast15DaysWithConfig(
         Table table,
         string forumName,
         string threadName)
       {
           DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
           var filter = new QueryFilter("Id", QueryOperator.Equal, forumName + 
"#" + threadName);
           filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan, 
twoWeeksAgoDate);

           var config = new QueryOperationConfig()
           {
               Filter = filter,

               // Optional parameters.
               Select = SelectValues.SpecificAttributes,
               AttributesToGet = new List<string>
               {
                 "Message",
                 "ReplyDateTime",
                 "PostedBy",
               },
               ConsistentRead = true,
           };

           Search search = table.Query(config);

           do
           {
               var documentSet = await search.GetNextSetAsync();
               Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing ...........");

               foreach (var document in documentSet)
               {
                   PrintDocument(document);
               }
           }
           while (!search.IsDone);
       }

       /// <summary>
       /// Displays the contents of the passed DynamoDB document on the console.
       /// </summary>
       /// <param name="document">A DynamoDB document to display.</param>
       public static void PrintDocument(Document document)
       {
           Console.WriteLine();
           foreach (var attribute in document.GetAttributeNames())
           {
               string stringValue = null;
               var value = document[attribute];

               if (value is Primitive)
               {
                   stringValue = value.AsPrimitive().Value.ToString();
               }
               else if (value is PrimitiveList)
               {
                   stringValue = string.Join(",", (from primitive
                     in value.AsPrimitiveList().Entries
                                                   select
primitive.Value).ToArray());
               }

               Console.WriteLine($"{attribute} - {stringValue}");
           }
       }
   }
```

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して DynamoDB 用の高レベルのオブジェクト永続性モデルを使用する

次のコード例は、DynamoDB と AWS SDK のオブジェクト永続性モデルを使用して、作成、読み取り、更新、削除 (CRUD) オペレーションとバッチオペレーションを実行する方法を示しています。

詳細については、「オブジェクト永続性モデル」を参照してください。

.NET

AWS SDK for .NET

> Note
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

高レベルのオブジェクト永続性モデルを使用して CRUD オペレーションを実行します。

```
    /// <summary>
    /// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
    /// table.
    /// </summary>
    public class HighLevelItemCrud
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await PerformCRUDOperations(context);
        }

        public static async Task PerformCRUDOperations(IDynamoDBContext context)
        {
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

高レベルのオブジェクト永続性モデルを使用してバッチ書き込みオペレーションを実行します。

```
/// <summary>
/// Performs high-level batch write operations to an Amazon DynamoDB table.
/// This example was written using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class HighLevelBatchWriteItem
{
    public static async Task SingleTableBatchWrite(IDynamoDBContext context)
    {
        Book book1 = new Book
        {
            Id = 902,
            InPublication = true,
            Isbn = "902-11-11-1111",
            PageCount = "100",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book3 in batch write",
        };

        Book book2 = new Book
        {
            Id = 903,
            InPublication = true,
            Isbn = "903-11-11-1111",
            PageCount = "200",
            Price = 10,
            ProductCategory = "Book",
                Title = "My book4 in batch write",
            };

            var bookBatch = context.CreateBatchWrite<Book>();
            bookBatch.AddPutItems(new List<Book> { book1, book2 });

            Console.WriteLine("Adding two books to ProductCatalog table.");
            await bookBatch.ExecuteAsync();
        }

        public static async Task MultiTableBatchWrite(IDynamoDBContext context)
        {
            // New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0,
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
            forumBatch.AddPutItem(newForum);

            // New Thread item.
            Thread newThread = new Thread
            {
                ForumName = "S3 forum",
                Subject = "My sample question",
                KeywordTags = new List<string> { "S3", "Bucket" },
                Message = "Message text",
            };

            DynamoDBOperationConfig config = new DynamoDBOperationConfig();
            config.SkipVersionCheck = true;
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key 
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

            Console.WriteLine("Performing batch write in 
MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
        }
        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

オブジェクト永続性モデルを使用して、任意のデータをテーブルにマッピングします。

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table,
retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };

            Book myBook = new Book
            {
                Id = 501,
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };

            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

高レベルのオブジェクト永続性モデルを使用してテーブルをクエリおよびスキャンします。

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);

            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";

            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName, 
threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }

        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn} 
\n No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15 
days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the 
query.</param>
        /// <param name="forumName">The name of the forum that we're interested 
in.</param>
        /// <param name="threadSubject">The thread object containing the query 
parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string replyId = $"{forumName} #{threadSubject}";
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            List<object> times = new List<object>();
            times.Add(twoWeeksAgoDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId,
cfg);
            IEnumerable<Reply> latestReplies = await
response.GetRemainingAsync();

            Console.WriteLine("\nReplies in last 15 days:");

            foreach (Reply r in latestReplies)
            {

Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries for replies posted within a specific time period.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">Information about the subject that we're
        /// interested in.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nReplies posted within time period:");
            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);

            List<object> times = new List<object>();
            times.Add(startDate);
            times.Add(endDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId,
cfg);
            IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

            foreach (Reply r in repliesInAPeriod)
            {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries the DynamoDB ProductCatalog table for products costing less
        /// than zero.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to perform the
        /// query.</param>
        public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
        {
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
            scs.Add(sc2);

            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
        }
    }
```

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した DynamoDB 用のサーバーレスサンプル

次のコード例は、AWS SDK で DynamoDB を使用する方法を示しています。

例

- DynamoDB トリガーから Lambda 関数を呼び出す
- DynamoDB トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

DynamoDB トリガーから Lambda 関数を呼び出す

次のコードの例では、DynamoDB ストリームからレコードを受信することによってトリガーされるイベントを受け取る Lambda 関数の実装方法が示されています。関数は DynamoDB ペイロードを取得し、レコードの内容をログ記録します。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で DynamoDB イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Go を使用して Lambda で DynamoDB イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

JavaScript

SDK for JavaScript (v3)

> ⓘ Note
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で DynamoDB イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

TypeScript を使用した Lambda での DynamoDB イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

PHP を使用した Lambda での DynamoDB イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK for Python (Boto3)

> Note
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Python を使用して Lambda で DynamoDB イベントの消費。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDK for Ruby

> Note
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Ruby を使用して Lambda で DynamoDB イベントの消費。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK for Rust

> Note
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で DynamoDB イベントを利用します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };
```

```
// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features = ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();
    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

DynamoDB トリガーで Lambda 関数のバッチアイテムの失敗をレポートする

次のコードの例では、DynamoDB ストリームからイベントを受け取る Lambda 関数の部分的なバッチレスポンスの実装方法が示されています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

.NET

AWS SDK for .NET

> ⓘ Note
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

.NET を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
```

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Go を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Java を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 Serializable> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

JavaScript を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

TypeScript を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBBatchItemFailure, DynamoDBStreamEvent } from "aws-lambda";

export const handler = async (event: DynamoDBStreamEvent):
 Promise<DynamoDBBatchItemFailure[]> => {

    const batchItemsFailures: DynamoDBBatchItemFailure[] = []
    let curRecordSequenceNumber

    for(const record of event.Records) {
        curRecordSequenceNumber = record.dynamodb?.SequenceNumber

        if(curRecordSequenceNumber) {
            batchItemsFailures.push({
                itemIdentifier: curRecordSequenceNumber
            })
        }
    }
    return batchItemsFailures
}
```

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

PHP を使用した Lambda での DynamoDB バッチ項目失敗のレポート。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Python を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Ruby を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK for Rust

> Note
>
> GitHub には、その他のリソースもあります。サーバーレスサンプルリポジトリで完全な例を検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で DynamoDB のバッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
```

```
/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
```

```
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した DynamoDB のクロスサービスの例

次のサンプルアプリケーションでは、AWS SDK を使用して DynamoDB を他の AWS のサービス と組み合わせます。それぞれの例には、GitHub へのリンクがあり、アプリケーションを設定および実行する方法についての説明を参照できます。

例

- DynamoDB テーブルにデータを送信するアプリケーションを構築する
- COVID-19 データを追跡する API Gateway REST API を作成する
- ステップ関数でメッセンジャーアプリケーションを作成する

- ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成
- DynamoDB データを追跡するウェブアプリケーションを作成する
- API Gateway で WebSocket チャットアプリケーションを作成する
- AWS SDK を使用して、Amazon Rekognition で画像内の PPE を検出する
- ブラウザからの Lambda 関数の呼び出し
- AWS SDK を使用した Amazon DynamoDB のパフォーマンスのモニタリング
- AWS SDK を使用して、EXIF とその他の画像情報を保存する
- API Gateway を使用して Lambda 関数を呼び出す
- Step Functions を使用して Lambda 関数を呼び出す
- スケジュールされたイベントを使用した Lambda 関数の呼び出し

DynamoDB テーブルにデータを送信するアプリケーションを構築する

次のコード例は、Amazon DynamoDB テーブルにデータを送信し、ユーザーがテーブルを更新したときに通知するアプリケーションを構築する方法を示しています。

Java

SDK for Java 2.x

Amazon DynamoDB Java API を使用してデータを送信し、Amazon Simple Notification Service Java API を使用してテキストメッセージを送信する動的ウェブアプリケーションを作成する方法について説明します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

この例では、ユーザーが Amazon DynamoDB テーブルにデータを送信し、Amazon Simple Notification Service (Amazon SNS) を使用して管理者にテキストメッセージを送信できるようにするアプリを構築する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- DynamoDB
- Amazon SNS

Kotlin

SDK for Kotlin

Amazon DynamoDB Kotlin API を使用してデータを送信し、Amazon SNS Kotlin API を使用してテキストメッセージを送信するネイティブ Android アプリケーションを作成する方法を示しています。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SNS

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

COVID-19 データを追跡する API Gateway REST API を作成する

次のコード例は、架空のデータを使用して、米国の COVID-19 の日常的なケースを追跡するシステムをシミュレートする REST API を作成する方法を示しています。

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3) で AWS Chalice を使用して Amazon API Gateway、AWS Lambda、Amazon DynamoDB を使用するサーバーレス REST API を作成する方法を示しています。REST API は、架空のデータを使用して、米国の COVID-19 の日常的なケースを追跡するシステムをシミュレートします。以下ではその方法を説明しています。

- API Gateway を介して送信される REST リクエストを処理するために呼び出される Lambda 関数内のルートを、AWS Chalice を使って定義します。
- Lambda 関数を使用して、DynamoDB テーブルにデータを取得して保存し、REST リクエストを処理します。
- AWS CloudFormation テンプレート内のテーブル構造とセキュリティロールのリソースを定義します。
- AWS Chalice と CloudFormation を使用して、必要なすべてのリソースをパッケージ化してデプロイします。
- CloudFormation を使用して、作成されたすべてのリソースをクリーンアップします。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

ステップ関数でメッセンジャーアプリケーションを作成する

次のコード例は、データベーステーブルからメッセージレコードを取得する AWS Step Functions メッセンジャーアプリケーションを作成する方法を示しています。

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3) を AWS Step Functions と使用して、Amazon DynamoDB テーブルからメッセージレコードを取得し、Amazon Simple Queue Service (Amazon SQS) で送信するメッセンジャーアプリケーションを作成する方法を紹介します。ステートマシンは AWS Lambda 関数を使用して、データベースで未送信メッセージをスキャンします。

- Amazon DynamoDB テーブルからメッセージレコードを取得および更新するステートマシンを作成します。
- ステートマシンの定義を更新して、Amazon Simple Queue Service (Amazon SQS) にもメッセージを送信します。
- ステートマシンの実行を開始および停止します。
- サービス統合を使用して、ステートマシンから Lambda、DynamoDB、および Amazon SQS に接続します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Lambda
- Amazon SQS
- ステップ関数

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成

以下のコード例は、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションの作成方法を示しています。

.NET

AWS SDK for .NET

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK for C++

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK for Java 2.x

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK for Kotlin

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK for PHP

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK for Rust

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

DynamoDB データを追跡するウェブアプリケーションを作成する

次のコードは、Amazon DynamoDB の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを送信するウェブアプリケーションを作成する方法を示します。

.NET

AWS SDK for .NET

Amazon DynamoDB .NET API を使用して、DynamoDB 作業データを追跡する動的ウェブアプリケーションを作成する方法を示しています。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SES

Java

SDK for Java 2.x

Amazon DynamoDB API を使用して、DynamoDB 作業データを追跡する動的ウェブアプリケーションを作成する方法を示しています。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SES

JavaScript

SDK for JavaScript (v3)

Amazon DynamoDB API を使用して、DynamoDB 作業データを追跡する動的ウェブアプリケーションを作成する方法を示しています。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SES

Kotlin

SDK for Kotlin

Amazon DynamoDB API を使用して、DynamoDB 作業データを追跡する動的ウェブアプリケーションを作成する方法を示しています。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SES

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3) を使用して Amazon DynamoDB の作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信する REST サービスを作成する方法を示します。この例では、Flask ウェブフレームワークを使用して HTTP ルーティングを処理し、React ウェブページと統合して完全に機能するウェブアプリケーションを提供しています。

- AWS のサービス と統合できる Flask REST サービスを構築します。
- DynamoDB テーブルに保存されている作業項目の読み取り、書き込み、更新を行います。
- Amazon SES を使用して作業項目のレポートを E メールで送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub の「AWS コード例のレポジトリ」で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SES

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

API Gateway で WebSocket チャットアプリケーションを作成する

次のコード例は、Amazon API Gateway 上に構築された WebSocket API によって提供されるチャットアプリケーションを作成する方法を示しています。

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3) と Amazon API Gateway V2 を使用して、AWS Lambda や Amazon DynamoDB と統合する WebSocket API を作成する方法を示します。

- API Gateway で提供される WebSocket API を作成します。
- DynamoDB に接続を保存し、他のチャット参加者にメッセージを投稿する Lambda ハンドラを定義します。
- WebSocket チャットアプリケーションに接続し、WebSockets パッケージを使用してメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して、Amazon Rekognition で画像内の PPE を検出する

次のコード例は、Amazon Rekognition を使用して画像内の個人用防護具 (PPE) を検出するアプリケーションを構築する方法を示しています。

Java

SDK for Java 2.x

AWS Lambda 個人用保護具のイメージを検出する関数の作成方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript (v3)

Amazon Rekognition で、AWS SDK for JavaScript を使用して Amazon Simple Storage Service (Amazon S3) バケット内にあるイメージで個人用保護具 (PPE) を検出するアプリケーションを作成する方法を示します。アプリケーションは、結果を Amazon DynamoDB テーブルに保存し、Amazon Simple Email Service (Amazon SES) を使用して結果を記載した E メール通知を管理者に送信します。

以下ではその方法を説明しています。

- Amazon Cognito を使用して認証されていないユーザーを作成します。
- Amazon Rekognition を使用して、PPE の画像を分析します。
- Amazon SES の E メールアドレスを検証します。
- 結果で DynamoDB テーブルを更新します。
- Amazon SES を使用して E メール通知を送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

ブラウザからの Lambda 関数の呼び出し

次のコード例は、ブラウザから AWS Lambda 関数を呼び出す方法を示しています。

JavaScript

SDK for JavaScript (v2)

Amazon DynamoDB のテーブルをユーザーの選択内容で更新する AWS Lambda 関数を使用した、ブラウザベースのアプリケーションを作成することができます。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Lambda

SDK for JavaScript (v3)

Amazon DynamoDB のテーブルをユーザーの選択内容で更新する AWS Lambda 関数を使用した、ブラウザベースのアプリケーションを作成することができます。このアプリは AWS SDK for JavaScript v3 を使用します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Lambda

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した Amazon DynamoDB のパフォーマンスのモニタリング

次のコード例は、DynamoDB のパフォーマンスをモニタリングするように、アプリケーションを設定する方法を示しています。

Java

SDK for Java 2.x

この例では、DynamoDB のパフォーマンスをモニタリングするように Java アプリケーションを設定する方法を示します。アプリケーションからメトリクスデータを CloudWatch に送信してパフォーマンスをモニタリングできます。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- CloudWatch
- DynamoDB

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して、EXIF とその他の画像情報を保存する

次のコードサンプルは、以下の操作方法を示しています。

- JPG、JPEG、または PNG ファイルから EXIF 情報を取得します。

- Amazon S3 バケットにイメージファイルをアップロードします。
- Amazon Rekognition を使用して、ファイル内の 3 つの上位属性 (ラベル) を特定します。
- EXIF およびラベル情報を、リージョンの Amazon DynamoDB テーブルに追加します。

Rust

SDK for Rust

JPG、JPEG、または PNG ファイルから EXIF 情報を取得し、イメージファイルを Amazon S3 バケットにアップロードし、Amazon Rekognition を使用してファイル内の 3 つの上位属性 (Amazon Rekognition のラベル) を特定し、リージョンの Amazon DynamoDB テーブルに EXIF およびラベル情報を追加します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

API Gateway を使用して Lambda 関数を呼び出す

次のコード例は、Amazon API Gateway によって呼び出される AWS Lambda 関数の作成方法を示しています。

Java

SDK for Java 2.x

Lambda Java ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、特定のユースケースを実行する異なる AWS サービスを呼び出します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification

Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Lambda JavaScript ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、特定のユースケースを実行する異なる AWS サービスを呼び出します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Step Functions を使用して Lambda 関数を呼び出す

次のコード例は、AWS Lambda 関数を順次呼び出す AWS Step Functions ステートマシンを作成する方法を示しています。

Java

SDK for Java 2.x

AWS Step Functions と AWS SDK for Java 2.x を使用して AWS サーバーレスワークフローを作成する方法を示します。各ワークフローステップは、AWS Lambda 関数を使用して実装されます。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

JavaScript

SDK for JavaScript (v3)

AWS Step Functions と AWS SDK for JavaScript を使用して AWS サーバーレスワークフローを作成する方法を示します。各ワークフローステップは、AWS Lambda 関数を使用して実装されます。

Lambda はサーバーをプロビジョニングまたは管理サーバーなしでもコードを実行し、有効にするコンピューティングサービスです。Step Functions は、ビジネス上重要なアプリケーションを構築するために Lambda 関数と他の AWS のサービスを組み合わせることができるサーバーレスオーケストレーションサービスです。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- DynamoDB
- Lambda
- Amazon SES
- ステップ関数

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

Java

SDK for Java 2.x

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda Java ランタイム API を使用して Lambda 関数を作成します。この例では、特定のユースケースを実行する異なる AWS サービスを呼び出します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、特定のユースケースを実行する異なる AWS サービスを呼び出します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例は、AWS SDK for JavaScript v3 デベロッパーガイドでも使用できます。

この例で使用されているサービス

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「AWS SDK で DynamoDB を使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon DynamoDB のセキュリティとコンプライアンス

AWS では、クラウドのセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された AWS のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、AWS と顧客の間の責任共有です。責任共有モデルでは、この責任がクラウドのセキュリティおよびクラウド内のセキュリティとして説明されています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する責任を負います。また、AWS は、使用するサービスを安全に提供します。セキュリティの有効性は、AWS コンプライアンスプログラムの一環として、サードパーティーの審査機関によって定期的にテストおよび検証されています。DynamoDB に適用するコンプライアンスプログラムの詳細については、コンプライアンスプログラムによる対象範囲内の AWS サービスを参照してください。
- クラウド内のセキュリティ – お客様の責任は使用する AWS のサービスによって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントでは、DynamoDB を使用するときに、責任共有モデルを適用する方法を理解できます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために DynamoDB を設定する方法を示します。また、DynamoDB リソースのモニタリングや保護に役立つ他の AWS サービスの用法についても説明します。

トピック

- Amazon DynamoDB のセキュリティベストプラクティス

Amazon DynamoDB の AWS マネージドポリシー

DynamoDB は、AWS マネージドポリシーを使用して、サービスが特定のアクションを実行するために必要な一連のアクセス許可を定義します。DynamoDB は、その AWS マネージドポリシーを維持および更新します。AWS マネージドポリシーのアクセス許可を変更することはできません。AWS マネージドポリシーの詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

DynamoDB は、新しい機能をサポートするために、AWS マネージドポリシーに新しいアクセス許可を追加する場合があります。この種の更新は、ポリシーがアタッチされているすべてのアイデンティティ (ユーザー、グループ、ロール) に影響を与えます。AWS マネージドポリシーは、新しい機能がリリースされたときや新しいオペレーションが使用可能になったときに、更新される可能性が最も高くなります。DynamoDB は AWS マネージドポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が破棄されることはありません。

AWS マネージドポリシー: DynamoDBReplicationServiceRolePolicy

`DynamoDBReplicationServiceRolePolicy` ポリシーを IAM エンティティにアタッチすることはできません。このポリシーは、ユーザーに代わってアクションを実行することを DynamoDB に許可する、サービスにリンクされたロールにアタッチします。詳細については、「グローバルテーブルでの IAM の使用」を参照してください。

このポリシーは、サービスにリンクされたロールに対して、グローバルテーブルのレプリカ間でデータをレプリケートするアクセス許可を付与します。また、ユーザーに代わってグローバルテーブルのレプリカを管理する管理者アクセス許可も付与します。

アクセス許可の詳細

このポリシーは、以下を行うアクセス許可を付与します。

- `dynamodb` — データのレプリケーションを実行し、テーブルのレプリカを管理します。
- `application-autoscaling` — テーブルの AutoScaling 設定を取得および管理します。
- `account` — レプリカのアクセシビリティを評価するためのリージョンのステータスを取得します。
- `iam` — サービスにリンクされたロールが現時点で存在しない場合、アプリケーションの AutoScaling 用のサービスにリンクされたロールを作成します。

このマネージドポリシーの定義については、こちらを参照してください。

AWS 管理ポリシー: AmazonDynamoDBReadOnlyAccess

`AmazonDynamoDBReadOnlyAccess` ポリシーは IAM ID にアタッチできます。

このポリシーは Amazon DynamoDB への読み取り専用アクセスを許可します。

許可の詳細

このポリシーには、以下の許可が含まれています。

- `Amazon DynamoDB` – Amazon DynamoDB への読み取り専用アクセスを提供します。
- `Amazon DynamoDB Accelerator (DAX)` – Amazon DynamoDB Accelerator (DAX) への読み取り専用アクセスを提供します。
- `Application Auto Scaling` — Application Auto Scaling の設定をプリンシパルが表示できるようにします。これは、テーブルにアタッチされているオートスケーリングポリシーをユーザーが表示できる場合に必須です。
- `CloudWatch` - CloudWatch で設定されたメトリクスデータとアラームをプリンシパルが表示できるようにします。これは、テーブルに対して設定された請求対象テーブルサイズと CloudWatch アラームをユーザーが表示できる場合に必須です。
- `AWS Data Pipeline` — プリンシパルが AWS Data Pipeline と関連オブジェクトを表示することを許可します。
- `Amazon EC2` – プリンシパルが Amazon EC2 VPC、サブネット、およびセキュリティグループを表示することを許可します。
- `IAM` — プリンシパルが IAM ロールを表示することを許可します。
- `AWS KMS` — AWS KMS で設定されたキーをプリンシパルが表示できるようにします。これは、ユーザーが、各自のアカウントで作成して管理している AWS KMS keys を表示するために必要です。
- `Amazon SNS` – プリンシパルが Amazon SNS のトピックとトピック別のサブスクリプションを一覧表示することを許可します。
- `AWS Resource Groups` — プリンシパルがリソースグループとクエリを表示することを許可します。
- `AWS Resource Groups Tagging` — プリンシパルがリージョン内のタグ付けされたリソースまたは以前にタグ付けされたリソースのすべてを一覧表示することを許可します。
- `Kinesis` — プリンシパルが Kinesis データストリーム記述を表示することを許可します。

- Amazon CloudWatch Contributor Insights — プリンシパルが Contributor Insights ルールによって収集された時系列データを表示することを許可します。

ポリシーを JSON 形式で確認するには、「AmazonDynamoDBReadOnlyAccess」を参照してください。

DynamoDB での AWS マネージドポリシーの更新

この表は、DynamoDB の AWS アクセス管理ポリシーの更新を示しています。

変更	説明	変更日
AmazonDynamoDBReadOnlyAccess の既存のポリシーに対する更新	AmazonDynamoDBReadOnlyAccess にアクセス許可 dynamodb:GetResourcePolicy を追加しました。このアクセス許可は、DynamoDB リソースにアタッチされたリソースベースのポリシーの読み取りアクセスを提供します。	2024 年 3 月 20 日
DynamoDBReplicationServiceRolePolicy の既存のポリシーに対する更新	DynamoDBReplicationServiceRolePolicy にアクセス許可 dynamodb:GetResourcePolicy を追加しました。このアクセス許可では、サービスリンクロールは、DynamoDB リソースにアタッチされたリソースベースのポリシーを読み取ることができます。	2023 年 12 月 15 日
DynamoDBReplicationServiceRolePolicy の既存のポリシーに対する更新	DynamoDBReplicationServiceRolePolicy にアクセス許可 account:ListRegions を追加しました。このアクセス許可は、サービスにリンクされたロールに対して、レプリカのアクセシビ	2023 年 5 月 10 日

変更	説明	変更日
	リティを評価することを許可します。	
DynamoDBReplicationServiceRolePolicy をマネージドポリシーのリストに追加	マネージドポリシー DynamoDBReplicationServiceRolePolicy に関する情報を追加しました。このポリシーは DynamoDB グローバルテーブルのサービスにリンクされたロールで使用されます。	2023 年 5 月 10 日
DynamoDB グローバルテーブルが変更の追跡を開始	DynamoDB グローバルテーブルが AWS マネージドポリシーの変更の追跡を開始しました。	2023 年 5 月 10 日

DynamoDB 用のリソースベースのポリシーを使用する

DynamoDB では、テーブル、インデックス、ストリームに対してリソースベースのポリシーを利用できます。リソースベースのポリシーでは、各リソースにアクセスできるユーザーと、各リソースに対して実行できるアクションを指定することで、アクセス許可を定義できます。

リソースベースのポリシーを DynamoDB リソース (テーブルやストリームなど) にアタッチできます。このポリシーでは、Identity and Access Management (IAM) プリンシパルに対して、該当する DynamoDB リソースで特定のアクションを実行するためのアクセス許可を指定します。例えば、テーブルにアタッチされたポリシーでは、そのテーブルとそのインデックスへのアクセス許可を指定します。そのため、リソースベースのポリシーを使用すれば、リソース単位でアクセス許可を定義することで、DynamoDB のテーブル、インデックス、ストリームへのアクセスを簡単に制御できます。DynamoDB リソースにアタッチできるポリシーの最大サイズは 20 KB です。

リソースベースのポリシーを使用する大きな利点は、さまざまな AWS アカウント で IAM プリンシパルにクロスアカウントアクセスを許可するための、クロスアカウントアクセス制御が簡単になることです。詳細については、「クロスアカウントアクセスのリソースベースのポリシー」を参照してください。

リソースベースのポリシーは、IAM Access Analyzer の外部アクセスアナライザーやブロックパブリックアクセス (BPA) 機能と統合することもできます。IAM Access Analyzer は、リソースベースのポリシーで指定された外部エンティティへのクロスアカウントアクセスについて報告します。また、情報を可視化してくれるので、アクセス許可を調整し、最小特権の原則を守るうえでも役立ちます。BPA では、DynamoDB のテーブル、インデックス、ストリームへのパブリックアクセスを阻止できます。BPA は、リソースベースのポリシーの作成と変更のワークフローで自動的に有効になります。

トピック

リソースベースのポリシーを指定してテーブルを作成する

DynamoDB コンソール、CreateTable API、AWS CLI、AWS SDK、または AWS CloudFormation テンプレートを使用して、テーブルの作成時にリソースベースのポリシーを追加できます。

AWS CLI

次の例では、`create-table` AWS CLI コマンドを使用して、*MusicCollection* という名前のテーブルを作成します。このコマンドでは、そのテーブルにリソースベースのポリシーを追加する `resource-policy` パラメータも指定されています。このポリシーは、ユーザー *John* に対して、該当のテーブルで RestoreTableToPointInTime、GetItem、PutItem の API アクションを実行することを許可しています。

*######*のテキストを、リソース固有の情報に必ず置き換えてください。

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --resource-policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::123456789012:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:RestoreTableToPointInTime\",
                        \"dynamodb:GetItem\",
                        \"dynamodb:DescribeTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS Management Console

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ダッシュボードで、[テーブルの作成] を選択します。
3. [テーブルの詳細] に、テーブル名、パーティションキー、ソートキーの詳細を入力します。
4. [テーブル設定] セクションで [設定をカスタマイズ] を選択します。
5. (オプション) [テーブルクラス]、[キャパシティー計算ツール]、[読み込み/書き込みキャパシティーの設定]、[セカンダリインデックス]、[保管時の暗号化]、[削除保護] のオプションを指定します。
6. [リソースベースのポリシー] で、テーブルとそのインデックスへのアクセス許可を定義するポリシーを追加します。このポリシーでは、これらのリソースにアクセスできるユーザーと、各リ

ソースに対して実行できるアクションを指定します。ポリシーを追加するには、以下のいずれかを実行します。

- JSON ポリシードキュメントを入力するか貼り付けます。IAM ポリシー言語の詳細については、「IAM ユーザーガイド」の「JSON エディターを使用したポリシーの作成」を参照してください。

  Tip

  「Amazon DynamoDB デベロッパーガイド」のリソースベースのポリシーの例を確認するには、[ポリシーの例] を選択してください。

- [新しいステートメントを追加] を選択して新しいステートメントを追加し、提示されたフィールドに情報を入力します。このステップを、追加するステートメントの数だけ繰り返します。

Important

セキュリティ警告、エラー、提案がある場合は、ポリシーを保存する前に解決してください。

次の IAM ポリシーの例では、ユーザー *John* に対して、テーブル *MusicCollection* で RestoreTableToPointInTime、GetItem、PutItem の API アクションを実行することを許可しています。

*######*のテキストを、リソース固有の情報に必ず置き換えてください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/John"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (オプション) 新しいポリシーがリソースへのパブリックアクセスおよびクロスアカウントアクセスにどのように影響するかをプレビューするには、[外部アクセスをプレビュー] を選択します。ポリシーを保存する前に、新しい IAM Access Analyzer の結果が導入されているかどうかや、既存の結果を解決するかどうかを確認できます。アクティブなアナライザーが表示されない場合は、[Access Analyzer へ移動] を選択し、[IAM Access Analyzer] でアカウントアナライザーを作成します。詳細については、「アクセスのプレビュー」を参照してください。
8. [Create table (テーブルの作成)] を選択します。

AWS CloudFormation テンプレート

Using the AWS::DynamoDB::Table resource

次の CloudFormation テンプレートでは、AWS::DynamoDB::Table リソースを使用して、ストリームを有効にしたテーブルを作成します。このテンプレートでは、該当するテーブルとストリームの両方にアタッチされたリソースベースのポリシーも指定しています。

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MusicCollectionTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "Artist",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    }
                ],
                "BillingMode": "PROVISIONED",
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "StreamSpecification": {
                  "StreamViewType": "OLD_IMAGE",
                  "ResourcePolicy": {
                    "PolicyDocument": {
                      "Version": "2012-10-17",
                      "Statement": [
                        {
                            "Principal": {
                                "AWS": "arn:aws:iam::111122223333:user/John"
                            },
                            "Effect": "Allow",
                            "Action": [
                                "dynamodb:GetRecords",
                                "dynamodb:GetShardIterator",
                                "dynamodb:DescribeStream"
                            ],
                            "Resource": "*"
                        }
                      ]
                    }
                  }
                },
                "TableName": "MusicCollection",
                "ResourcePolicy": {
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Principal": {
                                    "AWS": [
                                        "arn:aws:iam::111122223333:user/John"
                                    ]
                                },
                                "Effect": "Allow",
                                "Action": "dynamodb:GetItem",
                                "Resource": "*"
                            }
                        ]
                    }
                }
```

```
            }

        }
    }
}
```

Using the AWS::DynamoDB::GlobalTable resource

次の CloudFormation テンプレートでは、AWS::DynamoDB::GlobalTable リソースを使用してテーブルを作成し、リソースベースのポリシーを該当テーブルとそのストリームにアタッチします。

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "GlobalMusicCollection": {
            "Type": "AWS::DynamoDB::GlobalTable",
            "Properties": {
                "TableName": "MusicCollection",
                "AttributeDefinitions": [{
                    "AttributeName": "Artist",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "Artist",
                    "KeyType": "HASH"
                }],
                "BillingMode": "PAY_PER_REQUEST",
                "StreamSpecification": {
                    "StreamViewType": "NEW_AND_OLD_IMAGES"
                },
                "Replicas": [
                    {
                        "Region": "us-east-1",
                        "ResourcePolicy": {
                            "PolicyDocument": {
                                "Version": "2012-10-17",
                                "Statement": [{
                                    "Principal": {
                                        "AWS": [
                                            "arn:aws:iam::111122223333:user/John"
                                        ]
                                    },
                                    "Effect": "Allow",
                                    "Action": "dynamodb:GetItem",
                                    "Resource": "*"
                                }]
                            }
                        },
                        "ReplicaStreamSpecification": {
                            "ResourcePolicy": {
                                "PolicyDocument": {
                                    "Version": "2012-10-17",
                                    "Statement": [{
                                        "Principal": {
                                            "AWS":
 "arn:aws:iam::111122223333:user/John"
                                        },
                                        "Effect": "Allow",
                                        "Action": [
                                            "dynamodb:GetRecords",
                                            "dynamodb:GetShardIterator",
                                            "dynamodb:DescribeStream"
                                        ],
                                        "Resource": "*"
                                    }]
                                }
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

既存のテーブルにポリシーをアタッチする

DynamoDB コンソール、PutResourcePolicy API、AWS CLI、AWS SDK、または AWS CloudFormation テンプレートを使用して、リソースベースのポリシーを既存のテーブルにアタッチしたり、既存のポリシーを変更したりできます。

AWS CLI の例: 新しいポリシーをアタッチする

次の IAM ポリシーの例では、put-resource-policy AWS CLI コマンドを使用して、リソースベースのポリシーを既存のテーブルにアタッチします。この例では、ユーザー *John* に対して、*MusicCollection* という名前の既存のテーブルで GetItem、PutItem、UpdateItem、UpdateTable の API アクションを実行することを許可しています。

*######*のテキストを、リソース固有の情報に必ず置き換えてください。

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:PutItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS CLI の例: 既存のポリシーを条件に従って更新する

テーブルの既存のリソースベースポリシーを条件に従って更新する場合は、オプションの expected-revision-id パラメータを使用できます。次の例では、ポリシーが DynamoDB に存在し、現在のリビジョン ID が expected-revision-id パラメータに指定された値と一致する場合にのみ、ポリシーを更新します。

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --expected-revision-id 1709841168699 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

AWS Management Console

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ダッシュボードから、既存のテーブルを選択します。
3. [アクセス許可] タブに移動し、[テーブルポリシーを作成] を選択します。
4. [リソースベースのポリシー] エディターで、アタッチするポリシーを追加し、[ポリシーを作成] を選択します。

   次の IAM ポリシーの例では、ユーザー *John* に対して、*MusicCollection* という名前の既存のテーブルで GetItem、PutItem、UpdateItem、UpdateTable の API アクションを実行することを許可しています。

   *######*のテキストを、リソース固有の情報に必ず置き換えてください。

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
   ```

```
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/John"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:UpdateTable"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
```

AWS SDK for Java 2.x

次の IAM ポリシーの例では、putResourcePolicy メソッドを使用して、リソースベースのポリシーを既存のテーブルにアタッチしています。このポリシーは、既存のテーブルで GetItem API アクションを実行することをユーザーに許可します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutResourcePolicyRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * Get started with the AWS SDK for Java 2.x
 */
public class PutResourcePolicy {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableArn> <allowedAWSPrincipal>

                Where:
                    tableArn - The Amazon DynamoDB table ARN to attach the policy to.
For example, arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection.
                    allowedAWSPrincipal - Allowed AWS principal ARN that the example
policy will give access to. For example, arn:aws:iam::123456789012:user/John.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableArn = args[0];
        String allowedAWSPrincipal = args[1];
        System.out.println("Attaching a resource-based policy to the Amazon DynamoDB
table with ARN " +
                tableArn);
        Region region = Region.US_WEST_2;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        String result = putResourcePolicy(ddb, tableArn, allowedAWSPrincipal);
        System.out.println("Revision ID for the attached policy is " + result);
        ddb.close();
    }

    public static String putResourcePolicy(DynamoDbClient ddb, String tableArn, String
allowedAWSPrincipal) {
        String policy = generatePolicy(tableArn, allowedAWSPrincipal);
        PutResourcePolicyRequest request = PutResourcePolicyRequest.builder()
                .policy(policy)
                .resourceArn(tableArn)
                .build();

        try {
            return ddb.putResourcePolicy(request).revisionId();
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        return "";
    }
    private static String generatePolicy(String tableArn, String allowedAWSPrincipal) {
        return "{\n" +
                "    \"Version\": \"2012-10-17\",\n" +
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Effect\": \"Allow\",\n" +
                "            \"Principal\": {\"AWS\":\"" + allowedAWSPrincipal + "\"},
\n" +
                "            \"Action\": [\n" +
                "                \"dynamodb:GetItem\"\n" +
                "            ],\n" +
                "            \"Resource\": \"" + tableArn + "\"\n" +
                "        }\n" +
                "    ]\n" +
                "}";
    }
}
```

リソースベースのポリシーをストリームにアタッチする

DynamoDB コンソール、PutResourcePolicy API、AWS CLI、AWS SDK、または AWS CloudFormation テンプレートを使用して、リソースベースのポリシーを既存のテーブルのストリームにアタッチしたり、既存のポリシーを変更したりできます。

> **Note**
>
> CreateTable API や UpdateTable API で作成中のストリームには、ポリシーをアタッチできません。ただし、テーブルの削除後にポリシーを変更または削除することはできます。また、無効になっているストリームのポリシーは変更または削除できます。

AWS CLI

次の IAM ポリシーの例では、`put-resource-policy` AWS CLI コマンドを使用して、*MusicCollection* という名前のテーブルのストリームにリソースベースのポリシーをアタッチしています。この例では、ユーザー *John* に対して、該当ストリームで GetRecords、GetShardIterator、DescribeStream の API アクションを実行することを許可しています。

*######*のテキストを、リソース固有の情報に必ず置き換えてください。

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetRecords\",
                        \"dynamodb:GetShardIterator\",
                        \"dynamodb:DescribeStream\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492\"
                }
            ]
        }"
```

AWS Management Console

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. DynamoDB コンソールのダッシュボードで [テーブル] を選択し、既存のテーブルを選択します。

   ストリームが有効になっているテーブルを選択してください。テーブルのストリームを有効にする方法については、「ストリームの有効化」を参照してください。
3. [アクセス許可] タブを選択します。
4. [アクティブストリームについてのリソースベースのポリシー] で、[ストリームポリシーを作成] を選択します。
5. [リソースベースのポリシー] エディターで、ストリームに対するアクセス許可を定義するポリシーを追加します。このポリシーでは、ストリームにアクセスできるユーザーと、ストリームに対して実行できるアクションを指定します。ポリシーを追加するには、以下のいずれかを実行します。

- JSON ポリシードキュメントを入力するか貼り付けます。IAM ポリシー言語の詳細については、「IAM ユーザーガイド」の「JSON エディターを使用したポリシーの作成」を参照してください。

  > Tip
  >
  > 「Amazon DynamoDB デベロッパーガイド」のリソースベースのポリシーの例を確認するには、[ポリシーの例] を選択してください。

- [新しいステートメントを追加] を選択して新しいステートメントを追加し、提示されたフィールドに情報を入力します。このステップを、追加するステートメントの数だけ繰り返します。

  > Important
  >
  > セキュリティ警告、エラー、提案がある場合は、ポリシーを保存する前に解決してください。

6. (オプション) 新しいポリシーがリソースへのパブリックアクセスおよびクロスアカウントアクセスにどのように影響するかをプレビューするには、[外部アクセスをプレビュー] を選択します。ポリシーを保存する前に、新しい IAM Access Analyzer の結果が導入されているかどうかや、既存の結果を解決するかどうかを確認できます。アクティブなアナライザーが表示されない場合は、[Access Analyzer へ移動] を選択し、[IAM Access Analyzer] でアカウントアナライザーを作成します。詳細については、「アクセスのプレビュー」を参照してください。
7. [Create policy] を選択します。

次の IAM ポリシーの例では、*MusicCollection* という名前のテーブルのストリームにリソースベースのポリシーをアタッチします。この例では、ユーザー *John* に対して、該当ストリームで GetRecords、GetShardIterator、DescribeStream の API アクションを実行することを許可しています。

*######*のテキストを、リソース固有の情報に必ず置き換えてください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/John"
      },
      "Action": [
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeStream"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

リソースベースのポリシーをテーブルから削除する

DynamoDB コンソール、DeleteResourcePolicy API、AWS CLI、AWS SDK、または AWS CloudFormation テンプレートを使用して、既存のテーブルからリソースベースのポリシーを削除できます。

AWS CLI

次の例では、delete-resource-policy AWS CLI コマンドを使用して、*MusicCollection* という名前のテーブルからリソースベースのポリシーを削除しています。

*######*のテキストを、リソース固有の情報に必ず置き換えてください。

```
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

AWS Management Console

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. DynamoDB コンソールのダッシュボードで [テーブル] を選択し、既存のテーブルを選択します。
3. [アクセス許可] を選択します。
4. [ポリシーを管理] ドロップダウンから [ポリシーを削除] を選択します。

5. [テーブルについてのリソースベースのポリシーを削除] ダイアログボックスで、「**confirm**」と入力して削除アクションを確定します。
6. [削除] を選択します。

リソースベースのポリシーを使用したクロスアカウントアクセス

リソースベースのポリシーを使用して、異なる AWS アカウント で利用可能なリソースへのクロスアカウントアクセスを許可できます。リソースベースのポリシーで許可されているクロスアカウントアクセスについては、IAM Access Analyzer が該当リソースと同じ AWS リージョン にある場合は、IAM Access Analyzer で外部アクセスの検出結果としてすべて報告されます。IAM Access Analyzer は、IAM ポリシーの文法とベストプラクティスに照らしてポリシーチェックを行います。これらのチェックにより、機能的でセキュリティのベストプラクティスに準拠したポリシーを作成するのに、役立つ結果と実行可能なレコメンデーションが示されます。IAM Access Analyzer のアクティブな検出結果を DynamoDB コンソールの [アクセス許可] タブで表示できます。

IAM Access Analyzer を使用したポリシーの検証の詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。IAM Access Analyzer によって返される警告、エラー、および提案のリストを表示するには、「IAM Access Analyzer ポリシーチェックリファレンス」を参照してください。

アカウント A のユーザー A に、アカウント B のテーブル B にアクセスするための GetItem 権限を付与するには、以下の手順を実行します。

1. `GetItem` アクションの実行権限をユーザー A に付与するリソースベースのポリシーを、テーブル B にアタッチします。
2. テーブル B に対して `GetItem` アクションを実行する権限をユーザー A に付与するアイデンティティベースのポリシーを、ユーザー A にアタッチします。

DynamoDB コンソールにある [外部アクセスをプレビュー] オプションを使用すると、リソースへのパブリックアクセスとクロスアカウントアクセスに新しいポリシーがどのように影響するかをプレビューできます。ポリシーを保存する前に、新しい IAM Access Analyzer の結果が導入されているかどうかや、既存の結果を解決するかどうかを確認できます。アクティブなアナライザーが表示されない場合は、[Access Analyzer へ移動] を選択し、[IAM Access Analyzer] でアカウントアナライザーを作成します。詳細については、「アクセスのプレビュー」を参照してください。

DynamoDB のデータプレーンとコントロールプレーンの API のテーブル名パラメータには、クロスアカウントのオペレーションをサポートするため、テーブルの完全な Amazon リソースネーム

(ARN) を指定できます。完全な ARN ではなくテーブル名パラメータのみを指定した場合、API オペレーションは、リクエスタが属するアカウントの該当テーブルに対して実行されます。クロスアカウントアクセスを使用するポリシーの例については、「クロスアカウントアクセスのリソースベースのポリシー」を参照してください。

リソース所有者のアカウントは、別のアカウントのプリンシパルが所有者のアカウントの DynamoDB テーブルに対して読み書きを実行している場合にも課金されます。テーブルにプロビジョンドスループットが指定されている場合、所有者アカウントと他のアカウントのリクエスタからのすべてのリクエストの合計によって、リクエストをスロットリングするか (自動スケーリングが無効な場合)、スケールアップ/スケールダウンするか (自動スケーリングが有効な場合) が決まります。

リクエストは所有者アカウントとリクエスタアカウントの両方の CloudTrail ログに記録されるため、2 つのアカウントはそれぞれ、どちらのアカウントがどのデータにアクセスしたかを追跡できます。

ⓘ Note

コントロールプレーン API のクロスアカウントアクセスでは、1 秒あたりのトランザクション数 (TPS) の上限が低く、500 リクエストです。

リソースベースのポリシーでパブリックアクセスをブロックする

ブロックパブリックアクセス (BPA) は、Amazon Web Services (AWS) アカウント全体で DynamoDB のテーブル、インデックス、ストリームへのパブリックアクセスを許可するリソースベースのポリシーを特定し、アタッチされないように防ぐ機能です。BPA を使用すると、DynamoDB リソースへのパブリックアクセスを阻止できます。BPA はリソースベースのポリシーの作成時や変更時にチェックを行い、DynamoDB のセキュリティ体制の改善に貢献します。

BPA は自動推論を活用して、リソースベースのポリシーで許可されるアクセスを分析し、該当するアクセス許可がリソースベースのポリシーの管理時に見つかった場合は警告します。分析では、リソースベースのポリシーのステートメント、アクション、ポリシーで使用されている条件キーセットをすべてまたいでアクセスを検証します。

⚠ Important

BPA は、DynamoDB リソース (テーブル、インデックス、ストリームなど) に直接アタッチされたリソースベースのポリシーによってパブリックアクセスが許可されることがないよう

に防いで、リソースを保護します。BPA を利用したうえでさらに、次のポリシーを注意深く調べて、パブリックアクセスが許可されていないことを確かめてください。

- 関連する AWS プリンシパル (IAM ロールなど) にアタッチされているアイデンティティベースのポリシー
- 関連する AWS リソース (AWS Key Management Service (KMS) キーなど) にアタッチされているリソースベースのポリシー

プリンシパルに * エントリを含めないでください。または、指定された条件キーのいずれかでプリンシパルからリソースへのアクセスが制限されていることを確認する必要があります。リソースベースのポリシーが AWS アカウント 全体でテーブル、インデックス、またはストリームへのパブリックアクセスを許可している場合、ポリシー内の指定内容が修正され、非パブリックと判断されない限り、そのポリシーの作成または変更は阻止されます。

`Principal` ブロック内に 1 つ以上のプリンシパルを指定することで、ポリシーを非パブリックにすることができます。次のリソースベースのポリシーの例では、2 つのプリンシパルを指定することで、パブリックアクセスをブロックしています。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
}
```

特定の条件キーを指定してアクセスを制限するポリシーも、パブリックとは見なされません。リソースベースのポリシーで指定されているプリンシパルを評価するほかに、以下の信頼できる条件キーも、リソースベースのポリシーが付与するアクセスが非パブリックであるという評価の補足に使用できます。

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`

- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`
- `aws:Ec2InstanceSourceVpc`
- `aws:SourceOrgID`
- `aws:SourceOrgPaths`

さらに、リソースベースのポリシーが非パブリックとなるには、Amazon リソースネーム (ARN) と文字列キーの値にワイルドカードや変数が含まれていてはいけません。リソースベースのポリシーで `aws:PrincipalIsAWSService` キーが使用されている場合は、キー値を true に設定する必要があります。

次のポリシーでは、指定されたアカウントのユーザー `John` にアクセスが限定されています。この条件により `Principal` が制限されるため、パブリックとは見なされません。

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

次の例は非パブリックのリソースベースのポリシーです。`StringEquals` 演算子を使用して `sourceVPC` を制限しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

リソースベースのポリシーでサポートされる API オペレーション

このトピックでは、リソースベースのポリシーでサポートされる API オペレーションを示します。ただし、クロスアカウントアクセスの場合、リソースベースのポリシーで使用できる DynamoDB API は特定のセットに限られます。リソースベースのポリシーをリソースタイプ (バックアップやインポートなど) にアタッチすることはできません。これらのリソースタイプで動作する API に対応する IAM アクションは、リソースベースのポリシーでサポートされる IAM アクションからは除外されます。テーブル管理者は同一アカウント内で内部テーブル設定を構成するため、UpdateTimeToLive や DisableKinesisStreamingDestination などの API は、リソースベースのポリシーによるクロスアカウントアクセスに対応していません。

クロスアカウントアクセスに対応した DynamoDB データプレーン API とコントロールプレーン API では、テーブル名のオーバーロードにも対応しています。そのため、テーブル名の代わりにテーブル ARN を指定することができます。これらの API の TableName パラメータにテーブル ARN を指定できます。ただし、これらの API がすべてクロスアカウントアクセスに対応しているわけではありません。

次の表には、リソースベースのポリシーとクロスアカウントアクセスのサポート状況を API 別にまとめています。

API アクション	リソースベースのポリシーのサポート	クロスアカウントのサポート
Data Plane - Tables/indexes		
DeleteItem	Yes	Yes
GetItem	Yes	Yes
PutItem	Yes	Yes
Query	Yes	Yes
Scan	Yes	Yes
UpdateItem	Yes	Yes
TransactGetItems	Yes	Yes
TransactWriteItems	Yes	Yes
BatchGetItem	Yes	Yes
BatchWriteItem	Yes	Yes
PartiQL		
BatchExecuteStatement	Yes	No
ExecuteStatement	Yes	No
ExecuteTransaction	Yes	No
Control Plane - Tables		
CreateTable	No	No
DeleteTable	Yes	Yes
DescribeTable	Yes	Yes
UpdateTable	Yes	Yes

API アクション	リソースベースのポリシーのサポート	クロスアカウントのサポート
Version 2019.11.21 (Current) global tables		
DescribeTableReplicaAutoScaling	Yes	No
UpdateTableReplicaAutoScaling	Yes	No
Version 2017.11.29 (Legacy) global table		
CreateGlobalTable	No	No
DescribeGlobalTable	No	No
DescribeGlobalTableSettings	No	No
ListGlobalTables	No	No
UpdateGlobalTable	No	No
UpdateGlobalTableSettings	No	No
Tags		
ListTagsOfResource	Yes	Yes
TagResource	Yes	Yes
UntagResource	Yes	Yes
Backup/Restore		
CreateBackup	Yes	No
DescribeBackup	No	No
DeleteBackup	No	No
RestoreTableFromBackup	No	No

API アクション	リソースベースのポリシーのサポート	クロスアカウントのサポート
Continuous Backup/Restore (PITR)		
DescribeContinuousBackups	Yes	No
RestoreTableToPointInTime	Yes	No
UpdateContinuousBackups	Yes	No
Contributor Insights		
DescribeContributorInsights	Yes	No
ListContributorInsights	No	No
UpdateContributorInsights	Yes	No
Export		
DescribeExport	No	No
ExportTableToPointInTime	Yes	No
ListExports	No	No
Import		
DescribeImport	No	No
ImportTable	No	No
ListImports	No	No
Kinesis		
DescribeKinesisStreamingDestination	Yes	No
DisableKinesisStreamingDestination	Yes	No

API アクション	リソースベースのポリシーのサポート	クロスアカウントのサポート
EnableKinesisStreamingDestination	Yes	No
UpdateKinesisStreamingDestination	Yes	No
Resource policies		
GetResourcePolicy	Yes	No
PutResourcePolicy	Yes	No
DeleteResourcePolicy	Yes	No
Time-to-Live		
DescribeTimeToLive	Yes	No
UpdateTimeToLive	Yes	No
Others		
DescribeLimits	No	No
DescribeEndpoint	No	No
ListBackups	No	No
ListTables	No	No

次の表には、リソースベースのポリシーとクロスアカウントアクセスのサポート状況を DynamoDB Streams API 別にまとめています。

API アクション	リソースベースのポリシーのサポート	クロスアカウントのサポート
DescribeStream	はい	はい

API アクション	リソースベースのポリシーのサポート	クロスアカウントのサポート
GetRecords	はい	はい
GetShardIterator	はい	はい
ListStreams	いいえ	いいえ

IAM アイデンティティベースのポリシーと DynamoDB リソースベースのポリシーによる認証

アイデンティティベースのポリシーはアイデンティティ (IAM ユーザー、ユーザーグループ、ロールなど) にアタッチされます。これらは、アイデンティティが実行できるアクション、実行対象となるリソース、実行条件を制御する IAM ポリシードキュメントです。アイデンティティベースのポリシーには、管理ポリシーまたはインラインポリシーがあります。

リソースベースのポリシーは、リソース (DynamoDB テーブルなど) にアタッチする IAM ポリシードキュメントです。これらのポリシーでは、そのリソースに対して特定のアクションを実行するために指定されたプリンシパルのアクセス許可を付与するとともに、このアクセス許可が適用される条件を定義します。例えば、DynamoDB テーブルのリソースベースのポリシーには、そのテーブルに関連付けられたインデックスも含まれます。リソースベースのポリシーはインラインポリシーです。マネージド型のリソースベースのポリシーはありません。

これらのポリシーの詳細については、「IAM ユーザーガイド」の「アイデンティティベースおよびリソースベースのポリシー」を参照してください。

IAM プリンシパルがリソース所有者と同じアカウントに属している場合、そのリソースへのアクセス権限の指定には、リソースベースのポリシーで事足ります。さらに、IAM アイデンティティベースのポリシーをリソースベースのポリシーと併せて用意しておくこともできます。クロスアカウントアクセスの場合は、アイデンティティポリシーとリソースポリシーの両方でアクセスを明示的に許可する必要があります。詳細については、「リソースベースのポリシーを使用したクロスアカウントアクセス」に記載されています。両タイプのポリシーを併用する場合、「アカウント内でのリクエストの許可または拒否の決定」の解説通りにポリシーが評価されます。

リソースベースのポリシーの例

リソースベースのポリシーの `Resource` フィールドで ARN を指定した場合は、その ARN がアタッチ先の DynamoDB リソースの ARN と一致する場合にのみ、ポリシーが有効になります。

> **Note**
>
> *######*のテキストを、リソース固有の情報に必ず置き換えてください。

テーブルのリソースベースのポリシー

次のリソースベースのポリシーは、*MusicCollection* という名前の DynamoDB テーブルにアタッチされています。IAM ユーザーの *John* と *Jane* に対して、*MusicCollection* リソースで GetItem アクションと BatchGetItem アクションを実行する権限を付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/John",
            "arn:aws:iam::111122223333:user/Jane"
          ]
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
          "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

ストリームのリソースベースのポリシー

次のリソースベースのポリシーは、2024-02-12T18:57:26.492 という名前の DynamoDB ストリームにアタッチされています。IAM ユーザーの *John* と *Jane* に対して、2024-02-12T18:57:26.492 リソースで GetRecords、GetShardIterator、DescribeStream の API アクションを実行する権限を付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/John",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

指定されたリソースに対するあらゆるアクションの実行権限を付与するリソースベースのポリシー

テーブルと、テーブルに関連付けられたすべてのインデックスに対するすべてのアクションの実行をユーザーに許可する場合は、ワイルドカード (*) を使用して、テーブルに関連するアクションとリソースを表すことができます。リソースにワイルドカード文字を使用した場合、該当する DynamoDB テーブルとそのすべての関連インデックス (未作成のものも含む) へのアクセスがユー

ザーに許可されます。例えば、次のポリシーは、ユーザー *John* に対して、*MusicCollection* テーブルとそのすべてのインデックス (今後作成されるインデックスを含む) に対するあらゆるアクションの実行権限を付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal":"arn:aws:iam::111122223333:user/John",
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/index/*"
      ]
    }
  ]
}
```

クロスアカウントアクセスのリソースベースのポリシー

クロスアカウントの IAM アイデンティティに対して、DynamoDB リソースへのアクセス権限を指定できます。例えば、信頼できるアカウントのユーザーが、特定の項目とその項目内の特定の属性のみにアクセスするという条件で、テーブルの内容の読み取り権限を取得できるようにする場合が考えられます。次のポリシーでは、信頼できる AWS アカウント ID *111111111111* のユーザー *John* に対して、GetItem API を使用してアカウント *123456789012* のテーブルのデータにアクセスすることを許可します。このポリシーでは、プライマリキーが *Jane* の項目のみに該当ユーザーがアクセスして、属性 `Artist` と `SongTitle` のみを取得可能であり、その他の属性は取得できないようになっています。

> ⚠ Important
>
> SPECIFIC_ATTRIBUTES 条件を指定しない場合は、返された項目のすべての属性が表示されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountTablePolicy",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:user/John"
            },
            "Action": "dynamodb:GetItem",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

前述のリソースベースのポリシーに加えて、クロスアカウントアクセスを実現するためには、ユーザー *John* にアタッチしたアイデンティティベースのポリシーでも `GetItem` API アクションを許可する必要があります。以下は、ユーザー *John* にアタッチする必要があるアイデンティティベースのポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

ユーザー John は、アカウント *123456789012* のテーブル *MusicCollection* にアクセスするため、table-name パラメータにテーブル ARN を指定して GetItem リクエストを行うことができます。

```
aws dynamodb get-item \
    --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --key '{"Artist": {"S": "Jane"}' \
    --projection-expression 'Artist, SongTitle' \
    --return-consumed-capacity TOTAL
```

IP アドレス条件を指定したリソースベースのポリシー

条件を適用して、発信元の IP アドレス、仮想プライベートクラウド (VPC)、VPC エンドポイント (VPCE) を制限できます。リクエストの発信元のアドレスに基づいてアクセス許可を指定できます。例えば、特定の IP ソース (企業の VPN エンドポイントなど) がアクセス元である場合に限り、ユーザーに DynamoDB リソースへのアクセスを許可したい場合が考えられます。これらの IP アドレスを Condition ステートメントに指定します。

次の例では、ユーザー *John* に対して、発信元 IP が 54.240.143.0/24 および 2001:DB8:1234:5678::/64 である場合に任意の DynamoDB リソースへのアクセスを許可します。

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal":"arn:aws:iam::111111111111:user/John",
      "Action":"dynamodb:*",
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

また、発信元が特定の VPC エンドポイント (*vpce-1a2b3c4d* など) 以外である場合に、DynamoDB リソースへのアクセスを一切拒否することもできます。

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

IAM ロールを使用するリソースベースのポリシー

リソースベースのポリシーで IAM サービスロールを指定することもできます。このロールを引き受ける IAM エンティティは、そのロールに指定されているアクションのみを、リソースベースのポリシーで指定されているリソースセットに限定して実行できます。

次の例では、IAM エンティティが *MusicCollection* と *MusicCollection* の DynamoDB リソースですべての DynamoDB アクションを実行することを許可しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/John" },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

リソースベースのポリシーの考慮事項

DynamoDB リソースに対してリソースベースのポリシーを定義する際には、以下の点を考慮してください。

一般的な考慮事項

- リソースベースのポリシードキュメントでサポートされる最大サイズは 20 KB です。DynamoDB では、この上限に照らしてポリシーのサイズを計算する際に空白はカウントされません。
- 特定のリソースのポリシーの更新が 2 回目以降の場合、同じリソースに対するそのポリシーの前回の更新が正常に終わってから 15 秒間は更新できません。
- 現時点では、リソースベースのポリシーは既存のストリームにのみアタッチできます。作成中のストリームにポリシーをアタッチすることはできません。

グローバルテーブルに関する考慮事項

- リソースベースのポリシーは、グローバルテーブルバージョン 2017.11.29 (レガシー) のレプリカではサポートされていません。
- リソースベースのポリシー内で、グローバルテーブルのデータをレプリケートするアクションが DynamoDB のサービスにリンクされたロール (SLR) に対して拒否されている場合、レプリカの追加または削除はエラーで失敗します。
- AWS::DynamoDB::GlobalTable リソースでは、スタックの更新を行うリージョン以外のリージョンでは、スタックの更新時にレプリカの作成と、そのレプリカへのリソースベースのポリシーの追加を同時に行うことはできません。

クロスアカウントに関する考慮事項

- リソースベースのポリシーを使用したクロスアカウントアクセスでは、AWS マネージドキーで暗号化したテーブルはサポートされません。AWS KMS のマネージドポリシーへのクロスアカウントアクセスは許可できないからです。

AWS CloudFormation に関する考慮事項

- リソースベースのポリシーはドリフト検出には対応していません。AWS CloudFormation スタックテンプレートの外部でリソースベースのポリシーを更新した場合は、CloudFormation スタックに変更内容を反映させる必要があります。
- リソースベースのポリシーは、アウトオブバンドの変更に対応していません。CloudFormation テンプレートの外部でポリシーを追加、更新、または削除しても、テンプレート内のポリシーに変更がない限り、変更は上書きされません。

  例えば、テンプレートに含まれているリソースベースのポリシーを、後からテンプレートの外部で更新したとします。テンプレート内のポリシーを変更しない限り、DynamoDB 内の更新済みポリシーはテンプレート内のポリシーと同期されません。

  逆に、テンプレートにリソースベースのポリシーは含まれていないが、テンプレートの外部でポリシーを追加したとします。このポリシーは、テンプレートに追加しない限りは、DynamoDB から削除されることはありません。テンプレートにポリシーを追加してスタックを更新すると、DynamoDB の既存のポリシーが、テンプレートで定義されているポリシーと一致するように更新されます。

リソースベースのポリシーに関するベストプラクティス

このトピックでは、DynamoDB リソースに対するアクセス許可と、それらのリソースに対して実行できるアクションを定義する際のベストプラクティスについて説明します。

DynamoDB リソースへのアクセス制御を簡素化する

DynamoDB リソースにアクセスする必要がある AWS Identity and Access Management プリンシパルがリソース所有者と同じ AWS アカウント に属している場合、プリンシパルごとに IAM アイデンティティベースポリシーを用意する必要はありません。該当するリソースにリソースベースのポリシーをアタッチすれば十分です。このように構成すれば、アクセス制御が簡単になります。

リソースベースのポリシーで DynamoDB リソースを保護する

DynamoDB のすべてのテーブルとストリームに対してリソースベースのポリシーを作成し、これらのリソースへのアクセスを制御します。リソースベースのポリシーのおかげで、アクセス許可をリソース単位で一元管理し、DynamoDB のテーブル、インデックス、ストリームへのアクセス制御を簡素化し、管理オーバーヘッドを削減できます。リソースベースのポリシーがテーブルまたはストリームに対して指定されていない場合、そのテーブルやストリームへのアクセスは、IAM プリンシパルに関連付けられたアイデンティティベースのポリシーでアクセスが許可されていない限り、暗黙的に拒否されます。

最小特権アクセス許可を適用する

DynamoDB リソースに対するリソースベースのポリシーでアクセス許可を設定するときは、アクションの実行に必要なアクセス許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。その際、ワークロードやユースケースに必要なアクセス許可を検討しながら、大まかなアクセス許可から始めるとよいでしょう。ユースケースが成熟してきたら、最小特権になるように付与する権限を減らしていくことができます。

最小特権ポリシーの生成のためにクロスアカウントアクセスアクティビティを分析する

IAM Access Analyzer は、リソースベースのポリシーで指定された外部エンティティへのクロスアカウントアクセスを報告します。また、情報が可視化されるため、アクセス許可を調整し、最小特権の原則を守るうえでも役立ちます。ポリシー生成の詳細については、「IAM Access Analyzer ポリシーの生成」を参照してください。

IAM Access Analyzer を使用して最小特権ポリシーを生成する

タスクの実行に必要なアクセス許可のみを付与するには、AWS CloudTrail にログインしているアクセスアクティビティに基づいてポリシーを生成することができます。IAM Access Analyzer は、ポリシーで使用されるサービスとアクションを分析します。

DynamoDB におけるデータ保護

Amazon DynamoDB は、ミッションクリティカルで重要なデータストレージのために設計された、高い耐久性を備えたストレージインフラストラクチャです。冗長性を確保するために、データは、同一の Amazon DynamoDB リージョン内の複数の施設に分散した複数のデバイスに保存されます。

DynamoDB では、ユーザーデータは保管時に保護されるほか、オンプレミスのクライアントと DynamoDB 間の転送中のデータ、DynamoDB と同じ AWS リージョン内のその他の AWS リソース間の転送中のデータも保護されます。

トピック

- 保管時の DynamoDB 暗号化
- DynamoDB Accelerator でのデータ保護
- インターネットトラフィックのプライバシー

保管時の DynamoDB 暗号化

Amazon DynamoDB に保存されているすべてのユーザーデータは、保管時に完全に暗号化されます。DynamoDBの保管時の暗号化は、AWS Key Management Service (AWS KMS) に保存されている暗号化キーを使用してすべての保管中のデータを暗号化することにより高度なセキュリティを提供します。この機能は、機密データの保護における負担と複雑な作業を減らすのに役立ちます。保管時に暗号化することで、セキュリティを重視したアプリケーションを構築して、暗号化のコンプライアンスと規制の厳格な要件を満たすことができます。

DynamoDB の保管時の暗号化は、データを堅牢なメディアに保存されている場合は常に暗号化されたテーブル内にデータを配置して保護することにより、プライマリキー、ローカルおよびグローバルセカンダリインデックス、ストリーム、グローバルテーブル、バックアップ、DynamoDB Accelerator (DAX) クラスターなどに対する追加のデータ保護レイヤーを提供します。組織のポリシー、業界や政府の規制、またはコンプライアンス要件によって、アプリケーションのデータセキュリティを高めるために保管時の暗号化の使用が求められることがあります。

保管時の暗号化には、テーブルの暗号化に使用される暗号化キーを管理するための AWS KMS が統合されます。キーの種類と詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS Key Management Service の概念」を参照してください。

新しいテーブルを作成する場合、以下のいずれかの AWS KMS key の種類を選択してテーブルを暗号化できます。キーの種類は、いつでも切り替えることができます。

- AWS 所有のキー – デフォルトの暗号化タイプ。キーは DynamoDB により所有されます (追加料金なし)。
- AWS マネージドキー – キーはアカウントに保存され、AWS KMS によって管理されます (AWS KMS の料金が適用されます)。
- カスタマーマネージド型キー – キーはお客様のアカウントに保存され、ユーザーが作成、所有、管理します。ユーザーには KMS キーに対するフルコントロールの権限があります (AWS KMS の料金が適用されます)。

キーの種類の詳細については、「カスタマーキーと AWS キー」を参照してください。

ⓘ Note

- 保存時の暗号化を有効にして新しい DAX クラスターを作成する場合、AWS マネージドキー を使用して、クラスター内の保管中のデータを暗号化します。
- テーブルにソートキーが存在する場合、範囲の境界線を示すソートキーの一部が、プレーンテキスト形式でテーブルメタデータに保存されます。

暗号化されたテーブルにアクセスすると、DynamoDB はテーブルデータを透過的に復号化します。暗号化されたテーブルの使用あるいは管理のためにコードやアプリケーションを変更する必要はありません。DynamoDB は、期待される 1 桁ミリ秒のレイテンシーを継続的に提供し、すべての DynamoDB クエリは暗号化されたデータでシームレスに機能します。

新しいテーブルを作成する際に暗号化キーを指定したり、AWS Management Console、AWS Command Line Interface (AWS CLI)、Amazon DynamoDB API を使用して既存のテーブルで暗号化キーを切り替えたりすることができます。この方法については、「DynamoDB での暗号化テーブルの管理」を参照してください。

AWS 所有のキー を使用した保管時の暗号化に追加の料金はかかりません。ただし、AWS KMS 料金が AWS マネージドキー およびカスタマーマネージドキーに適用されます。料金の詳細については、「AWS KMS の料金」を参照してください。

DynamoDB の保管時の暗号化は、AWS 中国 (北京)、AWS 中国 (寧夏)、AWS GovCloud (米国) のリージョンを含むすべての AWS リージョンで利用できます。詳細については、「保存時の暗号化: 仕組み」および「DynamoDB の保管時の暗号化の使用に関する注意事項」を参照してください。

保存時の暗号化: 仕組み

Amazon DynamoDB の保管時の暗号化では、256 ビットの Advanced Encryption Standard (AES-256) を使用してデータの暗号化が行われるので、基盤となるストレージへの不正アクセスからデータを保護できます。

保管データ暗号化には、テーブルの暗号化に使用される暗号化キーを管理するための AWS Key Management Service (AWS KMS) が統合されます。

> Note
>
> 2022 年 5 月、AWS KMS は AWS マネージドキー のローテーションスケジュールを 3 年 (約 1,095 日間隔) ごとから毎年 (約 365 日間隔) に変更しました。
> 新しい AWS マネージドキーは、作成日から 1 年後に自動的にローテーションされ、それ以降はほぼ 1 年ごとにローテーションされます。
> 既存の AWS マネージドキー は、直近のローテーションから 1 年後にローテーションされ、その後毎年ローテーションされます。

AWS 所有のキー

AWS 所有のキー はお客様の AWS アカウントに保存されません。これらは、複数の AWS アカウントで使用するために AWS が所有および管理している KMS キーのコレクションの一部です。AWS のサービスでは、データの保護に AWS 所有のキー を使用できます。DynamoDB で使用する AWS 所有のキーは毎年 (約 365 日ごとに) ローテーションされます。

AWS 所有のキー は表示、管理、使用することはできず、その使用を監視することもできません。ただし、データを暗号化するキーを保護するための作業やプログラムを操作したり変更したりする必要はありません。

AWS 所有のキー のご利用に関しては、月額料金や使用料金は請求されません。また、アカウントの AWS KMS クォータにも影響しません。

AWS マネージドキー

AWS マネージドキー は、お客様のアカウントにある KMS キーであり、AWS KMS と統合されている AWS のサービスがお客様に代わって作成、管理、使用します。アカウントで AWS マネージドキー を表示して、キーポリシーを表示し、AWS CloudTrail ログでその使用を監査できます。ただし、これらの KMS キーを管理したり許可を変更したりすることはできません。

保管時の暗号化は、テーブルの暗号化に使用される DynamoDB (`aws/dynamodb`) 用の AWS マネージドキー を管理するために AWS KMS と自動的に統合されます。暗号化された DynamoDB テーブルを作成したときにAWS マネージドキー が存在しない場合、AWS KMS は自動的に新しいキーを作成します。このキーは、この先作成するテーブルの暗号化に使用されます。AWS KMS は、安全で可用性の高いハードウェアとソフトウェアを組み合わせて、クラウド向けに拡張されたキー管理システムを提供します。

AWS マネージドキー の許可を管理する方法の詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS マネージドキー の使用の承認」を参照してください。

カスタマーマネージドキー

カスタマーマネージドキーは、お客様が作成、所有、管理する AWS アカウントの KMS キーです。この KMS キーでは、キーポリシー、IAM ポリシー、および許可の確立と管理、有効化と無効化、暗号化対象のローテーション、タグの追加、KMS キーを参照するエイリアスの作成、削除スケジュールの設定などを完全に制御することができます。カスタマーマネージドキーの許可を管理する方法の詳細については、「カスタマーマネージドキーポリシー」を参照してください。

カスタマーマネージドキーをテーブルレベルの暗号化キーとして指定すると、DynamoDB テーブル、ローカルおよびグローバルセカンダリインデックス、およびストリームは、同じカスタマーマネージドキーで暗号化されます。オンデマンド Backup は、Backup の作成時に指定されたテーブルレベルの暗号化キーを使用して暗号化されます。テーブルレベルの暗号化キーを更新しても、既存のオンデマンド Backup に関連付けられている暗号化キーは変更されません。

カスタマーマネージドキーの状態を無効に設定するか、削除のスケジュールを設定すると、すべてのユーザーと DynamoDB サービスは、データの暗号化と復号化、およびテーブルに対する読み取り/書き込み操作を実行できなくなります。テーブルに対するアクセスを維持し、データ損失を防止するには、DynamoDB が暗号化キーにアクセスできる必要があります。

カスタマーマネージドキーを無効化したり、削除をスケジュールしたりすると、テーブルステータスはアクセス不能になります。テーブルの操作を続行できるようにするには、指定された暗号化キーへの DynamoDB アクセスを 7 日以内に提供する必要があります。暗号化キーにアクセスできないことが検出されると、DynamoDB から警告メール通知が送信されます。

> Note
>
> - DynamoDB サービスがカスタマーマネージドキーに 7 日以上アクセスできない場合、テーブルはアーカイブされてアクセスできなくなります。DynamoDB は、テーブルのオンデマンド Backup を作成し、それに対して課金されます。このオンデマンド Backup を使用して、データを新しいテーブルに復元できます。復元を開始するには、最後に使用したカスタマーマネージドキーをテーブルで有効にし、DynamoDB からのアクセスを確立します。
> - グローバルテーブルレプリカの暗号化に使用したカスタマーマネージドキーにアクセスできない場合、DynamoDB は、このレプリカをレプリケーショングループから除外します。レプリカは削除されず、このリージョンに対するレプリケーションは、カスタマーマネージドキーにアクセス不能と検出されてから 20 時間後に停止します。

詳細については、「キーの有効化」と「キーの削除」を参照してください。

AWS マネージドキー の使用に関する注意事項

AWS KMS アカウントに保存されている KMS キーにアクセスできない場合、Amazon DynamoDB はテーブルデータを読み取れません。DynamoDB は、エンベロープ暗号化とキー階層を使用してデータを暗号化します。AWS KMS 暗号化キーは、このキー階層のルートキーを暗号化するために使用されます。詳細については、「AWS Key Management Service デベロッパーガイド」の「Envelope encryption」(エンベロープ暗号化) を参照してください。

AWS CloudTrail および Amazon CloudWatch Logs を使用して、DynamoDB がお客様に代わって AWS KMS に送信するリクエストを追跡できます。詳細については、「AWS Key Management Service デベロッパーガイド」の「DynamoDB と AWS KMS の相互作用のモニタリング」を参照してください。

DynamoDB は、DynamoDB オペレーションごとに AWS KMS を呼び出すわけではありません。キーは、アクティブなトラフィックを持つ呼び出しごとに 5 分に 1 回更新されます。

SDK が接続を再利用するように設定されていることを確認してください。そうしないと、DynamoDB は、オペレーションごとに新しい AWS KMS キャッシュエントリを再確立しなければならなくなるのでレイテンシーが発生します。さらに、AWS KMSと CloudTrail のコストが上がってしまう可能性もあります。たとえば、Node.js SDK を使用してこれを行うには、`keepAlive` を有効にした状態で新しい HTTPS エージェントを作成することができます。詳細については、「AWS SDK for JavaScript デベロッパーガイド」の「Node.js での keepAlive の設定」を参照してください。

DynamoDB の保管時の暗号化の使用に関する注意事項

Amazon DynamoDB で保管時の暗号化を使用する場合は、以下の点を考慮してください。

すべてのテーブルデータの暗号化

保管時のサーバー側の暗号化は、すべての DynamoDB テーブルデータで有効になり、無効にできません。テーブル内の項目のサブセットのみを暗号化することはできません。

保管時の暗号化は、永続的ストレージメディアの静的 (保管時) のデータのみを暗号化します。転送中のデータあるいは使用中のデータでデータの安全性が考慮される場合には、追加の対策を実行する必要がある場合があります。

- 転送中のデータ: DynamoDB のすべてのデータは転送時に暗号化されます。デフォルトでは、DynamoDB との通信において、Secure Sockets Layer (SSL)/Transport Layer Security (TLS) 暗号化を使用してネットワークトラフィックを保護する HTTPS プロトコルが使用されます。
- 使用中のデータ: DynamoDB 送信する前のデータを保護するには、クライアント側の暗号化を使用します。詳細については、「Amazon DynamoDB Encryption Client デベロッパーガイド」の「クライアント側とサーバー側の暗号化」を参照してください。

暗号化されたテーブルでストリーミングを使用できます。DynamoDB Streams は、テーブルレベルの暗号化キーを使用して常に暗号化されます。詳細については、「DynamoDB Streams の変更データキャプチャ」を参照してください。

DynamoDB バックアップが暗号化され、バックアップから復元されたテーブルでも暗号化が有効になります。バックアップデータの暗号化に、AWS 所有のキー、AWS マネージドキー、またはカスタマーマネージドキーを使用できます。詳細については、「DynamoDB のオンデマンドバックアップおよび復元の使用」を参照してください。

ローカルセカンダリインデックスおよびグローバルセカンダリインデックスは、ベーステーブルと同じキーを使用して暗号化されます。

暗号化タイプ

> Note
>
> カスタマーマネージドキーは、グローバルテーブルバージョン 2017 ではサポートされていません。DynamoDB グローバルテーブルでカスタマーマネージドキーを使用する場合は、

テーブルをグローバルテーブルバージョン 2019 にアップグレードしてから有効にする必要があります。

AWS Management Console では、AWS マネージドキー またはカスタマーマネージドキーを使用してデータを暗号化すると、暗号化タイプは KMS になります。AWS 所有のキー を使用すると、暗号化タイプは DEFAULT になります。Amazon DynamoDB API では、AWS マネージドキー またはカスタマーマネージドキーを使用すると暗号化タイプは KMS になります。暗号化タイプがない場合、データは AWS 所有のキー を使用して暗号化されます。AWS 所有のキー、AWS マネージドキー、カスタマーマネージドキーはいつでも切り替えることができます。コンソール、AWS Command Line Interface (AWS CLI)、または Amazon DynamoDB API を使用して、暗号化キーを切り替えることができます。

カスタマーマネージドキーを使用する場合、次の制限に注意してください。

- DynamoDB アクセラレーター (DAX) クラスターではカスタマーマネージドキーは使用できません。詳細については、「保管時の DAX 暗号化」を参照してください。
- カスタマーマネージドキーを使用して、トランザクションを使用するテーブルを暗号化できます。ただし、トランザクションの伝達の堅牢性を確保するために、トランザクションリクエストのコピーはサービスによって一時的に保存され、AWS 所有のキー を使用して暗号化されます。テーブルとセカンダリインデックスのコミット済みデータは、カスタマーマネージドキーを使用して常に保管時に暗号化されます。
- カスタマーマネージドキーを使用して、Contributor Insights を使用するテーブルを暗号化できます。ただし、Amazon CloudWatch に送信されるデータは、AWS 所有のキー を使用して暗号化されます。
- 新しいカスタマーマネージドキーに移行する場合は、プロセスが完了するまで元のキーを有効にしておいてください。AWS では、新しいキーでデータを暗号化する前に、元のキーを使用してデータを復号化する必要があります。テーブルの SSEDescription ステータスが有効になり、新しいカスタマーマネージドキーの KMSmasterKeyArn が表示されると、プロセスは完了します。この時点で、元のキーを無効にするか、削除のスケジュールを設定できます。
- 新しいカスタマー管理のキーが表示されると、テーブルと新しいオンデマンドバックアップが新しいキーで暗号化されます。
- 既存のオンデマンドバックアップは、バックアップの作成時に使用されたカスタマー管理のキーで暗号化されたままになります。これらのバックアップを復元するには、同じキーが必要です。DescribeBackup API を使用してバックアップの SSEDescription を表示することで、各バックアップが作成された期間のキーを識別できます。

- カスタマーマネージドキーを無効化した場合、または削除をスケジュールした場合、DynamoDB Streams 内のデータは 24 時間保持されます。作成後 24 時間を超えた未取得のアクティビティデータはすべて、トリミングの対象となります。
- カスタマーマネージドキーを無効化した場合、または削除をスケジュールした場合、有効期限 (TTL) は 30 分間続きます。これらの TTL 削除は引き続き DynamoDB Streams に出力され、標準のトリミングまたは保持間隔が適用されます。

  詳細については、「キーの有効化」と「キーの削除」を参照してください。

KMS キーとデータキーを使用する

DynamoDB の保存時の暗号化機能は、AWS KMS key およびデータキーの階層を使用してテーブルのデータを保護します。DynamoDB は、同じキー階層を使用して、DynamoDB Streams、グローバルテーブル、およびバックアップが耐久性のあるメディアに書き込まれるときに保護します。

DynamoDB にテーブルを実装する前に、暗号化戦略を計画することをお勧めします。機密データまたは機密データを DynamoDB に保存する場合は、クライアント側の暗号化をプランに含めることを検討してください。これにより、データをできるだけ送信元に近い状態で暗号化し、ライフサイクル全体にわたってデータを確実に保護できます。詳細については、「DynamoDB 暗号化クライアント」ドキュメントを参照してください。

AWS KMS key

保存時の暗号化機能は、AWS KMS key の DynamoDB テーブルを保護します。デフォルトでは、DynamoDB は AWS 所有のキー (DynamoDB サービスアカウントで作成および管理されるマルチテナントキー) を使用します。ただし、DynamoDB テーブルは、カスタマーマネージドキーまたは AWS アカウント の DynamoDB (`aws/dynamodb`) 用で暗号化できます。テーブルごとに異なる KMS キーを選択できます。テーブル用に選択した KMS キーも、ローカルおよびグローバルのセカンダリインデックス、ストリーム、バックアップの暗号化に使用されます。

テーブルを作成または更新するときは、テーブル用の KMS キーを選択します。テーブル用 KMS キーは、DynamoDB コンソールで、または UpdateTable オペレーションを使用していつでも変更できます。キーの切り替えプロセスはシームレスであり、ダウンタイムやサービスの低下を必要としません。

> ⚠ Important
>
> DynamoDB は、対称 KMS キーのみをサポートします。非対称 KMS キーを使用して DynamoDB テーブルを暗号化することはできません。

カスタマーマネージドキーを使用して次の機能を取得します。

- KMS キーを作成および管理します。これには、キーポリシーおよび IAM ポリシーの設定、KMS キーへのアクセスを制御するグラントが含まれます。KMS キーの有効化または無効化、自動キーローテーションの有効化または無効化、使用しなくなった KMS キーの削除を実行できます。
- インポートされたキーマテリアルを持つカスタマーマネージドキー、またはユーザーが所有して管理するカスタムキーストアで、カスタマーマネージドキーを使用できます。
- DynamoDB テーブルの暗号化と復号を監査するには、 AWS CloudTrail ログで AWS KMS への DynamoDB API コールを調べます。

次のいずれかの機能が必要なときは、AWS マネージドキー を使用します。

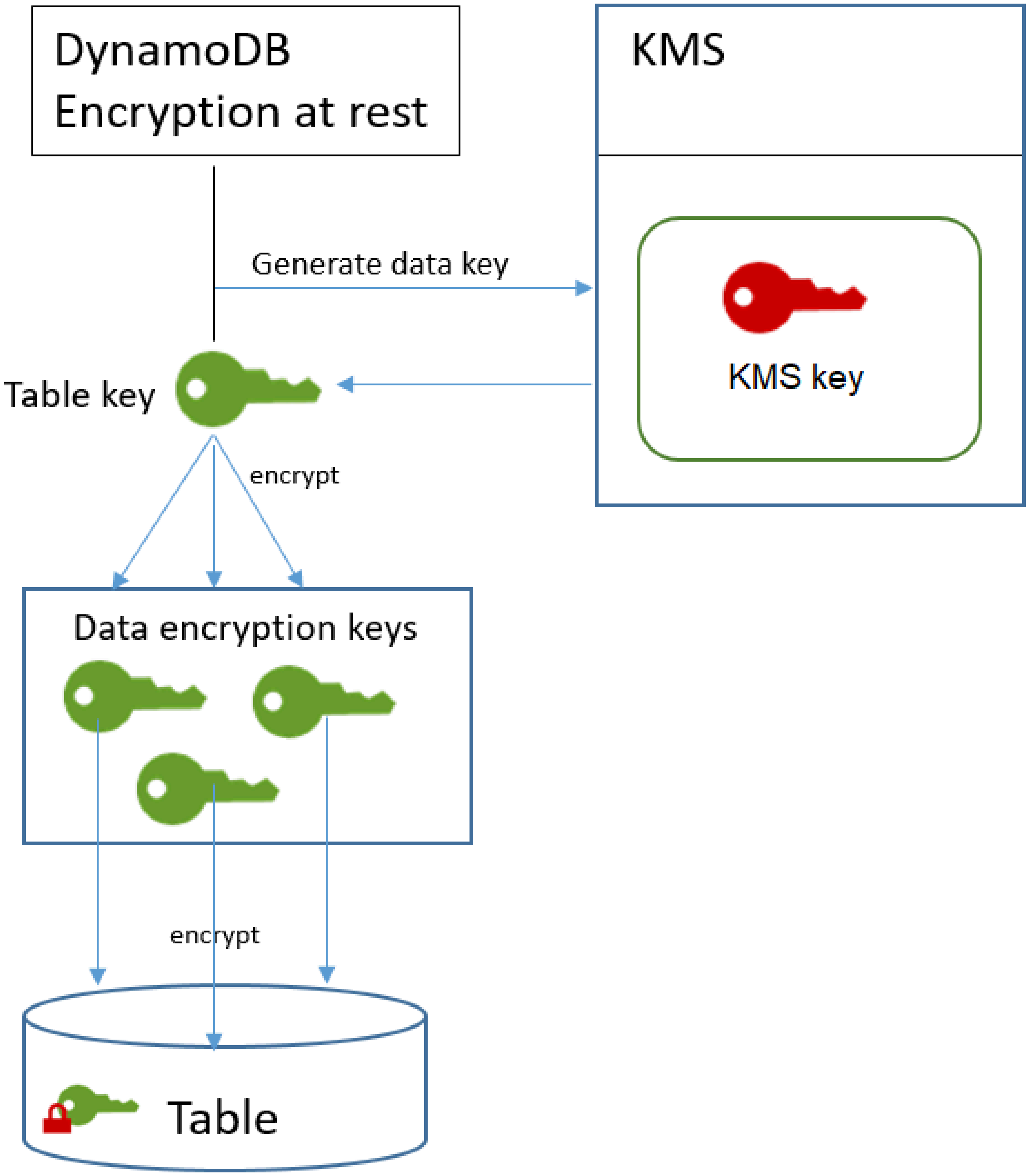
- KMS キーを表示し、そのキーポリシーを表示できます。(キーポリシーの変更はできません)。
- DynamoDB テーブルの暗号化と復号を監査するには、 AWS CloudTrail ログで AWS KMS への DynamoDB API コールを調べます。

ただし、AWS 所有のキー は無料で、その使用は AWS KMS リソースクォータまたはリクエストクォータに対してカウントされません。カスタマーマネージドキーおよび AWS マネージドキーには、API コールごとに料金が発生し、これらの KMS キーには AWS KMS クォータが適用されます。

テーブルキー

DynamoDB は、テーブルの KMS キーを使用して、テーブルキーと呼ばれるテーブルの一意のデータキーを生成して暗号化します。テーブルキーは、暗号化されたテーブルの存続期間中は保持されます。

テーブルキーは、キー暗号化キーとして使用されます。DynamoDB は、このテーブルキーを使用して、テーブルデータの暗号化に使用されるデータ暗号化キーを保護します。DynamoDB は、テーブル内の基礎となる各構造に対して一意のデータ暗号化キーを生成しますが、複数のテーブル項目が同じデータ暗号化キーで保護される場合があります。



暗号化されたテーブルに最初にアクセスすると、DynamoDB は、KMS キーを使用してテーブルキーを復号するリクエストを AWS KMS に送信します。その後、プレーンテキストテーブルキーを使用してデータ暗号化キーを復号化し、プレーンテキストデータ暗号化キーを使用してテーブルデータを復号化します。

DynamoDB は AWS KMS の外部でテーブルキーとデータ暗号化キーを保存して使用します。これによって、Advanced Encryption Standard (AES) 暗号化および 256 ビット暗号化キーのすべて

のキーが保護されます。続いて、暗号化されたキーを暗号化されたデータと一緒に保存します。これらのキーおよびデータは、必要なときにテーブルデータの暗号化に使用できます。

テーブルの KMS キーを変更すると、DynamoDB は新しいテーブルキーを生成します。次に、新しいテーブルキーを使用してデータ暗号化キーの再暗号化が行われます。

テーブルキーのキャッシュ

DynamoDB オペレーションごとに AWS KMS を呼び出さないように、DynamoDB は各呼び出しのプレーンテキストのテーブルキーをメモリにキャッシュします。DynamoDB では、キャッシュしたテーブルキーが 5 分間非アクティブ状態であった後にリクエストを取得すると、AWS KMS に新しいリクエストを送信してテーブルキーを復号します。この呼び出しは、テーブルキーの復号を求める前回のリクエスト以降に AWS KMS または AWS Identity and Access Management (IAM) で KMS キーのアクセスポリシーに加えられた変更をすべてキャプチャします。

KMS キーの使用を認可する

DynamoDB テーブルを保護するために、アカウントでカスタマーマネージドキーまたは AWS マネージドキー を使用する場合は、その KMS キーのポリシーが DynamoDB に、ユーザーに代わってキーを使用する許可を付与する必要があります。DynamoDB の AWS マネージドキー の認可コンテキストには、そのキーポリシーとそれを使用するアクセス許可を委任するグラントが含まれます。

AWS マネージドキー はアカウントにあり、そのポリシーとグラントを表示できるため、カスタマーマネージドキーのポリシーとグラントを完全に制御することができます。ただし、AWS によって管理されているため、ポリシーを変更することはできません。

DynamoDBでは、デフォルトの AWS 所有のキー を使用して AWS アカウント の DynamoDB テーブルを保護するための追加の認可は不要です。

トピック

- AWS マネージドキー のキーポリシー
- カスタマーマネージドキーのキーポリシー
- グラントを使用した DynamoDB の承認

AWS マネージドキー のキーポリシー

DynamoDB は、DynamoDB リソースにアクセスしているユーザーの代わりに、暗号化オペレーションで DynamoDB (aws/dynamodb) の AWS マネージドキー を使用します。AWS マネージドキー の

キーポリシーはアカウントのすべてのユーザーに、指定されたオペレーションで AWS マネージドキー を使用する許可を付与します。ただし、アクセス許可が付与されるのは、DynamoDB がユーザーの代わりにリクエストを行う場合のみです。キーポリシーの ViaService 条件では、リクエストが DynamoDB サービスで発生しない限り、ユーザーに AWS マネージドキー の使用を許可しません。

このキーポリシーは、すべての AWS マネージドキー のポリシーと同様に、AWS によって確立されます。このポリシーを変更することはできませんが、いつでも表示できます。詳細については、「Viewing a key policy」を参照してください。

このキーポリシーのポリシーステートメントには次の効果があります

- リクエストがユーザーの代わりに DynamoDB から送信された場合、アカウントのユーザーが DynamoDB の AWS マネージドキー を暗号化オペレーションで使用することを許可します。また、このポリシーはユーザーに KMS キーのグラントの作成も許可します。
- 認可された IAM がアカウントで DynamoDB の AWS マネージドキー のプロパティを表示し、DynamoDB の KMS キー使用を許可するグラントを取り消すことができるようにします。DynamoDB は、継続的なメンテナンス操作にグラントを使用します。
- DynamoDB が読み取り専用オペレーションを実行して、アカウントで DynamoDB の AWS マネージドキー を検索できるようにします。

```
{
  "Version" : "2012-10-17",
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the account
 that are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*",
 "kms:GenerateDataKey*", "kms:CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow DynamoDB Service with service principal name dynamodb.amazonaws.com
 to describe the key directly",
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "dynamodb.amazonaws.com"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
    "Resource" : "*"
  } ]
}
```

カスタマーマネージドキーのキーポリシー

カスタマーマネージドキーを選択して DynamoDB テーブルを保護する際、DynamoDB は、選択を行うプリンシパルの代わりに KMS キーを使用するアクセス許可を取得します。そのプリンシパル、ユーザー、ロールは、DynamoDB に必要な KMS キーに対するアクセス許可を持っている必要があります。これらの許可は、キーポリシー、IAM ポリシー、またはグラントで指定できます。

DynamoDB には、少なくとも、カスタマーマネージドキーに対する次のアクセス許可が必要です。

- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt* (for kms:ReEncryptFrom および kms:ReEncryptTo 向け)
- kms:GenerateDataKey* (for kms:GenerateDataKey および kms:GenerateDataKeyWithoutPlaintext 向け)
- kms:DescribeKey
- kms:CreateGrant

例えば、次のキーポリシーの例では、必要なアクセス許可のみを提供します。このポリシーには、以下の影響があります。

- DynamoDB が、DynamoDB の使用許可を持つアカウントのプリンシパルの代わりに動作している場合にのみ、DynamoDB が暗号化オペレーションで KMS キーを使用し、グラントを作成することを許可します。ポリシーステートメントで指定されたプリンシパルに DynamoDB を使用する権限がない場合、呼び出しは DynamoDB サービスからのものであっても失敗します。
- KMS: viaService 条件キーでは、ポリシ ーステートメントにリストされているプリンシパルの代わりに DynamoDB からリクエストが送信された場合にのみアクセス許可が許可されます。これらのプリンシパルは、これらのオペレーションを直接呼び出すことはできません。kms:ViaService の値である dynamodb.*.amazonaws.com は、リージョンの位置にアスタリスク (*) が付いていることに注意してください。DynamoDB では、クロスリージョン呼び出しを実行して DynamoDB グローバルテーブルをサポートできるように、特定の AWS リージョン から独立したアクセス許可が必要です。
- KMS キー管理者 (db-team ロールを引き受けることができるユーザー) に、KMS キーへの読み取り専用アクセス許可、およびグラント (テーブルを保護するために DynamoDB が必要とするグラントを含む) を取り消す許可を付与します。

サンプルキーポリシーを使用する前に、サンプルプリンシパルを AWS アカウント の実際のプリンシパルに置き換えます。

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the account
 that are authorized to use Amazon DynamoDB",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the KMS key and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

グラントを使用した DynamoDB の承認

キーポリシーに加えて、DynamoDB は DynamoDB (aws/dynamodb) のためにカスタマーマネージドキーまたは AWS マネージドキー の許可を設定するためのグラントを使用します。アカウントの KMS キーのグラントを表示するには、ListGrants 操作を使用します。DynamoDB では、AWS 所有のキー を使用してテーブルを保護するために、グラントや追加のアクセス許可は必要ありません。

DynamoDB は、バックグラウンドシステムメンテナンスと継続的なデータ保護タスクを実行するときに、許可を付与します。また、テーブルキーの生成にグラントを使用します。

各グラントは、テーブルに固有です。アカウントに同じ KMS キーで暗号化された複数のテーブルがある場合、テーブルごとに各タイプのグラントがあります。グラントは、テーブル名と AWS アカウント ID を含む DynamoDB 暗号化コンテキストによって制約を受けます。また、グラントには、それが不要になった場合にグラントを使用停止にする許可が含まれます。

グラントを作成するには、DynamoDB に、暗号化されたテーブルを作成したユーザーに代わって CreateGrant を呼び出すための許可が必要です。AWS マネージドキー では、DynamoDB が認可されたユーザーの代わりにリクエストを行う場合にのみ、アカウントのユーザーに KMS キーで CreateGrant の呼び出しを許可するキーポリシーからの kms:CreateGrant 許可を DynamoDB が取得します。

キーポリシーは、アカウントが KMS キーのグラントを取り消すことも許可できます。ただし、アクティブな暗号化されたテーブルのグラントを取り消すと、DynamoDB はテーブルを保護および維持できなくなります。

DynamoDB 暗号化コンテキスト

暗号化コンテキスト は、一連のキー値のペアおよび任意非シークレットデータを含みます。データを暗号化するリクエストに暗号化コンテキストを組み込むと、AWS KMS は暗号化コンテキストを暗号化されたデータに暗号化してバインドします。データを復号するには、同じ暗号化コンテキストに渡す必要があります。

DynamoDB は、すべての AWS KMS 暗号化オペレーションで同じ暗号化コンテキストを使用します。カスタマーマネージドキーまたは AWS マネージドキー を使用して DynamoDB テーブルを保護する場合、暗号化コンテキストを使用して監査レコードやログにおける KMS キーの使用を特定することができます。これは、AWS CloudTrail や Amazon CloudWatch Logs などのログにもプレーンテキストで表示されます。

また、暗号化コンテクストはポリシーとグラントの認証用の条件としても使用できます。DynamoDB は暗号化コンテキストを使用して、アカウントとリージョンのカスタマーマネージドキーまたは AWS マネージドキー へのアクセスを許可するグラントを制限します。

DynamoDB は AWS KMS へのリクエストで、2 つのキーバリューペアを持つ暗号化コンテキストを使用します。

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- テーブル — 最初のキーと値のペアは、DynamoDB が暗号化しているテーブルを識別します。キーは、aws:dynamodb:tableName です。この値は、テーブルの名前です。

  ```
  "aws:dynamodb:tableName": "<table-name>"
  ```

  以下はその例です。

  ```
  "aws:dynamodb:tableName": "Books"
  ```

- アカウント - 2 番目のキーバリューペアは、AWS アカウント を識別します。キーは、aws:dynamodb:subscriberId です。値は、アカウント ID です。

```
"aws:dynamodb:subscriberId": "<account-id>"
```

以下はその例です。

```
"aws:dynamodb:subscriberId": "111122223333"
```

DynamoDB の AWS KMS との対話をモニタリングする

カスタマーマネージドキーまたは AWS マネージドキー を使用して DynamoDB テーブルを保護する場合は、AWS CloudTrail ログを使用して、DynamoDB がユーザーに代わって AWS KMS に送信するリクエストを追跡することができます。

このセクションでは、GenerateDataKey、Decrypt、および CreateGrant リクエストを説明します。さらに、DynamoDB は DescribeKey オペレーションを使用して、選択した KMS キーがアカウントとリージョンに存在するかどうかを判断します。また、 RetireGrant 操作を使用して、テーブルを削除するときにグラントを削除します。

GenerateDataKey

テーブルで保存時の暗号化を有効にすると、DynamoDB は一意のテーブルキーを作成します。また、GenerateDataKey リクエストを、テーブルの KMS キーを指定する AWS KMS に送信します。

GenerateDataKey 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは DynamoDB サービスアカウントです。パラメータには、KMS キーの Amazon リソースネーム (ARN)、256 ビットキーを必要とするキー指定子、テーブルと AWS アカウント を識別する暗号化コンテキストが含まれます。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dynamodb:tableName": "Services",
            "aws:dynamodb:subscriberId": "111122223333"
        },
        "keySpec": "AES_256",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "229386c1-111c-11e8-9e21-c11ed5a52190",
    "eventID": "e3c436e9-ebca-494e-9457-8123a1f5e979",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "bf915fa6-6ceb-4659-8912-e36b69846aad"
}
```

Decrypt

暗号化された DynamoDB テーブルにアクセスする場合、DynamoDB はテーブルキーを復号化して、階層内でその下にあるキーを復号化できるようにする必要があります。次に、テーブル内のデータを復号化します。テーブルキーを復号化するには、次が実行されます。DynamoDB は、テーブルの KMS キーを指定する復号リクエストを AWS KMS に送信します。

`Decrypt` 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは、テーブルにアクセスしている AWS アカウント のプリンシパルです。パラメータには、暗号化されたテーブルキー (暗号化テキストの blob として)、およびテーブルと AWS アカウント を識別する暗号化コンテキストが含まれます。AWS KMS は、暗号化テキストから KMS キーの ID を取得します。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T16:42:15Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDT3HGFQZX4RY6RU",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T16:42:39Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters":
    {
        "encryptionContext":
        {
            "aws:dynamodb:tableName": "Books",
            "aws:dynamodb:subscriberId": "111122223333"
        }
    },
    "responseElements": null,
    "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
    "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

CreateGrant

カスタマーマネージドキーまたは AWS マネージドキー を使用して DynamoDB テーブルを保護する場合、DynamoDB はグラントを使用して、サービスが継続的なデータ保護、メンテナンス、耐久性のタスクを実行できるようにします。これらのグラントは、AWS 所有のキー では不要です。

DynamoDB が作成するグラントは、テーブルに固有です。CreateGrant リクエストのプリンシパルは、テーブルを作成したユーザーです。

`CreateGrant` 演算を記録するイベントは、次のようなサンプルイベントになります。パラメータには、テーブルの KMS キーの Amazon リソースネーム (ARN)、被付与者プリンシパルと使用停止プリンシパル (DynamoDB サービス)、グラントの対象となるオペレーションが含まれます。また、指定された暗号化コンテキストを使用するすべての暗号化オペレーションを必要とする制約も含まれています。

```
{
    "eventVersion": "1.05",
    "userIdentity":
    {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T00:12:02Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:dynamodb:tableName": "Books",
                "aws:dynamodb:subscriberId": "111122223333"
            }
        },
        "granteePrincipal": "dynamodb.us-west-2.amazonaws.com",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
        ]
    },
    "responseElements": {
        "grantId":
 "5c5cd4a3d68e65e77795f5ccc2516dff057308172b0cd107c85b5215c6e48bde"
    },
    "requestID": "2192b82a-111c-11e8-a528-f398979205d8",
    "eventID": "a03d65c3-9fee-4111-9816-8bf96b73df01",
    "readOnly": false,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

DynamoDB での暗号化テーブルの管理

AWS Management Console または AWS Command Line Interface (AWS CLI) を使用すると、新しいテーブルで暗号化キーを指定し、Amazon DynamoDB の既存のテーブルで暗号化キーを更新できます。

トピック

- 新しいテーブルの暗号化キーを指定する
- 暗号化キーの更新

新しいテーブルの暗号化キーを指定する

Amazon DynamoDB コンソールまたは AWS CLI を使用して新しいテーブルで暗号化キーを指定するには、以下の手順に従います。

暗号化されたテーブルの作成 (コンソール)

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
3. [Create Table] (テーブルの作成) を選択します。[Table name] (テーブル名) に「**Music**」と入力します。プライマリキーには **Artist**、ソートキーには **SongTitle** を、それぞれ文字列で入力します。
4. [Settings] (設定) では、[Customize settings] (設定のカスタマイズ) が選択されていることを確認してください。

> Note
>
> [Use default settings] (デフォルト設定の使用) を選択した場合、テーブルは AWS 所有のキー を使用して保管時に暗号化され、追加コストはかかりません。

5. [Encryption at rest] (保管時の暗号化) で、暗号化タイプ – AWS 所有のキー、AWS マネージドキー、またはカスタマーマネージドキーを選択します。

   - [Owned by Amazon DynamoDB] (Amazon DynamoDB が所有)。AWS が所有するキーで、特に DynamoDB が所有および管理します。このキーは追加料金なしで使用されます。
   - [AWS managed key] (AWS マネージドキー) キーエイリアス: `aws/dynamodb`。キーはアカウントに保存され、AWS Key Management Service (AWS KMS) によって管理されます (AWS KMS の料金が適用されます)。
   - [Stored in your account, and owned and managed by you.] (アカウントに保存され、お客様が所有および管理。) カスタマーマネージドキー。キーはアカウントに保存され、AWS Key Management Service (AWS KMS) によって管理されます (AWS KMS の料金が適用されます)。

   > Note
   >
   > 独自のキーを所有および管理することを選択した場合は、KMS キーポリシーが適切に設定されていることを確認してください。例を含む詳細については、「カスタマーマネージドキーのキーポリシー」を参照してください。

6. [Create table] (テーブルの作成) を選択して暗号化テーブルを作成します。暗号化タイプを確認するには、[Overview] (概要) タブのテーブルの詳細を選択し、[Additional details] (その他の詳細) セクションを表示します。

暗号化されたテーブルの作成 (AWS CLI)

AWS CLI により、Amazon DynamoDB のデフォルトの AWS 所有のキー、AWS マネージドキー、またはカスタマーマネージドキーを使用してテーブルを作成します。

デフォルトの AWS 所有のキー で暗号化テーブルを作成するには

- 暗号化された `Music` テーブルを次のとおりに作成します。

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5
```

> **Note**
>
> このテーブルは、DynamoDB サービスアカウントでデフォルトの AWS 所有のキー を使用して暗号化されます。

AWS マネージドキー で DynamoDB 用の暗号化テーブルを作成するには

- 暗号化された Music テーブルを次のとおりに作成します。

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS
```

テーブルの説明の SSEDescription ステータスは、ENABLED に設定され、SSEType は KMS になります。

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

カスタマーマネージドキーで DynamoDB 用の暗号化テーブルを作成するには

- 暗号化された Music テーブルを次のとおりに作成します。

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

テーブルの説明の `SSEDescription` ステータスは、`ENABLED` に設定され、`SSEType` は `KMS` になります。

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

暗号化キーの更新

DynamoDB コンソールまたは AWS CLI を使用して、AWS 所有のキー、AWS マネージドキー、カスタマーマネージドキーの間でいつでも既存のテーブルの暗号化キーを更新することができます。

暗号化キーの更新 (コンソール)

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
3. 更新するテーブルを選択します。
4. [Actions] (アクション) ドロップダウンを選択し、[Update settings] (設定の更新) オプションを選択します。
5. [Additional settings] (追加設定) タブに移動します。
6. [Encryption] (暗号化) の下の、[Manage encryption] (暗号化の管理) を選択します。

7. 暗号化タイプを選択します。

   - [Owned by Amazon DynamoDB.] (Amazon DynamoDB によって所有される。) AWS KMS キーは DynamoDB によって所有、管理されます。このキーは追加料金なしで使用されます。
   - [AWS managed key] (AWS マネージドキー) のキーエイリアス: `aws/dynamodb`。キーはアカウントに保存され、AWS Key Management Service (AWS KMS) によって管理されます (AWS KMS の料金が適用されます)。
   - [Stored in your account, and owned and managed by you.] (アカウントに保存され、お客様が所有および管理。) キーはアカウントに保存され、AWS Key Management Service (AWS KMS) によって管理されます (AWS KMS の料金が適用されます)。

   > Note
   >
   > 独自のキーを所有および管理することを選択した場合は、KMS キーポリシーが適切に設定されていることを確認してください。詳細については、「カスタマーマネージドキーのキーポリシー」を参照してください。

   続いて、[Save] (保存) を選択して暗号化されたテーブルを更新します。暗号化タイプを確認するには、[概要] タブでテーブルの詳細を参照します。

暗号化キーの更新 (AWS CLI)

次の例は、AWS CLI を使用して暗号化テーブルを更新する方法を示しています。

デフォルトの AWS 所有のキー で暗号化テーブルを更新するには

- 暗号化された `Music` テーブルを次の例のとおりに更新します。

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> Note
>
> このテーブルは、DynamoDB サービスアカウントでデフォルトの AWS 所有のキー を使用して暗号化されます。

DynamoDB 用の AWS マネージドキー で暗号化テーブルを更新するには

- 暗号化された Music テーブルを次の例のとおりに更新します。

  ```
  aws dynamodb update-table \
    --table-name Music \
    --sse-specification Enabled=true
  ```

  テーブルの説明の SSEDescription ステータスは、ENABLED に設定され、SSEType は KMS になります。

  ```
  "SSEDescription": {
    "SSEType": "KMS",
    "Status": "ENABLED",
    "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
  a123-ab1234a1b234",
  }
  ```

DynamoDB のカスタマーマネージドキーで暗号化テーブルを更新するには

- 暗号化された Music テーブルを次の例のとおりに更新します。

  ```
  aws dynamodb update-table \
    --table-name Music \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
  a123-ab1234a1b234
  ```

  テーブルの説明の SSEDescription ステータスは、ENABLED に設定され、SSEType は KMS になります。

  ```
  "SSEDescription": {
    "SSEType": "KMS",
    "Status": "ENABLED",
    "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
  a123-ab1234a1b234",
  }
  ```

DynamoDB Accelerator でのデータ保護

Amazon DynamoDB Accelerator (DAX) 保管時の暗号化は、基になるストレージへの不正アクセスからデータを保護することで、データ保護のレイヤーを追加します。組織のポリシー、業界や政府の規制、またはコンプライアンス要件によって、データを保護するために保管時の暗号化の使用が求められる場合があります。暗号化を使用すると、クラウドにデプロイされたアプリケーションデータの安全性を向上できます。

DAX のデータ保護の詳細については、「保管時の DAX 暗号化」を参照してください。

インターネットトラフィックのプライバシー

接続は、Amazon DynamoDB とオンプレミスのアプリケーション間、および同じの AWS リージョン内の DynamoDB と他の AWS リソース間で保護されます。

エンドポイントに必要なポリシー

Amazon DynamoDB には、リージョンのエンドポイント情報を列挙できる DescribeEndpoints API が用意されています。VPC エンドポイントからのリクエストの場合、IAM エンドポイントポリシーと仮想プライベートクラウド (VPC) エンドポイントポリシーの両方が、IAM の `dynamodb:DescribeEndpoints` アクションを使用して、リクエスト元の ID およびアクセス管理 (IAM) プリンシパルの `DescribeEndpoints` API コールを承認する必要があります。それ以外の場合、`DescribeEndpoints` API へのアクセスは拒否されます。`DescribeEndpoints` API 呼び出しの IAM および VPC エンドポイントポリシーの認証手順は、DynamoDB のパブリックエンドポイントにアクセスする場合には適用されません。

以下は、エンドポイントポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "(Include IAM Principals)",
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

サービスとオンプレミスのクライアントおよびアプリケーションとの間のトラフィック

プライベートネットワークと AWS との間には 2 つの接続オプションがあります:

- AWS Site-to-Site VPN 接続。詳細については、AWS Site-to-Site VPN ユーザーガイド の「AWS Site-to-Site VPN とは」を参照してください。
- AWS Direct Connect 接続。詳細については、AWS Direct Connect ユーザーガイド の「AWS Direct Connect とは」を参照してください。

ネットワークを介した DynamoDB へのアクセスは、AWS が発行する API を利用して行われます。クライアントは Transport Layer Security (TLS) 1.2 をサポートしている必要があります。TLS 1.3 をお勧めします。クライアントは、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) などの Perfect Forward Secrecy (PFS) を備えた暗号スイートもサポートする必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。また、リクエストには、IAM プリンシパルに関連付けられたアクセスキー ID およびシークレットアクセスキーによる署名が必要です。または、リクエストへの署名のために一時的にセキュリティ認証情報を生成する AWS Security Token Service (STS) を使用することもできます。

同じリージョン内の AWS リソース間のトラフィック

Amazon Virtual Private Cloud (Amazon VPC) endpoint for DynamoDB は、DynamoDB への接続のみを許可する VPC 内の論理エンティティです。Amazon VPC はリクエストを DynamoDB にルーティングし、レスポンスを VPC にルーティングします。詳細については、Amazon VPC ユーザーガイドの「VPC エンドポイント」を参照してください。VPC エンドポイントからのアクセスのコントロールに使用できるポリシーの例については、「IAM ポリシーを使用して DynamoDB へのアクセスをコントロールします」を参照してください。

> Note
>
> Amazon VPC エンドポイントには、AWS Site-to-Site VPN または AWS Direct Connect を使用してアクセスすることはできません。

AWS Identity and Access Management (IAM)

AWS Identity and Access Management は、管理者が AWS リソースへのアクセスを安全にコントロールするために役立つ AWS のサービスです。管理者は、誰を認証 (サインイン) し、誰に Amazon

DynamoDB および DynamoDB Accelerator リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM を使用して、Amazon DynamoDB と DynamoDB Accelerator の両方のアクセス許可を管理し、セキュリティポリシーを実装できます。IAM は、AWS のサービスで追加料金は発生しません。

トピック

- Amazon DynamoDB の Identity and Access Management
- 詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用
- DynamoDB アクセラレーターでの Identity and Access Management

Amazon DynamoDB の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービスです。IAM 管理者は、誰が認証 (サインイン) され、DynamoDB リソースを使用する認可 を受ける (アクセス許可がある) ことができるかを管理します。IAM は、追加費用なしで使用できる AWS のサービスです。

トピック

- 対象者
- アイデンティティによる認証
- ポリシーを使用したアクセス権の管理
- Amazon DynamoDB で IAM が機能する仕組み
- Amazon DynamoDB のアイデンティティベースのポリシーの例
- Amazon DynamoDB アイデンティティとアクセスのトラブルシューティング
- DynamoDB リザーブドキャパシティの購入を防止する IAM ポリシー

対象者

AWS Identity and Access Management (IAM) の用途は、DynamoDB で行う作業によって異なります。

サービスユーザー – ジョブを実行するために DynamoDB サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。作業を実行するためにさらに多くの DynamoDB の機

能を使用する際は、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解すると、管理者から適切な権限をリクエストするのに役に立ちます。DynamoDB の機能にアクセスできない場合は、「Amazon DynamoDB アイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 - 社内の DynamoDB リソースを担当している場合は、通常、DynamoDB へのフルアクセスがあります。サービスのユーザーがどの DynamoDB 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。お客様の会社において DynamoDB で IAM を利用する方法の詳細については、「Amazon DynamoDB で IAM が機能する仕組み」を参照してください。

IAM 管理者 – IAM 管理者には、DynamoDB へのアクセスを管理するポリシーの作成方法の詳細を理解することが推奨されます。IAM で使用できる DynamoDB アイデンティティベースのポリシーの例を表示する方法については、「Amazon DynamoDB のアイデンティティベースのポリシーの例」を参照してください。

アイデンティティによる認証

認証とは、アイデンティティ認証情報を使用して AWSにサインインする方法です。ユーザーは、AWS アカウントのルートユーザー として、IAM ユーザーとして、または IAM ロールを引き受けることによって、認証済み (AWS にサインイン済み) である必要があります。

ID ソースから提供された認証情報を使用すると、フェデレーティッドアイデンティティとして AWS にサインインできます。AWS IAM Identity Centerフェデレーティッドアイデンティティの例としては、(IAM アイデンティティセンター) ユーザー、貴社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Console または AWSアクセスポータルにサインインできます。AWS へのサインインの詳細については、『AWS サインイン ユーザーガイド』の「AWS アカウント にサインインする方法」を参照してください。

プログラムで AWS にアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報でリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、『IAM ユーザーガイド』の「AWS API リクエストの署名」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、AWS では、アカウントのセキュリティ強化のために多要素認証 (MFA) の使用をお勧めしています。詳細については、『AWS IAM Identity Center ユーザーガイド』の「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWS における多要素認証 (MFA) の使用」を参照してください。

AWS アカウントのルートユーザー

AWS アカウントを作成する場合は、このアカウントのすべての AWS のサービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用したメールアドレスとパスワードでサインインすることによってアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、『IAM ユーザーガイド』の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに対し、ID プロバイダーとのフェデレーションを使用して、一時認証情報の使用により、AWS のサービス にアクセスすることを要求します。

フェデレーテッドアイデンティティ は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザーか、または ID ソースから提供された認証情報を使用して AWS のサービス にアクセスするユーザーです。フェデレーテッドアイデンティティ が AWS アカウント にアクセスすると、ロールが継承され、ロールは一時認証情報を提供します。

アクセスを一元管理する場合は、AWS IAM Identity Center を使用することをお勧めします。IAM アイデンティティセンター でユーザーとグループを作成するか、すべての AWS アカウント とアプリケーションで使用するために、独自の ID ソースで一連のユーザーとグループに接続して同期することもできます。IAM Identity Center の詳細については、『AWS IAM Identity Center ユーザーガイド』の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたは 1 つのアプリケーションに対して特定の権限を持つ AWS アカウント 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めしま

す。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、『IAM ユーザーガイド』の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、『IAM ユーザーガイド』の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定の権限を持つ、AWS アカウント 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールの切り替えによって、AWS Management Console で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、AWS CLI または AWSAPI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、『IAM ユーザーガイド』の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッドアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」 を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。

- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービスでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス権 - 一部の AWS のサービスでは、他の AWS のサービスの機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスにリンクされたロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、『IAM ユーザーガイド』の「AWS のサービス に権限を委任するロールの作成」を参照してください。
  - サービスリンクロール - サービスリンクロールは、AWS のサービス にリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。
- Amazon EC2 で実行されるアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API 要求を行っているアプリケーションの一時的な認証情報を管理するために、IAM ロールを使用できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファ

イルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、『IAM ユーザーガイド』の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセス権を管理するには、ポリシーを作成して AWS アイデンティティまたはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらの権限を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWSに保存されます。JSON ポリシードキュメントの構造と内容の詳細については、『IAM ユーザーガイド』の「JSON ポリシー概要」を参照してください。

管理者は AWSJSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれます。管理ポリシーは、AWS アカウント 内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーおよびカス

タマーマネージドポリシーがあります。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、『IAM ユーザーガイド』の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは IAM の AWSマネージドポリシーは使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、『Amazon Simple Storage Service デベロッパーガイド』の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界界の詳細については、『IAM ユーザーガイド』の「IAM エンティティのアクセス許可の境界」を参照してください。

- サービスコントロールポリシー (SCP) - SCP は、AWS Organizations で組織や組織単位 (OU) の最大権限を指定する JSON ポリシーです。AWS Organizations は、顧客のビジネスが所有する複数の AWS アカウント をグループ化し、一元的に管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対する権限を制限します (各 AWS アカウントのルートユーザー など)。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、『IAM ユーザーガイド』の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、「IAM ユーザーガイド」の「ポリシーの評価論理」を参照してください。

Amazon DynamoDB で IAM が機能する仕組み

IAM を使用して DynamoDB へのアクセスを管理する前に、DynamoDB で利用できる IAM の機能について学びます。

Amazon DynamoDB で使用できる IAM の機能

IAM の機能	DynamoDB のサポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	はい
ポリシーアクション	Yes
ポリシーリソース	Yes

IAM の機能	DynamoDB のサポート
ポリシー条件キー	Yes
ACL	No
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	Yes
プリンシパル権限	Yes
サービスロール	あり
サービスリンクロール	[いいえ]

大部分の IAM 機能が DynamoDB および AWS のその他のサービスでどのように連携するかに関するおおまかな説明については、「IAM ユーザーガイド」の「IAM と連携する AWS のサービス」を参照してください。

DynamoDB のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

DynamoDB のアイデンティティベースのポリシーの例

DynamoDB アイデンティティベースのポリシーの例を確認するには、「Amazon DynamoDB のアイデンティティベースのポリシーの例」を参照してください。

DynamoDB 内のリソースベースのポリシー

リソースベースのポリシーのサポート	はい

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる AWS アカウント にある場合、信頼できるアカウントの IAM 管理者は、リソースにアクセスするための権限をプリンシパルエンティティ (ユーザーまたはロール) に付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

DynamoDB のポリシーアクション

ポリシーアクションに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWSAPI オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があ

ります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

DynamoDB アクションのリストを確認するには、「サービス認可リファレンス」の「Amazon DynamoDB で定義されるアクション」を参照してください。

DynamoDB のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。

```
aws
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

DynamoDB アイデンティティベースのポリシーの例を確認するには、「Amazon DynamoDB のアイデンティティベースのポリシーの例」を参照してください。

DynamoDB のポリシーリソース

ポリシーリソースに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

DynamoDB リソースのタイプとその ARN のリストを確認するには、「サービス認証リファレンス」の「Amazon DynamoDB で定義されるリソース」を参照してください。各リソースの ARN を指定できるアクションについては、「Amazon DynamoDB で定義されているアクション」を参照してください。

DynamoDB アイデンティティベースのポリシーの例を確認するには、「Amazon DynamoDB のアイデンティティベースのポリシーの例」を参照してください。

DynamoDB のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、AWS は AND 論理演算子を使用してそれらを評価します。単一の条件キーに複数の値を指定すると、AWS は OR 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS はグローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイド の「AWS グローバル条件コンテキストキー」を参照してください。

DynamoDB の条件キーの一覧については、「サービス認可リファレンス」の「Amazon DynamoDB の条件キー」を参照してください。どのアクションおよびリソースで条件キーを使用できるかについては、「Amazon DynamoDB で定義されるアクション」を参照してください。

DynamoDB アイデンティティベースのポリシーの例を確認するには、「Amazon DynamoDB のアイデンティティベースのポリシーの例」を参照してください。

DynamoDB のアクセスコントロールリスト (ACL)

ACL のサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

DynamoDB での属性ベースのアクセス制御 (ABAC)

ABAC (ポリシー内のタグ) のサポート	部分的

属性ベースのアクセスコントロール (ABAC) は、属性に基づいて権限を定義する認可戦略です。AWS では、属性は タグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール)、および多数の AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合に操作を許可するように ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値ははいです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、『IAM ユーザーガイド』の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

DynamoDB での一時的な認証情報の使用

一時的な認証情報のサポート	Yes

AWS のサービスには、一時認証情報を使用してサインインしても機能しないものがあります。一時的な認証情報を利用できる AWS のサービス を含めた詳細情報については、『IAM ユーザーガイド』の「IAM と連携する AWS のサービス」を参照してください。

ユーザー名とパスワード以外の方法で AWS Management Consoleにサインインする場合は、一時認証情報を使用していることになります。例えば、会社のシングルサインオン (SSO) リンクを使用して AWS にアクセスすると、そのプロセスは自動的に一時認証情報を作成します。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、『IAM ユーザーガイド』の「ロールへの切り替え (コンソール)」を参照してください。

一時認証情報は、AWS CLI または AWSAPI を使用して手動で作成できます。作成後、一時認証情報を使用して AWSにアクセスできるようになります。AWS は、長期的なアクセスキーを使用する代わりに、一時認証情報を動的に生成することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

DynamoDB のクロスサービスプリンシパル許可

フォワードアクセスセッション (FAS) をサポート	Yes

IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルとみなされます。一部のサービスを使用する際に、アクションを実行してから、別のサービスの別のアクションを開始することがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

DynamoDB のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細につい

ては、『IAM ユーザーガイド』の「AWS のサービス に権限を委任するロールの作成」を参照してください。

> ⚠ Warning
>
> サービスロールの許可を変更すると、DynamoDB の機能が破損する可能性があります。DynamoDB が指示する場合以外は、サービスロールを編集しないでください。

DynamoDB 用のサービスリンクロール

サービスにリンクされたロールのサポート	いいえ

サービスリンクロールとは、AWS のサービスにリンクされるサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携する AWS のサービス」を参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] リンクを選択します。

Amazon DynamoDB のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールには、DynamoDB リソースを作成または変更するアクセス許可はありません。また、AWS Management Console、AWS Command Line Interface (AWS CLI)、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

DynamoDB が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認証リファレンス」の「Amazon DynamoDB のアクション、リソース、および条件キー」を参照してください。

トピック

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰かが DynamoDB リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマーマネージドポリシーを定義することで、権限をさらに減らすことをお勧めします。詳細については、『IAM ユーザーガイド』の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービスを介して使用する場合、条件を使ってサービスアクションへのアクセス権を付与することもできます。詳細については、『IAM ユーザーガイド』の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。

- 多要素認証 (MFA) を要求する – AWS アカウント で IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、『IAM ユーザーガイド』の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

DynamoDB コンソールの使用

Amazon DynamoDB コンソールにアクセスするには、許可の最小限のセットが必要です。これらのアクセス許可により、AWS アカウント の DynamoDB リソースの詳細をリストおよび表示できます。最小限必要なアクセス許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) ではコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソール権限を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスを許可します。

ユーザーとロールが引き続き DynamoDB コンソールを使用できるようにするには、エンティティに DynamoDB の `ConsoleAccess` または `ReadOnly` AWS マネージドポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了する権限が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon DynamoDB でのアイデンティティベースポリシーの使用

このトピックでは、Amazon DynamoDB でのアイデンティティベースの AWS Identity and Access Management (IAM) ポリシーの使用についての説明と、例を示します。これらの例は、アカウント管理者が IAM アイデンティティ (ユーザー、グループ、およびロール) に許可ポリシーをアタッチし、それによって Amazon DynamoDB リソースに対するオペレーションを実行するための許可を付与する方法を示しています。

このセクションでは、次のトピックを対象としています。

- Amazon DynamoDB コンソールの使用に必要な IAM 許可
- Amazon DynamoDB の AWS マネージド (事前定義) IAM ポリシー
- カスタマーマネージドポリシーの例

以下は、アクセス権限ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:account-id:table/Books"
        }
    ]
}
```

上記のポリシーには、`us-west-2` AWS リージョンのテーブルで 3 つの DynamoDB アクション (`dynamodb:DescribeTable`、`dynamodb:Query`、`dynamodb:Scan`) を許可する 1 つのステートメントがあります。これは、*`account-id`* で指定される AWS アカウントで所有されています。`Resource` 値の Amazon リソースネーム (ARN) では、許可が適用されるテーブルを指定します。

Amazon DynamoDB コンソールの使用に必要な IAM 許可

ユーザーが DynamoDB コンソールを使用するには、AWS アカウントで DynamoDB リソースの使用を許可する最小限の許可セットが必要です。これらの DynamoDB 許可に加えて、コンソールでは次の許可が必要になります。

- メトリクスとグラフを表示する Amazon CloudWatch 許可。
- DynamoDB データをエクスポートおよびインポートする AWS Data Pipeline アクセス許可。
- AWS Identity and Access Managementエクスポートおよびインポートに必要なロールにアクセスする アクセス許可。
- CloudWatch アラームがトリガーされるたびに通知する Amazon Simple Notification Service 許可。
- DynamoDB Streams レコードを処理するための AWS Lambda 許可。

これらの最小限必要なアクセス権限よりも制限された IAM ポリシーを作成している場合、その IAM ポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しません。Amazon

DynamoDB の AWS マネージド (事前定義) IAM ポリシー で説明されている通り、ユーザーが引き続き DynamoDB コンソールを使用できること、および `AmazonDynamoDBReadOnlyAccessAWS` マネージドポリシーがユーザーにアタッチされていることを確認してください。

AWS CLI または Amazon DynamoDB API のみを呼び出すユーザーには、最小限のコンソール許可を付与する必要はありません。

> Note
>
> VPC エンドポイントを参照する場合、IAM アクション (dynamodb:DescribeEndpoints) を使用して、リクエストしている IAM プリンシパルの DescribeEndpoints API コールを認証する必要もあります。詳細については、「エンドポイントに必要なポリシー」を参照してください。

Amazon DynamoDB の AWS マネージド (事前定義) IAM ポリシー

AWS は、AWS によって作成され管理されるスタンドアロンの IAM ポリシーが提供する多くの一般的ユースケースに対応します。これらの AWS 管理ポリシーは、一般的ユースケースに必要なアクセス権限を付与することで、どの権限が必要なのかをユーザーが調査する必要をなくすことができます。詳細については、IAM ユーザーガイドの「AWS 管理ポリシー」を参照してください。

アカウントのユーザーにアタッチ可能な以下の AWS マネージドポリシーは、DynamoDB 固有のもので、ユースケースシナリオ別にグループ化されます。

- AmazonDynamoDBReadOnlyAccess – AWS Management Console を介して DynamoDB リソースへの読み込み専用アクセスを許可します。
- AmazonDynamoDBFullAccess – AWS Management Console を介して DynamoDB リソースへのフルアクセスを許可します。

IAM コンソールにサインインし、特定のポリシーを検索することで、これらの AWS マネージド許可ポリシーを確認できます。

> Important
>
> ベストプラクティスは、ユーザー、ロール、またはそれらを必要とするグループに最小特権を付与するカスタム IAM ポリシーを作成することです。

カスタマーマネージドポリシーの例

このセクションでは、さまざまな DynamoDB アクションの許可を付与するポリシー例を示しています。これらのポリシーは、AWS SDK または AWS CLI を使用しているときに機能します。コンソールを使用している場合は、コンソールに固有の追加許可を付与する必要があります。詳細については、「Amazon DynamoDB コンソールの使用に必要な IAM 許可」を参照してください。

> **Note**
>
> 以下のすべてのポリシー例では、AWS リージョンの 1 つを使用しており、架空のアカウント ID とテーブル名が含まれています。

例:

- テーブル上のすべての DynamoDB アクションに許可を付与する IAM ポリシー
- DynamoDB テーブルの項目に読み込み専用許可を付与する IAM ポリシー
- 特定の DynamoDB テーブルとそのインデックスへのアクセスを付与する IAM ポリシー
- DynamoDB テーブルに対するアクセスの読み込み、書き込み、更新、削除を行う IAM ポリシー
- 同じ AWS アカウントで DynamoDB 環境を分離する IAM ポリシー
- DynamoDB リザーブドキャパシティの購入を防止する IAM ポリシー
- DynamoDB ストリームのみの読み込みアクセスを付与する IAM ポリシー (テーブルを除く)
- AWS Lambda 関数が DynamoDB ストリームレコードへのアクセスを許可する IAM ポリシー
- DynamoDB アクセラレーター (DAX) クラスターへの読み込みおよび書き込みアクセスに関する IAM ポリシー

「IAM ユーザーガイド」には、追加の DynamoDB 例が 3 つ含まれています。

- Amazon DynamoDB: 特定のテーブルへのアクセスの許可
- Amazon DynamoDB: 特定の列へのアクセスの許可
- Amazon DynamoDB: Amazon Cognito ID に基づいて DynamoDB への行レベルのアクセスを許可する

テーブル上のすべての DynamoDB アクションに許可を付与する IAM ポリシー

以下のポリシーでは、Books というテーブルでのすべての DynamoDB アクションの許可を付与します。Resource で指定されたリソース ARN は、特定の AWS リージョンのテーブルを識別します。Resource ARN のテーブル名 Books をワイルドカード文字 (*) で置き換えると、アカウント内のすべてのテーブルですべての DynamoDB アクションが許可されます。このポリシーまたは IAM ポリシーでワイルドカード文字を使用する前に、考えられるセキュリティへの影響を慎重に検討してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> Note
>
> これは、ワイルドカード文字 (*) を使用して、すべてのアクション (管理、データオペレーション、モニタリング、DynamoDB リザーブドキャパシティーの購入など) を許可する例です。代わりに、付与する各アクションと、そのユーザー、ロール、またはグループが必要とするアクションのみを明示的に指定することをお勧めします。

DynamoDB テーブルの項目に読み込み専用許可を付与する IAM ポリシー

次の許可ポリシーは、GetItem、BatchGetItem、Scan、Query、および ConditionCheckItem DynamoDB アクションのみに許可を付与し、その結果、Books テーブルへの読み込み専用アクセスを設定します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

特定の DynamoDB テーブルとそのインデックスへのアクセスを付与する IAM ポリシー

以下のポリシーでは、Books という DynamoDB テーブルと、そのテーブルのすべてのインデックスに対するデータ変更アクションの許可を付与します。インデックスの動作の詳細については、「セカンダリインデックスを使用したデータアクセス性の向上」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessTableAllIndexesOnBooks",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

DynamoDB テーブルに対するアクセスの読み込み、書き込み、更新、削除を行う IAM ポリシー

アプリケーションが Amazon DynamoDB テーブル、インデックス、およびストリーミングのデータを作成、読み込み、更新、および削除できるようにする必要がある場合は、このポリシーを使用します。必要に応じて、AWS リージョン名、アカウント ID、 テーブル名またはワイルドカード文字 (*) に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBIndexAndStreamAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/stream/*"
            ]
        },
        {
            "Sid": "DynamoDBTableAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:DescribeTable",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        },
        {
            "Sid": "DynamoDBDescribeLimitsAccess",
            "Effect": "Allow",
            "Action": "dynamodb:DescribeLimits",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

このポリシーを拡張して、このアカウントのすべての AWS リージョンのすべての DynamoDB テーブルをカバーするには、リージョンとテーブル名にワイルドカード (*) を使用します。以下に例を示します。

```
"Resource":[
                "arn:aws:dynamodb:*:123456789012:table/*",
                "arn:aws:dynamodb:*:123456789012:table/*/index/*"
                ]
```

同じ AWS アカウントで DynamoDB 環境を分離する IAM ポリシー

たとえば、各環境で、`ProductCatalog` という名前のテーブルを独自のバージョンで維持するとします。2 つの `ProductCatalog` テーブルを同じ AWS アカウントから作成する場合、許可の設定方法により、片方の環境での動作が他の環境に影響を与える可能性があります。例えば、同時制御プレーンオペレーション (`CreateTable` など) の数に対するクォータは AWS アカウントレベルで設定されれます。

その結果、1 つの環境内の各アクションによって、もう一方の環境で利用可能なオペレーションの数が減少します。また、片方の環境内のコードが他の環境内のテーブルに偶然にアクセスしてしまう危険性もあります。

> **Note**
>
> 本番ワークロードとテストワークロードを分離して、イベントの潜在的な「爆発半径」を制御する場合、ベストプラクティスは、テストワークロードと本番ワークロード用に別々の AWS アカウントを作成することです。詳細については、AWS アカウントの管理および分離を参照してください。

さらに、Amit と Alice という 2 人のデベロッパーが ProductCatalog テーブルをテストしているとします。デベロッパーごとに別々の AWS アカウントを作成する代わりに、デベロッパー間で同じテスト AWS アカウントを共有することができます。このテスト用アカウントでは、Alice_ProductCatalog や Amit_ProductCatalog など、対象のデベロッパーごとに、同じテーブルのコピーを作成できます。この場合、テスト環境用に作成した AWS アカウント内で、ユーザーの Alice と Amit を作成できます。次に、所有するテーブルに対して DynamoDB アクションを実行できるように、これらのユーザーに許可を付与することができます。

これらの IAM ユーザーの許可を付与するには、次のいずれかを実行します。

- 各ユーザーに別々のポリシーを作成し、各ポリシーをユーザーごとに個別にアタッチします。たとえば、以下のポリシーをユーザーである Alice にアタッチすることで、Alice_ProductCatalog テーブルに対する DynamoDB アクションへのアクセスをすべて許可することができます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnAliceTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:DeleteItem",
              "dynamodb:DescribeContributorInsights",
              "dynamodb:RestoreTableToPointInTime",
              "dynamodb:ListTagsOfResource",
              "dynamodb:CreateTableReplica",
              "dynamodb:UpdateContributorInsights",
              "dynamodb:CreateBackup",
              "dynamodb:DeleteTable",
              "dynamodb:UpdateTableReplicaAutoScaling",
              "dynamodb:UpdateContinuousBackups",
              "dynamodb:TagResource",
              "dynamodb:DescribeTable",
              "dynamodb:GetItem",
              "dynamodb:DescribeContinuousBackups",
              "dynamodb:BatchGetItem",
              "dynamodb:UpdateTimeToLive",
              "dynamodb:BatchWriteItem",
              "dynamodb:ConditionCheckItem",
              "dynamodb:UntagResource",
              "dynamodb:PutItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteTableReplica",
              "dynamodb:DescribeTimeToLive",
              "dynamodb:RestoreTableFromBackup",
              "dynamodb:UpdateTable",
              "dynamodb:DescribeTableReplicaAutoScaling",
              "dynamodb:GetShardIterator",
              "dynamodb:DescribeStream",
              "dynamodb:GetRecords",
              "dynamodb:DescribeLimits",
              "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
Alice_ProductCatalog/*"
        }
    ]
}
```

次に、ユーザー Amit に対して、同様のポリシーを異なるリソース (Amit_ProductCatalog テーブル) で作成できます。

- 個々のユーザーにポリシーを付与する代わりに、IAM ポリシー変数を使用して単一のポリシーを記述し、グループに付与することができます。この例では、グループを作成し、そのグループにユーザー Alice とユーザー Amit の両者を追加する必要があります。次の例では、${aws:username}_ProductCatalog テーブルのすべての DynamoDB アクションを実行できる許可を付与します。ポリシーが評価されると、ポリシー変数 ${aws:username} はリクエスターのユーザー名に置き換えられます。たとえば、Alice が項目の追加要求を送信する場合、Alice が Alice_ProductCatalog テーブルにアイテムを追加する場合にのみ、そのアクションが許可されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ActionsOnUserSpecificTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
${aws:username}_ProductCatalog"
        },
        {
            "Sid": "AdditionalPrivileges",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
        }
    ]
}
```

Note

IAM ポリシー変数を使用する場合、ポリシーで明示的に IAM ポリシー言語の 2012-10-17 バージョンを指定する必要があります。IAM ポリシー言語のデフォルトバージョン (2008-10-17) では、ポリシー変数をサポートしていません。

通常のように具体的なテーブルをリソースとして特定する代わりに、ワイルドカード文字 (*) を使用して、すべてのテーブルに許可を付与することができます。次の例に示すように、テーブル名の先頭には、リクエストを行っているユーザーの名前が付きます。

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

DynamoDB リザーブドキャパシティの購入を防止する IAM ポリシー

Amazon DynamoDB リザーブドキャパシティーでは、1 回限りの前払い料金を支払い、期間中、大幅な節約ができる最小使用レベルを支払う契約を結びます。AWS Management Console を使用して、リザーブドキャパシティーを表示し、購入できます。ただし、組織のすべてのユーザーが、リ

ザーブドキャパシティーを購入できないようにしたい場合があります。リザーブドキャパシティーの詳細については、Amazon DynamoDB の料金を参照してください。

DynamoDB は、リザーブドキャパシティー管理へのアクセスを制御するために、次の API オペレーションを提供します。

- `dynamodb:DescribeReservedCapacity` – 現在有効なリザーブドキャパシティーの購入を返します。
- `dynamodb:DescribeReservedCapacityOfferings` – AWS で現在提供されているリザーブドキャパシティープランについての詳細を返します。
- `dynamodb:PurchaseReservedCapacityOfferings` – リザーブドキャパシティーの実際の注文を実行します。

AWS Management Console はこれらの API アクションを使用してリザーブドキャパシティーの情報を表示し、購入を行います。アプリケーションプログラムからこれらのオペレーションを呼び出すことはできません。オペレーションにはコンソールからのみアクセスできるためです。ただし、IAM 許可ポリシーでこれらのオペレーションへのアクセスを許可または拒否することができます。

以下のポリシーでは、ユーザーは AWS Management Console を使用して、リザーブドキャパシティの購入およびサービスを表示できますが、新規購入は拒否されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

このポリシーでは、ワイルドカード文字 (*) を使用して、すべてのユーザーに説明許可を付与し、すべての DynamoDB リザーブドキャパシティーの購入を拒否することに注意してください。

DynamoDB ストリームのみの読み込みアクセスを付与する IAM ポリシー (テーブルを除く)

テーブルで DynamoDB Streams を有効にすると、テーブル内の項目に加えられたすべての変更に関する情報をキャプチャします。詳細については、「」を参照してくださいDynamoDB Streams の変更データキャプチャ

場合によっては、アプリケーションが DynamoDB テーブルからデータを読み取らないようにする一方で、そのテーブルのストリーミングへのアクセスは許可したいことがあります。たとえば、項目の更新が検出されたときにストリーミングをポーリングし、Lambda 関数をコールして、追加の処理を実行するよう AWS Lambda を設定できます。

DynamoDB Streams へのアクセスを制御するために、以下のアクションを使用できます。

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

次のポリシー例では、`GameScores` という名前のテーブルでストリーミングにアクセスする許可をユーザーに付与します。ARN のワイルドカード文字 (*) は、そのテーブルに関連付けられた任意のストリーミング ID に一致します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessGameScoresStreamOnly",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/
stream/*"
        }
    ]
}
```

このポリシーは GameScores テーブルのストリーミングへのアクセスを付与しますが、テーブル自体へのアクセスは付与しないことに注意してください。

AWS Lambda 関数が DynamoDB ストリームレコードへのアクセスを許可する IAM ポリシー

DynamoDB Streams のイベントに基づいて特定のアクションを実行する場合、それらのイベントによってトリガーされる AWS Lambda 関数を書き込むことができます。このような Lambda 関数には、DynamoDB Streams からデータを読み込む許可が必要です。DynamoDB Streams で Lambda を使用する詳細方法については、DynamoDB Streams と AWS Lambda のトリガー を参照してください。

Lambda に許可を付与するには、Lambda 関数の IAM ロール (実行ロールともいう) に関連付けられた許可ポリシーを使用します。Lambda 関数の作成時に、このポリシーを指定します。

たとえば、次の許可ポリシーを実行ロールに関連付けて、リストされた DynamoDB Streams アクションを実行する許可を Lambda に付与できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "APIAccessForDynamoDBStreams",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/
stream/*"
        }
    ]
}
```

詳細については、AWS Lambda デベロッパーガイドの AWS Lambda 許可を参照してください。

DynamoDB アクセラレーター (DAX) クラスターへの読み込みおよび書き込みアクセスに関する IAM ポリシー

次のポリシーは、DynamoDB アクセラレーター (DAX) クラスターへの読み込み、書き込み、更新、および削除アクセスを許可しますが、関連する DynamoDB テーブルへのアクセスは許可しません。このポリシーを使用するには、AWS リージョン名、アカウント ID、DAX クラスターの名前に置き換えます。

> **Note**
>
> このポリシーは、DAX クラスターへのアクセスを許可しますが、関連する DynamoDB テーブルへのアクセスは許可しません。DAX クラスターに、ユーザーに代わって DynamoDB テーブルでこれらの同じオペレーションを実行するための正しいポリシーがあることを確認してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonDynamoDBDAXDataOperations",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:PutItem",
                "dax:ConditionCheckItem",
                "dax:BatchGetItem",
                "dax:BatchWriteItem",
                "dax:DeleteItem",
                "dax:Query",
                "dax:UpdateItem",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:eu-west-1:123456789012:cache/MyDAXCluster"
        }
    ]
}
```

このポリシーを拡張して、アカウントのすべての AWS リージョンの DAX アクセスをカバーするには、リージョン名にワイルドカード文字 (*) を使用します。

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Amazon DynamoDB アイデンティティとアクセスのトラブルシューティング

次の情報は、DynamoDB と IAM の使用に伴って発生する可能性がある一般的な問題の診断や解決に役立ちます。

トピック

- DynamoDB でアクションを実行する権限がない
- iam:PassRole を実行する権限がない
- 自分の AWS アカウント 以外のユーザーに DynamoDB リソースへのアクセスを許可したい

DynamoDB でアクションを実行する権限がない

AWS Management Console から、アクションを実行することが認可されていないと通知された場合、管理者に問い合わせ、サポートを依頼する必要があります。担当の管理者はお客様のユーザー名とパスワードを発行した人です。

以下のエラー例は、`mateojackson` ユーザーがコンソールを使用して架空の *`my-example-widget`* リソースに関する詳細情報を表示しようとしているが、架空の `aws:`*`GetWidget`* 許可がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

この場合、Mateo は、`aws:`*`GetWidget`* アクションを使用して *`my-example-widget`* リソースにアクセスできるように、管理者にポリシーの更新を依頼します。

iam:PassRole を実行する権限がない

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して DynamoDB にロールを渡せるようにする必要があります。

一部の AWS のサービスでは、新しいサービスロールまたはサービスリンクロールを作成せずに、既存のロールをサービスに渡すことが許可されています。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して DynamoDB でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。管理者とは、サインイン認証情報を提供した担当者です。

自分の AWS アカウント 以外のユーザーに DynamoDB リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセス制御リスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- DynamoDB でこれらの特徴がサポートされるかどうかを確認するには、「Amazon DynamoDB で IAM が機能する仕組み」を参照してください。
- 所有している AWS アカウント全体のリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「所有している別の AWS アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、『IAM ユーザーガイド』の「第三者が所有する AWS アカウント へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

DynamoDB リザーブドキャパシティの購入を防止する IAM ポリシー

Amazon DynamoDB リザーブドキャパシティーでは、1 回限りの前払い料金を支払い、期間中、大幅な節約ができる最小使用レベルを支払う契約を結びます。AWS Management Console を使用して、リザーブドキャパシティーを表示し、購入できます。ただし、組織のすべてのユーザーが、リザーブドキャパシティーを購入できないようにしたい場合があります。リザーブドキャパシティーの詳細については、Amazon DynamoDB の料金を参照してください。

DynamoDB は、リザーブドキャパシティー管理へのアクセスを制御するために、次の API オペレーションを提供します。

- dynamodb:DescribeReservedCapacity – 現在有効なリザーブドキャパシティーの購入を返します。
- dynamodb:DescribeReservedCapacityOfferings – AWS で現在提供されているリザーブドキャパシティープランについての詳細を返します。
- dynamodb:PurchaseReservedCapacityOfferings – リザーブドキャパシティーの実際の注文を実行します。

AWS Management Console はこれらの API アクションを使用してリザーブドキャパシティーの情報を表示し、購入を行います。アプリケーションプログラムからこれらのオペレーションを呼び出すことはできません。オペレーションにはコンソールからのみアクセスできるためです。ただし、IAM 許可ポリシーでこれらのオペレーションへのアクセスを許可または拒否することができます。

以下のポリシーでは、ユーザーは AWS Management Console を使用して、リザーブドキャパシティの購入およびサービスを表示できますが、新規購入は拒否されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

このポリシーでは、ワイルドカード文字 (*) を使用して、すべてのユーザーに説明許可を付与し、すべての DynamoDB リザーブドキャパシティーの購入を拒否することに注意してください。

詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用

DynamoDB で許可を付与するときは、許可ポリシーを有効にする方法を決める条件を指定できます。

概要

DynamoDB では、IAM ポリシー (Amazon DynamoDB の Identity and Access Management を参照) を使用して許可を付与するとき、条件を指定することもできます。たとえば、以下のことが可能です。

- テーブル内またはセカンダリインデックスの特定の項目と属性に対する読み込み専用アクセスをユーザーに許可する許可を付与します。
- ユーザーのアイデンティティに基づいて、テーブルの特定の属性への書き込み専用アクセスのアクセス権限をユーザーに付与します。

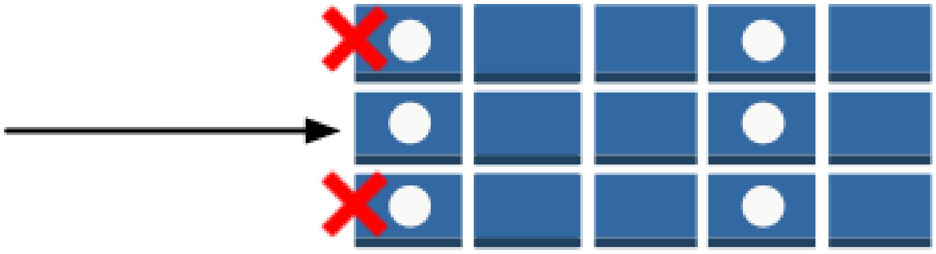
DynamoDB では、次のセクションのユースケースに示すように、条件キーを使用して IAM ポリシーの条件を指定できます。

> Note
>
> 一部の AWS サービスはタグベースの条件もサポートしますが、DynamoDB はサポートしません。

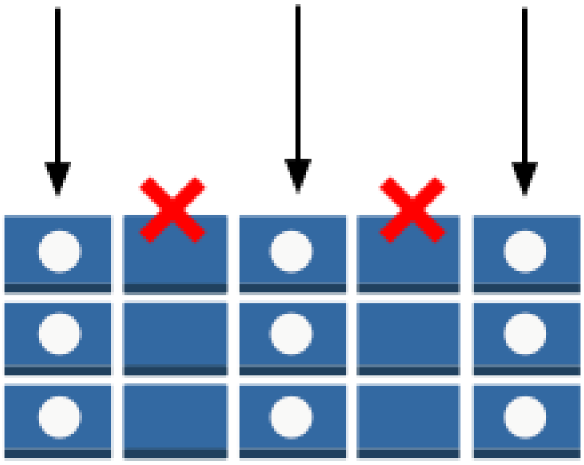
アクセス権限のユースケース

DynamoDB API アクションへのアクセスを制御するだけでなく、個別のデータ項目と属性へのアクセスも制御できます。例えば、次の操作を実行できます。

- テーブルでアクセス権限を付与しますが、特定のプライマリキー値に基づいて、そのテーブル内の特定の項目へのアクセスを制限します。この例として、ゲーム用のソーシャルネットワーキングアプリケーションがあります。次の図に示すように、すべてのユーザーのゲームデータは 1 つのテーブルに保存されますが、いずれのユーザーも、自分が所有していないデータ項目にアクセスすることはできません。



- 属性のサブセットのみがユーザー表示されるように、情報を非表示にします。この例として、ユーザーの位置に基づいて、近くの空港の運航便データを表示するアプリケーションがあります。航空会社名、到着時間、出発時間、および運航便番号がすべて表示されます。ただし、次の図に示すように、パイロット名や乗客数などの属性は非表示になります。



このような種類のきめ細かなアクセス制御を実装するには、セキュリティ認証情報と関連する許可にアクセスするための条件を指定した IAM 許可ポリシーを書き込みます。次に、IAM コンソールを使用して作成する ユーザー、グループ、またはロールにそのポリシーを適用します。IAM ポリシーによって、テーブルの個別項目へのアクセス、その項目の属性へのアクセス、またはその両方へのアクセスを同時に制御できます。

必要に応じて、ウェブアイデンティティフェデレーションを使用して、Login with Amazon、Facebook、または Google によって認証されるユーザーによるアクセスを制御できます。詳細については、「」を参照してくださいウェブ ID フェデレーションの使用

IAM の Condition 要素を使用して、詳細なアクセスコントロールポリシーを実装できます。Condition 要素を許可ポリシーに追加することで、特定のビジネス要件に基づいて、DynamoDB テーブルとインデックスの項目と属性へのアクセスを許可または拒否できます。

例として、プレーヤーがさまざまなゲームを選択してプレイできるモバイルゲームアプリケーションを考えてみます。このアプリケーションでは、GameScores という名前の DynamoDB テーブルを使用して、高スコアなどのユーザーデータを記録します。テーブルの各項目は、ユーザー ID とユーザーがプレイしたゲームの名前で一意に識別されます。GameScores テーブルには、パーティションキー (UserId) およびソートキー (GameTitle) で構成されているプライマリキーがあります。ユーザーは、そのユーザー ID に関連付けられたゲームデータだけにアクセスできます。ゲームをプレイするユーザーは、GameRole という名前の IAM ロールに属する必要があります。このロールには、セキュリティポリシーが添付されています。

このアプリケーションのユーザーアクセス権限を管理するには、次のようなアクセス権限ポリシーを記述できます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

`GameScores` テーブル (`Resource` 要素) の特定の DynamoDB アクション (`Action` 要素) に対する許可の付与に加えて、`Condition` 要素は、次のように許可を制限する DynamoDB に固有の次の条件キーを使用します。

- `dynamodb:LeadingKeys` – この条件キーにより、ユーザーはパーティションのキーバリューがユーザー ID に一致する項目にのみアクセスできます。この ID、`${www.amazon.com:user_id}` は、置換変数です。置換変数の詳細については、「ウェブ ID フェデレーションの使用」を参照してください。
- `dynamodb:Attributes` – この条件キーは指定された属性へのアクセスを制限し、許可ポリシーにリストされたアクションのみが、これらの属性の値を返すことができるようにします。さらに、`StringEqualsIfExists` 句は、アプリケーションが常に対象となる特定の属性のリストを提供し、すべての属性をリクエストできないようにします。

IAM ポリシーが評価されると、結果は必ず true (アクセス許可) または false (アクセス拒否) になります。`Condition` 要素のいずれかの部分が false の場合、ポリシー全体が false と評価され、アクセスは拒否されます。

Important

`dynamodb:Attributes` を使用する場合、ポリシーでリストに記載されているテーブルとすべてのセカンダリインデックスについて、すべてのプライマリキー属性とインデックス

キー属性の名前を指定する必要があります。指定しなかった場合、DynamoDB は、このキー属性を使用して、リクエストされたアクションを実行できません。

IAM ポリシードキュメントには、次の Unicode 文字のみを含めることができます。水平タブ (U+0009)、ラインフィード (U+000A)、キャリッジリターン (U+000D)、および U+0020 ~ U+00FF の範囲内の文字。

条件の指定: 条件キーの使用

AWS には、アクセス制御のために IAM をサポートするすべての AWS サービスに合わせて事前定義された一連の条件キー (AWS 全体に対する条件キー) が用意されています。たとえば、`aws:SourceIp` 条件キーを使用して、リクエスタの IP アドレスを確認してから、アクションの実行を許可できます。AWS 全体を対象とするキーの詳細およびリストについては、IAM ユーザーガイドの使用可能な条件キーを参照してください。

以下の表では、DynamoDB に適用される DynamoDB サービス固有の条件キーを示しています。

DynamoDB 条件キー	説明
`dynamodb:LeadingKeys`	テーブルの最初のキー属性、つまりパーティションキーを表します。キーが 1 つの項目のアクションで使用される場合も、キー名 `LeadingKeys` は複数形になります。また、条件で `ForAllValues` を使用するときには、`LeadingKeys` 修飾子を使用する必要があります。
`dynamodb:Select`	`Select` または `Query` リクエストの `Scan` パラメータを表します。`Select` には次のいずれかの値を指定できます。 • `ALL_ATTRIBUTES` • `ALL_PROJECTED_ATTRIBUTES` • `SPECIFIC_ATTRIBUTES` • `COUNT`
`dynamodb:Attributes`	リクエスト内の属性名のリスト、またはリクエストから返される属性を表します。`Attributes` の値は、次に示すように、特定の DynamoDB API アクションのパラメータと同じ方法で名前が付けられ、同じ意味を持ちます。

DynamoDB 条件キー	説明
	- AttributesToGet 用途: BatchGetItem, GetItem, Query, Scan - AttributeUpdates 用途: UpdateItem - Expected 用途: DeleteItem, PutItem, UpdateItem - Item 用途: PutItem - ScanFilter 用途: Scan
dynamodb:ReturnValues	リクエストの ReturnValues パラメータを表します。ReturnValues には次のいずれかの値を指定できます。 - ALL_OLD - UPDATED_OLD - ALL_NEW - UPDATED_NEW - NONE
dynamodb:ReturnConsumedCapacity	リクエストの ReturnConsumedCapacity パラメータを表します。ReturnConsumedCapacity には次のいずれかの値を指定できます。 - TOTAL - NONE

ユーザーアクセスの制限

多くの IAM アクセス権限ポリシーでは、パーティションキーの値がユーザー識別子と一致するテーブルの項目へのアクセスのみをユーザーに許可します。たとえば、

前述のゲームアプリケーションは、この方法でアクセスを制限し、ユーザーは自分のユーザー ID に関連付けらたゲームデータのみにアクセスできるようにします。IAM 置換変数 `${www.amazon.com:user_id}`、`${graph.facebook.com:id}`、および `${accounts.google.com:sub}` には、Login with Amazon、Facebook、および Google のユーザー識別子が格納されます。アプリケーションがこのようなアイデンティティプロバイダーのいずれかにログインし、この識別子を取得する方法については、「ウェブ ID フェデレーションの使用」を参照してください。

> Note
>
> 以下のセクションの各例では、`Effect` 句を `Allow` に設定し、許可されるアクション、リソース、およびパラメータのみを指定します。IAM ポリシーで明示的に指定されたものへのアクセスだけが許可されます。
> 場合によっては、拒否ベースとなるようにこのポリシーを書き直すことができます。つまり、`Effect` 句を `Deny` に設定して、ポリシーのすべてのロジックを逆にします。ただし、拒否ベースのポリシーを DynamoDB で使用しないことをお勧めします。このようなポリシーは、許可ベースのポリシーと比べて、正しく書き込むことが難しいためです。また、DynamoDB API への将来の変更 (または、既存の API 入力への変更) が原因で、拒否ベースのポリシーが機能しなくなる可能性があります。

ポリシー例: きめ細かなアクセスコントロールのための IAM ポリシー条件の使用

このセクションでは、DynamoDB テーブルとインデックスに対してきめ細かなアクセス制御を実装するための一部のポリシーについて説明します。

> Note
>
> すべての例で、us-west-2 リージョンを使用し、架空のアカウント ID を含めています。

次の動画では、IAM ポリシー条件を使用した DynamoDB でのきめ細かなアクセスコントロールについて説明します。

1: 特定のパーティションのキー値を持つ項目へのアクセスを制限するアクセス権限の付与

以下の許可ポリシーは、`GamesScore` テーブルで一連の DynamoDB アクションに対する許可を付与します。このポリシーは `dynamodb:LeadingKeys` 条件キーを使用して、`UserID` パーティション

キーの値が、このアプリケーション用の Login with Amazon の一意のユーザー ID と一致する項目のみでユーザーアクションを制限します。

> **Important**
>
> Scan 用のアクセス権限は、アクションのリストに含まれていません。これは、Scan が主要なキーにかかわらずすべての項目を返すためです。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"FullAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

Note

ポリシー変数を使用する場合は、2012-10-17 をポリシー内で明示的に指定する必要があります。アクセスポリシー言語のデフォルトバージョン、2008-10-17 では、ポリシー変数をサポートしていません。

読み取りアクセスを実装するには、データを変更可能なアクションを削除します。次のポリシーでは、読み込み専用アクセスを提供するアクションだけが条件に追加されています。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ReadOnlyAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

Important

dynamodb:Attributes を使用する場合、ポリシーでリストに記載されているテーブルとあらゆるセカンダリインデックスについて、すべてのプライマリキー属性とインデックス

キー属性の名前を指定する必要があります。指定しなかった場合、DynamoDB は、このキー属性を使用して、リクエストされたアクションを実行できません。

2: テーブルの特定の属性へのアクセスを制限するアクセス権限の付与

次のアクセス権限ポリシーは、`dynamodb:Attributes` 条件キーを追加して、テーブルの 2 つの特定の属性のみへアクセスを許可します。この属性に対して、読み込み、書き込み、または条件付き書き込みまたはスキャンフィルタでの評価を実行できます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToSpecificAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "UserId",
                  "TopScore"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
```

```
    ]
}
```

> Note
>
> このポリシーでは、許可リスト手法を使用し、列挙された一連の属性へのアクセスを許可します。代わりに、他の属性へのアクセスを拒否する同等のポリシーを記述することもできますが、この拒否リスト手法はお勧めしません。ユーザーは、ウィキペディア (http://en.wikipedia.org/wiki/Principle_of_least_privilege) で説明されている最小権限の原則に従って、これらの拒否属性の名前を判別し、拒否属性を指定する代わりに、許可リストアプローチを使用して、許可されたすべての値を列挙することができます。

このポリシーでは、PutItem、DeleteItem、または BatchWriteItem は許可されません。これらのアクションでは常に前の項目全体が置き換えられ、アクセスが許可されていない属性の以前の値を、ユーザーが削除できる可能性があるためです。

アクセス権限ポリシーの StringEqualsIfExists 句により、以下を実施することができます。

- ユーザーが Select パラメータを指定する場合、その値は SPECIFIC_ATTRIBUTES である必要があります。この要件により、インデックスの射影からなど、API アクションが許可されていない属性を返さないようにします。
- ユーザーが ReturnValues パラメータを指定する場合、その値は、NONE、UPDATED_OLD、または UPDATED_NEW にする必要があります。これが必要なのは、UpdateItem アクションでは、置換する前に項目が存在するかどうかをチェックするために、および、リクエストされた場合に前の属性値を返すことができるように、暗黙的な読み込みオペレーションが実行されるためです。ReturnValues をこのように制限することで、確実にユーザーが許可された属性だけを読み込むまたは書き込むことができるようにします。
- StringEqualsIfExists 節により、許可されたアクションのコンテキストで、リクエストごとに Select または ReturnValues パラメータのいずれか 1 つだけを使用できるようにすることができます。

次に、このポリシーのバリエーションをいくつか示します。

- 読み込みアクションだけを許可するには、許可されるアクションのリストから UpdateItem を削除できます。残りのどのアクションも ReturnValues を受け付けないので、条件から ReturnValues を削除できます。また、StringEqualsIfExists パラメーターには必ず値 (特

に指定のない限り、StringEquals) があるので、Select を ALL_ATTRIBUTES に変更できます。

- 書き込みアクションだけを許可するには、許可されるアクションのリストから、UpdateItem を除くすべてのアクションを削除します。UpdateItem では Select パラメーターを使用しないので、条件から Select を削除できます。また、StringEqualsIfExists パラメーターには必ず値 (特に指定のない限り、StringEquals) があるので、ReturnValues を NONE に変更する必要があります。
- 名前がパターンに一致するすべての属性を許可するには、StringLike ではなく StringEquals を使用し、複数文字パターン照合ワイルドカード文字 (*) を使用します。

3: 特定の属性で更新を回避するアクセス権限の付与

以下のアクセス権限のポリシーでは、dynamodb:Attributes 条件キーによって識別される特定の属性のみを更新するようユーザーアクセスを制限します。StringNotLike 条件によって、アプリケーションが、dynamodb:Attributes 条件キーを使用して指定された属性を更新できないようになります。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"PreventUpdatesOnCertainAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
         "Condition":{
            "ForAllValues:StringNotLike":{
               "dynamodb:Attributes":[
                  "FreeGamesAvailable",
                  "BossLevelUnlocked"
               ]
            },
            "StringEquals":{
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
```

```
                }
            }
        }
    ]
}
```

次の点に注意してください。

- UpdateItem アクションは、他の書き込みアクションと同じように、更新前の値と更新後の値を返すことができるように、項目への読み取りアクセスを必要とします。ポリシーでは、dynamodb:ReturnValues 条件キーを指定して、更新が許可される属性のみにアクセスするようアクションを制限します。条件キーは、リクエストの ReturnValues を制限して NONE、UPDATED_OLD、または UPDATED_NEW のみを指定し、ALL_OLD または ALL_NEW は含まれません。
- PutItem および DeleteItem アクションは項目全体を置き換えるため、アプリケーションは任意の属性を変更することができます。したがって、特定の属性のみを更新するようアプリケーションを制限するときは、それらの API 用のアクセス権限を付与しないようにします。

4: インデックスで射影された属性のみのクエリを実行するアクセス権限の付与

以下の許可ポリシーでは、dynamodb:Attributes 条件キーを使用して、セカンダリインデックス (TopScoreDateTimeIndex) でクエリを許可します。また、このポリシーでは、インデックスに射影された特定の属性だけをリクエストするようクエリを制限します。

アプリケーションがクエリで属性のリストを指定するよう要求するには、ポリシーで dynamodb:Select 条件キーを指定して、DynamoDB Query アクションの Select パラメータが SPECIFIC_ATTRIBUTES であることを要求します。属性のリストは、dynamodb:Attributes 条件キーを使用して提供される特定のリストに制限されます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryOnlyProjectedIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "TopScoreDateTime",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "Attempts"
               ]
            },
            "StringEquals":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

次のアクセス権限ポリシーは、似ていますが、クエリでは、インデックスに射影されたすべての属性をリクエストする必要があります。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryAllIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:Select":"ALL_PROJECTED_ATTRIBUTES"
            }
         }
      }
   ]
}
```

5: 特定の属性とパーティションキー値へのアクセスを制限するアクセス権限の付与

以下の許可ポリシーでは、特定の DynamoDB アクション (Action 要素で指定) をテーブルとテーブルインデックス (Resource 要素で指定) で許可します。このポリシーは、dynamodb:LeadingKeys 条件キーを使用して、パーティションキーの値がユーザーの Facebook ID に一致する項目のみにアクセス権限を制限します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToCertainAttributesAndKeyValues",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${graph.facebook.com:id}"
               ],
               "dynamodb:Attributes":[
                  "attribute-A",
                  "attribute-B"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
```

```
                    ]
                }
            }
        }
    ]
}
```

次の点に注意してください。

- ポリシー (UpdateItem) によって許可された書き込みアクションでは、attribute-A または attribute-B のみを変更できます。
- ポリシーでは UpdateItem が許可されるため、アプリケーションは新しい項目を挿入でき、その非表示の属性は、新しい項目では null として表示されます。この属性が TopScoreDateTimeIndex に射影されている場合、テーブルからのフェッチを引き起こすクエリを防止するという利点がポリシーに追加されます。
- アプリケーションは、dynamodb:Attributes に列挙されている属性以外の属性を読み込むことができません。このポリシーにより、アプリケーションは、読み込みリクエストで Select パラメータを SPECIFIC_ATTRIBUTES に設定する必要があり、許可リストの属性だけをリクエストすることができます。書き込みリクエストの場合、アプリケーションは、ReturnValues を ALL_OLD または ALL_NEW に設定できません。また、他の属性に基づく条件付き書き込みオペレーションを実行できません。

関連トピック

- Amazon DynamoDB の Identity and Access Management
- DynamoDB API の許可: アクション、リソース、条件リファレンス

ウェブ ID フェデレーションの使用

多数のユーザーを対象とするアプリケーションを記述している場合、必要に応じて、ウェブアイデンティティフェデレーションを使用して、認証と認可を実行できます。ウェブ ID フェデレーションは、個々の ユーザーを作成する必要性を排除します。代わりに、ユーザーは ID プロバイダーにサインインした後、一時的なセキュリティ資格情報を AWS Security Token Service (AWS STS) から取得できます。アプリケーションでは、この認証情報を使用して AWS サービスにアクセスできます。

ウェブアイデンティティフェデレーションでは、次のアイデンティティプロバイダーをサポートしています。

- Login with Amazon
- Facebook
- Google

ウェブアイデンティティフェデレーションに関するその他のリソース

ウェブアイデンティティフェデレーションの詳細を理解するのに、次のリソースが役に立ちます。

- AWS デベロッパーブログの「Web Identity Federation using the AWS SDK for .NET 」の記事では、ウェブ ID フェデレーションを Facebookで使用する方法について説明しています。記事には、ウェブ ID を持つ ロールを想定する方法や、一時的なセキュリティ認証情報を使用して AWS リソースにアクセスする方法を示す C# のコードスニペットが含まれています。
- AWS Mobile SDK for iOS と AWS Mobile SDK for Android には、サンプルアプリが含まれています。このアプリケーションには、アイデンティティプロバイダーを呼び出す方法、そのプロバイダーからの情報を使用して、一時的なセキュリティ認証情報を取得および使用する方法を示すコードが含まれています。
- 記事「Web Identity Federation with Mobile Applications」では、ウェブ ID フェデレーションについて説明し、ウェブ ID フェデレーションを使用して AWS リソースにアクセスする使用方法の例を挙げています。

ウェブ ID フェデレーションのポリシー例

DynamoDB でウェブ ID フェデレーションを使用する方法を示すには、詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用 で導入された GameScores テーブルに再度アクセスしてください。次に、GameScores のプライマリキーを示します。

テーブル名	プライマリキーのタイプ	パーティションキーの名前と型	ソートキーの名前と型
GameScores (UserId、GameTitle、...)	複合	属性名: UserId 型: 文字列	属性名: GameTitle 型: 文字列

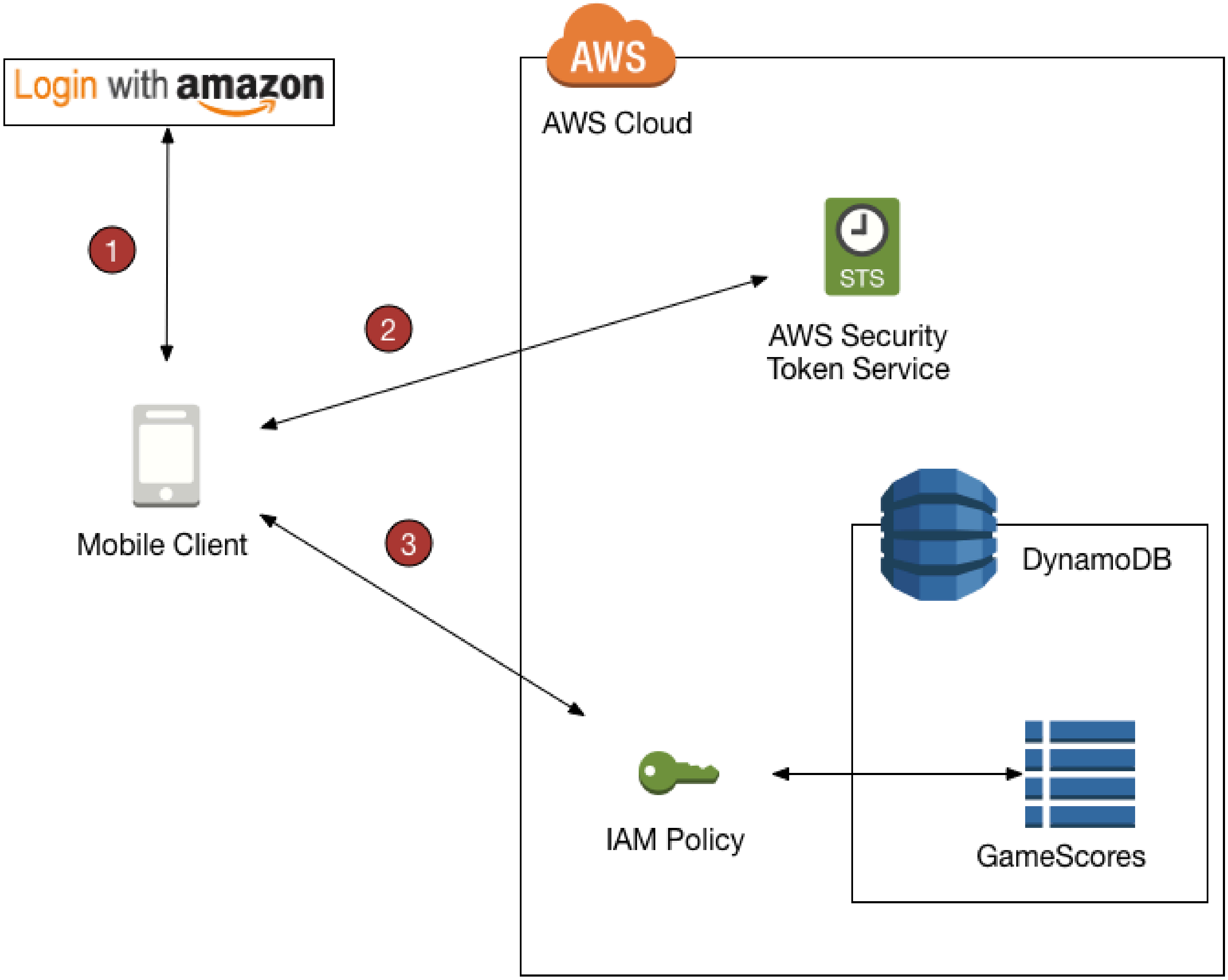
ここで、あるモバイルゲームアプリケーションがこのテーブルを使用し、そのアプリケーションでは、数千人、数百万人のユーザーをサポートする必要があるとします。この規模では、個別のアプ

リケーションユーザーを管理し、各ユーザーが GameScores テーブルの自分のデータにだけアクセスできることを確実にするのは非常に難しいです。多くのユーザーは、Facebook、Google、Login with Amazon などのサードパーティーの ID プロバイダーのアカウントを既に持っています。そのため、このようなプロバイダーのいずれかを利用して、認証タスクを行うことができます。

ウェブ ID フェデレーションを使用して認証を行うには、アプリケーションデベロッパーは、そのアプリケーションをアイデンティティプロバイダー (Login with Amazon など) に登録して、一意のアプリケーション ID を取得する必要があります。次に、デベロッパーはロールを作成する必要があります (この例では、このロールには GameRole という名前が付いています)。このロールには、IAM ポリシードキュメントを添付する必要があります。このドキュメントでは、アプリケーションが GameScores テーブルにアクセスできるように条件を指定します。

ユーザーは、ゲームをプレイするとき、ゲームアプリケーション内から Login with Amazon アカウントにサインインします。その後アプリケーションが AWS Security Token Service (AWS STS) を呼び出します。その際には Login with Amazon アプリ ID を指定し、GameRole でメンバーシップをリクエストします。AWS STS は一時的な AWS 認証情報をアプリケーションに返し、GameRole ポリシードキュメントに基づいて GameScores テーブルへのアクセスを許可します。

次の図は、これらの要素を組み合わせるとどのようになるかを示しています。



ウェブ ID フェデレーションの概要

1. アプリケーションは、サードパーティのアイデンティティプロバイダーを呼び出して、ユーザーとアプリケーションを認証します。アイデンティティプロバイダーは、ウェブアイデンティティトークンをアプリケーションに返します。
2. アプリは AWS STS を呼び出し、ウェブアイデンティティトークンを入力として渡します。AWS STS は、アプリケーションを認可し、一時的な AWS アクセス認証情報をアプリケーションに渡します。アプリケーションは、IAM ロール (GameRole) を想定し、そのロールのセキュリティポリシーに従って AWS リソースにアクセスすることが許可されます。

3. アプリケーションは、DynamoDB をコールして、GameScores テーブルにアクセスします。アプリケーションは GameRole を想定しているので、そのロールに関連付けられたセキュリティポリシーの影響を受けます。ポリシードキュメントによって、ユーザーに属していないデータにアプリケーションはアクセスできません。

繰り返しになりますが、詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用に示されている GameRole のセキュリティポリシーは次のとおりです。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
```

```
                }
            }
        }
    ]
}
```

Condition 節によって、アプリケーションに表示される GameScores の項目が決まります。Login with Amazon ID と UserId の GameScores パーティションキーの値を比較することで、これを行います。このポリシーに列挙されている DynamoDB アクションのいずれかを使用して処理できるのは、現在のユーザーに属する項目だけです。テーブルのその他の項目にはアクセスできません。さらに、ポリシーに列挙されている特定の属性にだけアクセスできます。

ウェブ ID フェデレーションを使用するための準備

アプリケーションデベロッパーがアプリケーションにウェブ ID フェデレーションを使用する場合、次のステップに従います。

1. 開発者としてサードパーティーの ID プロバイダーにサインアップします。次の外部リンクには、サポートされる ID プロバイダーへのサインアップに関する情報が記載されています。
    - Login with Amazon 開発者センター
    - Facebook サイトの登録
    - Google サイトの OAuth 2.0 を使用して Google API にアクセスする
2. アプリケーションを ID プロバイダーに登録します。登録すると、プロバイダーからアプリケーションに固有の ID が提供されます。複数のアイデンティティプロバイダーで機能するようにアプリケーションを構築する場合は、各プロバイダーからアプリケーション ID を取得する必要があります。
3. 1 つ以上の IAM ロールを作成します。各アプリケーションのアイデンティティプロバイダーごとに 1 つのロールが必要です。たとえば、ユーザーが Login with Amazon を使用してサインインするアプリケーションで想定できるロール、ユーザーが Facebook を使用してサインインする同じアプリケーションの 2 つ目のロール、およびユーザーが Google を使用してサインインするアプリケーションの 3 つ目のロールを作成します。

    ロール作成プロセスの中で、 ポリシーをロールにアタッチする必要があります。ポリシードキュメントでは、アプリケーションに必要な DynamoDB リソース、およびそれらのリソースにアクセスするための許可を定義する必要があります。

詳細については、IAM ユーザーガイドのウェブ ID フェデレーションについてを参照してください。

ⓘ Note

AWS Security Token Service の代わりに、Amazon Cognito を使用できます。Amazon Cognito は、モバイルアプリケーションの一時的な認証情報を管理するためのおすすめサービスです。詳細については、Amazon Cognito 開発者ガイド」の「認証情報の取得」を参照してください。

DynamoDB コンソールを使用して IAM ポリシーを生成する

DynamoDB コンソールでは、ウェブ ID フェデレーションで使用する IAM ポリシーを作成できます。作成するには、DynamoDB テーブルを選択し、ポリシーに含まれるアイデンティティプロバイダー、アクション、および属性を指定します。DynamoDB コンソールによって、IAM ロールにアタッチすることができるポリシーが生成されます。

1. AWS Management Console マネジメントコンソールにサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. テーブルの一覧で、IAM ポリシーを作成するテーブルを選択します。
4. [アクション] ボタン、[アクセス制御ポリシーの作成] の順に選択します。
5. ポリシーの ID プロバイダー、アクション、および属性を選択します。

   すべての設定が正しいことを確認したら、[ポリシーの生成] を選択します。生成されたポリシーが表示されます。
6. [ドキュメンテーションを参照] を選択し、生成されたポリシーを IAM ロールにアタッチするために必要な手順に従います。

ウェブ ID フェデレーションを使用するアプリケーションの記述

ウェブ ID フェデレーションを使用するには、アプリケーションで、作成した IAM ロールを引き受ける必要があります。その時点から、アプリケーションは、そのロールに関連付けられたアクセスポリシーに従います。

実行時、アプリケーションがウェブアイデンティティフェデレーションを使用する場合、次の手順に従う必要があります。

1. サードパーティーの ID プロバイダーと認証を行います。アプリケーションは、ID プロバイダーが提供するインターフェイスを使用してそのプロバイダーを呼び出す必要があります。ユーザー

を認証するときの正確な方法は、プロバイダーおよびアプリケーションを実行しているプラットフォームによって異なります。一般的に、ユーザーがまだサインインしていない場合、アイデンティティプロバイダーはそのプロバイダーのサインインページを表示するように対処します。

アイデンティティプロバイダーはユーザーを認証した後、ウェブアイデンティティトークンをアプリケーションに返します。このトークンの形式は、プロバイダーによって異なりますが、通常は、非常に長い文字列です。

2. 一時的な AWS セキュリティ認証情報を取得します。取得するために、アプリケーションは `AssumeRoleWithWebIdentity` リクエストを AWS Security Token Service (AWS STS) に送信します。このリクエストには次のものが含まれます。

   - 前の手順からのウェブアイデンティティトークン
   - アイデンティティプロバイダーからのアプリケーション ID
   - このアプリケーションのこのアイデンティティプロバイダー用に作成した IAM ロールの Amazon リソースネーム (ARN)

   AWS STS は、一定の時間 (デフォルトでは、3,600 秒) 経過した後に失効する AWS セキュリティ認証情報のセットを返します。

   次に、`AssumeRoleWithWebIdentity` で AWS STS アクションを実行するときのサンプルリクエストとレスポンスを示します。ウェブアイデンティティトークンは、Login with Amazon アイデンティティプロバイダーから取得しました。

```
GET / HTTP/1.1
Host: sts.amazonaws.com
Content-Type: application/json; charset=utf-8
URL: https://sts.amazonaws.com/?ProviderId=www.amazon.com
&DurationSeconds=900&Action=AssumeRoleWithWebIdentity
&Version=2011-06-15&RoleSessionName=web-identity-federation
&RoleArn=arn:aws:iam::123456789012:role/GameRole
&WebIdentityToken=Atza|IQEBLjAsAhQluyKqyBiYZ8-kclvGTYM81e...(remaining characters
 omitted)
```

```
<AssumeRoleWithWebIdentityResponse
  xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
  <AssumeRoleWithWebIdentityResult>
    <SubjectFromWebIdentityToken>amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE</
SubjectFromWebIdentityToken>
    <Credentials>
```

```
      <SessionToken>AQoDYXdzEMf//////////wEa8AP6nNDwcSLnf+cHupC...(remaining
 characters omitted)</SessionToken>
      <SecretAccessKey>8Jhi60+EWUUbbUShTEsjTxqQtM8UKvsM6XAjdA==</SecretAccessKey>
      <Expiration>2013-10-01T22:14:35Z</Expiration>
      <AccessKeyId>06198791C436IEXAMPLE</AccessKeyId>
    </Credentials>
    <AssumedRoleUser>
      <Arn>arn:aws:sts::123456789012:assumed-role/GameRole/web-identity-federation</
Arn>
      <AssumedRoleId>AROAJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
    </AssumedRoleUser>
  </AssumeRoleWithWebIdentityResult>
  <ResponseMetadata>
    <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
  </ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>
```

3. AWSリソースにアクセスする AWS STS からのレスポンスには、DynamoDB リソースにアクセスするためにアプリケーションに必要な情報が格納されています。

   - `AccessKeyID`、`SecretAccessKey`、および `SessionToken` フィールドには、このユーザーとこのアプリケーションにのみ有効なセキュリティ認証情報が入っています。
   - `Expiration` フィールドは、この認証情報の時間制限を示します。この時間を経過すると、認証情報は無効になります。
   - `AssumedRoleId` フィールドには、アプリケーションによって想定されたセッション固有の IAM ロールの名前が含まれています。アプリケーションは、このセッションの期間中、IAM ポリシードキュメントのアクセス制御に従います。
   - `SubjectFromWebIdentityToken` フィールドには、この特定のアイデンティティプロバイダー用の IAM ポリシー変数に表示される一意の ID が含まれています。次に、サポートされるプロバイダーの IAM ポリシー変数と、その値の例を示します。

ポリシー変数	値の例
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`123456789012345678901`

このポリシー変数が使用される IAM ポリシーの例については、ポリシー例: きめ細かなアクセスコントロールのための IAM ポリシー条件の使用 を参照してください。

AWS STS が一時的なアクセス認証情報を生成する詳しい方法については、IAM ユーザーガイドの一時的なセキュリティ認証情報のリクエストを参照してください。

DynamoDB API の許可: アクション、リソース、条件リファレンス

Amazon DynamoDB の Identity and Access Management を設定し、IAM アイデンティティにアタッチできる許可ポリシー (アイデンティティベースのポリシー) を書き込む場合は、リファレンスとして IAM ユーザーガイドの Amazon DynamoDB のアクション、リソース、および条件キーリストを使用できます。このページは、各 DynamoDB API オペレーション、アクションを実行するための許可を付与できる対応するアクション、および許可を付与できる AWS リソースを示しています。ポリシーの `Action` フィールドでアクションを指定し、ポリシーの `Resource` フィールドでリソースの値を指定します。

DynamoDB ポリシーで AWS 全体の条件キーを使用して、条件を表現することができます。AWS 全体を対象とするすべてのキーのリストについては、IAM ユーザーガイドの IAM JSON ポリシーエレメントリファレンスを参照してください。

AWS 全体を対象とする条件キーに加え、DynamoDB には条件内で使用可能な独自の固有キーもあります。詳細については、「」を参照してください詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用

関連トピック

- Amazon DynamoDB の Identity and Access Management
- 詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用

DynamoDB アクセラレーターでの Identity and Access Management

DynamoDB アクセラレーター (DAX) は DynamoDB と連携して動作し、アプリケーションにキャッシングレイヤーをシームレスに追加するように設計されています。ただし、DAX および DynamoDB には、個別のアクセス制御のメカニズムがあります。どちらのサービスもそれぞれのセキュリティポリシーの実装に AWS Identity and Access Management (IAM) を使用しますが、セキュリティモデルは DAX と DynamoDB で異なります。

DAX の Identity and Access Management に関する詳細については、DAX のアクセスコントロール を参照してください。

DynamoDB の業界別のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンスプログラム別の範囲」の「AWS のサービス」と「」の「AWS のサービス」を参照し、関心のあるコンプライアンスプログラムを選択してください。一般的な情報については、AWS コンプライアンスプログラム を参照してください。

AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

AWS のサービスを使用する際のユーザーのコンプライアンス責任は、ユーザーのデータの機密性や貴社のコンプライアンス目的、適用される法律および規制によって決まります。AWS では、コンプライアンスに役立つ次のリソースを提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド - これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境を AWS にデプロイするためのステップを示します。
- 「Amazon Web Services での HIPAA のセキュリティとコンプライアンスのためのアーキテクチャ」 - このホワイトペーパーは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法を説明しています。

> Note
>
> すべての AWS のサービスが HIPAA 適格であるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスのリソース – このワークブックおよびガイドのコレクションは、顧客の業界と拠点に適用されるものである場合があります。
- AWS Customer Compliance Guide — コンプライアンスの観点から見た責任共有モデルを理解できます。このガイドは、AWS のサービスを保護するためのベストプラクティスを要約したものであり、複数のフレームワーク (米国標準技術研究所 (NIST)、ペイメントカード業界セキュリティ標準評議会 (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティ統制へのガイダンスがまとめられています。
- 「AWS Config デベロッパーガイド」の「ルールでのリソースの評価」 - AWS Config サービスは、自社のプラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。

- AWS Security Hub - この AWS のサービスは、AWS 内のセキュリティ状態の包括的なビューを提供します。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – この AWS のサービスは、環境をモニタリングして、疑わしいアクティビティや悪意のあるアクティビティがないか調べることで、AWS アカウント、ワークロード、コンテナ、データに対する潜在的な脅威を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager - この AWS のサービスは、AWS の使用状況を継続的に監査して、リスクの管理方法や、規制および業界標準へのコンプライアンスの管理方法を簡素化するために役立ちます。

Amazon DynamoDB での耐障害性と災害対策

AWS のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心として構築されます。AWSリージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

データまたはアプリケーションを遠方でレプリケートする必要がある場合は、AWS ローカルリージョンを使用します。AWS ローカルリージョンは、既存の AWS リージョンを補完するように設計された単一のデータセンターです。すべての AWS リージョンと同じように、AWS ローカルリージョンは他の AWS リージョンから完全に分離されています。

AWS リージョンとアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

Amazon DynamoDB では、AWS グローバルインフラストラクチャに加えて、データの耐障害性と Backup のニーズに対応できるように複数の機能を提供しています。

オンデマンドバックアップと復元

DynamoDB には、オンデマンドバックアップ機能があります。この機能により、長期間の保存とアーカイブのために、テーブルの完全なバックアップを作成できます。詳細については、「DynamoDB のオンデマンドバックアップおよび復元」を参照してください。

ポイントインタイムリカバリ

ポイントインタイムリカバリを使用することで、DynamoDB テーブルを偶発的な書き込みや削除のオペレーションから保護できます。ポイントインタイムリカバリを有効化すれば、オンデマンドバックアップの作成、維持、スケジュールを心配する必要はありません。詳細については、「DynamoDB のポイントインタイムリカバリ」を参照してください。

AWS リージョン間でのグローバルテーブルの同期

DynamoDB では、一貫性のある高速パフォーマンスを維持しながら、スループットとストレージの要件を処理できるように、テーブルのデータとトラフィックが十分な数のサーバーに自動的に分散されます。また、すべてのデータをソリッドステートディスク (SSD) に保存し、AWS リージョン内の複数のアベイラビリティーゾーン間で自動的にレプリケートするため、組み込みの高い可用性とデータ堅牢性が実現します。グローバルテーブルを使用して、DynamoDB テーブルを AWS リージョン間で同期させることができます。

Amazon DynamoDB のインフラストラクチャセキュリティ

マネージドサービスとして、Amazon DynamoDB は、AWS Well-Architected フレームワークにあるインフラストラクチャ保護で説明されている AWS グローバルネットワークセキュリティ手順によって保護されます。

AWS が公開した API コールを使用して、ネットワーク経由で DynamoDB にアクセスします。クライアントは TLS (Transport Layer Security) バージョン 1.2 または 1.3 を使用できます。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもサポートしている必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token ServiceAWS STSを使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

DynamoDB 用の 仮想プライベートクラウド (VPC) エンドポイントを使用することで、VPC 内の Amazon EC2 インスタンスがパブリックインターネットにさらされることなく、プライベート IP ア

ドレスを使用して DynamoDB にアクセスできるようになります。詳細については、「Amazon VPC エンドポイントを使用して DynamoDB にアクセスする」を参照してください。

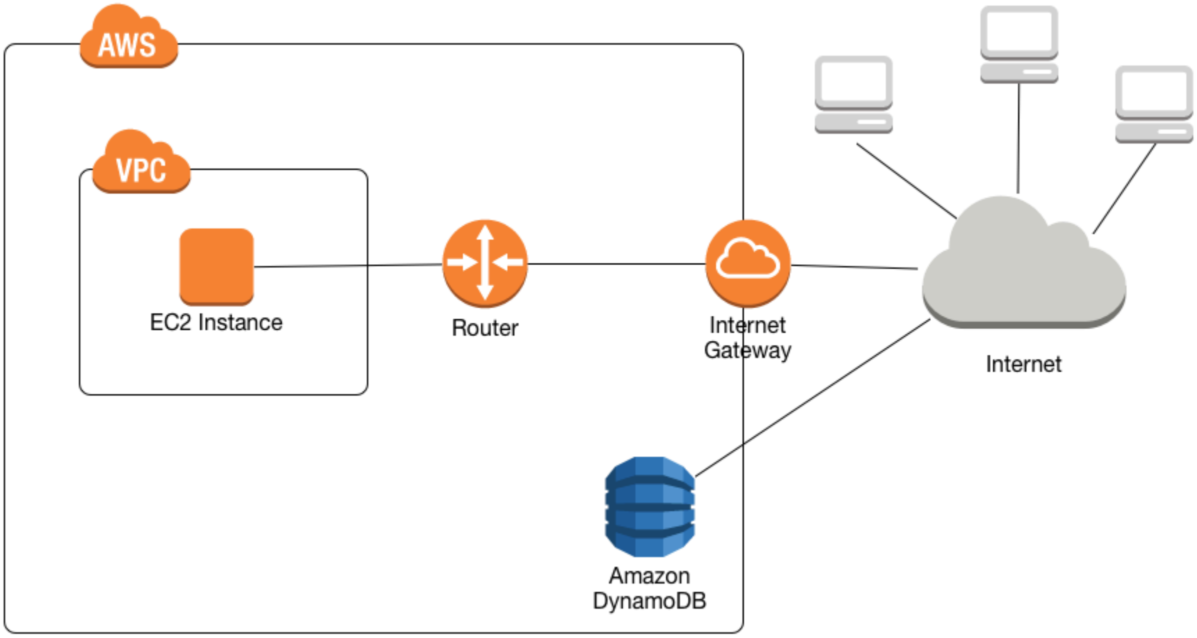
Amazon VPC エンドポイントを使用して DynamoDB にアクセスする

セキュリティ上の理由から、多くの AWS ユーザーがアプリケーションを Amazon Virtual Private Cloud 環境 (Amazon VPC) 内で実行しています。Amazon VPC を使用すると、Amazon EC2 インスタンスを仮想プライベートクラウドで作成できます。そのため、パブリックインターネットなどの他のネットワークから論理的に分離されます。Amazon VPC を使用すると、IP アドレスの範囲、サブネット、ルーティングテーブル、ネットワークゲートウェイ、セキュリティ設定を適切に管理できます。

> Note
>
> 2013 年 12 月 4 日以降に AWS アカウントを作成した場合は、各 AWS リージョンにデフォルトの VPC が用意されています。デフォルトの VPC は、使用できる状態になっています。追加で設定手順を実行することなく、すぐに利用開始できます。
>
> デフォルト VPC の詳細については、Amazon VPC ユーザーガイドの「デフォルト VPC とデフォルトサブネット」を参照してください。

パブリックインターネットにアクセスするには、VPC にインターネットゲートウェイ (VPC をインターネットに接続する仮想ルーター) が必要です。これにより、VPC 内の Amazon EC2 で実行されているアプリケーションで、Amazon DynamoDB などのインターネットリソースにアクセスすることが可能になります。

デフォルトでは、DynamoDB との通信において、SSL/TLS 暗号化を使用してネットワークトラフィックを保護する HTTPS プロトコルが使用されます。次の図は、VPC 内の Amazon EC2 インスタンスから DynamoDB にアクセスするために、VPC エンドポイントではなくインターネットゲートウェイを使用する例を示しています。



多くのお客様が、パブリックインターネット間のデータ送受信に関して、プライバシーとセキュリティに関する正当な懸念を抱いています。これらの懸念を解決するために、仮想プライベートネットワーク (VPN) を使用して、すべての DynamoDB ネットワークトラフィックをお客様の企業ネットワークのインフラストラクチャ経由でルーティングできます。ただし、このアプローチでは、帯域幅や可用性の課題が生じる場合があります。

DynamoDB 用の VPC エンドポイントでは、これらの課題は軽減されます。DynamoDB 用の VPC エンドポイントを使用すると、VPC 内の Amazon EC2 インスタンスがパブリックインターネットにさらされることなく、プライベート IP アドレスを使用して DynamoDB にアクセスできるようになります。EC2 インスタンス にパブリック IP アドレスは必要ありません。また、VPC にインターネットゲートウェイ、NAT デバイス、仮想プライベートゲートウェイは不要です。DynamoDB へのアクセスを制御するには、エンドポイントのポリシーを使用します。VPC と AWS サービス間のトラフィックは、Amazon ネットワークを離れません。

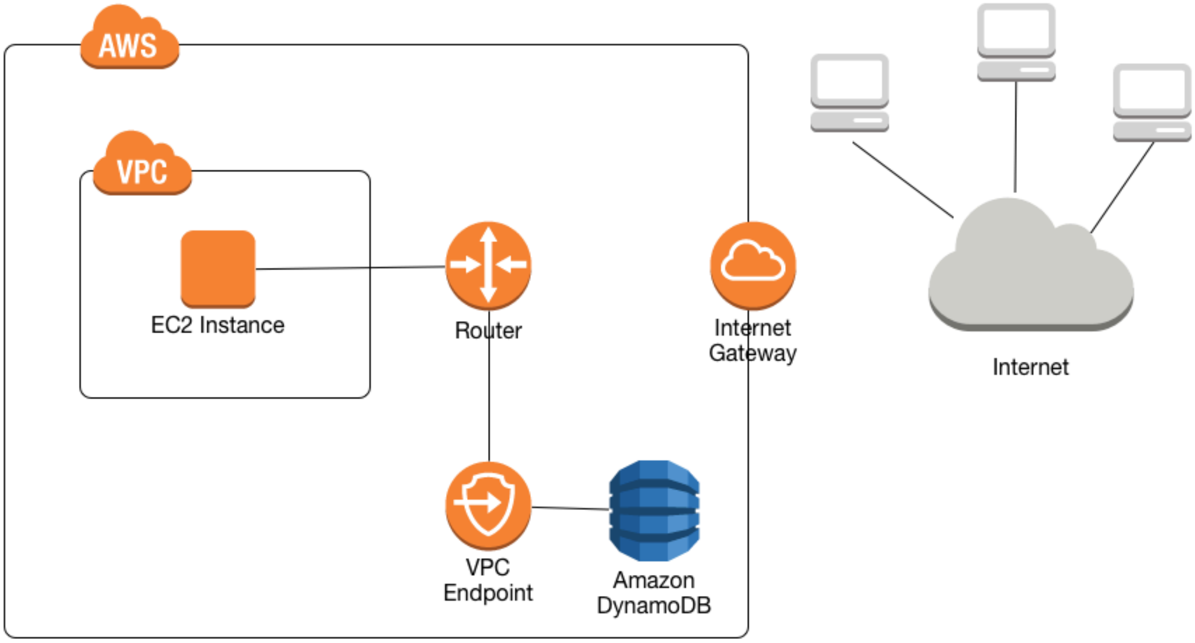
Note

パブリック IP アドレスを使用する場合でも、AWS でホストされているインスタンスとサービスの間で発生するすべての VPC 通信は、AWS ネットワーク内でのプライベートな通信となります。AWS ネットワークから発信され、送信先が AWS ネットワーク上のパケット

は、AWS 中国リージョンとの送受信されるトラフィックを除き、AWS グローバルネットワークに残ります。

DynamoDB 用の VPC エンドポイントを作成する際、リージョン内の DynamoDB エンドポイント (例: dynamodb.us-west-2.amazonaws.com) に対するリクエストはすべて、Amazon ネットワーク内のプライベートの DynamoDB エンドポイントにルーティングされます。VPC 内の EC2 インスタンスで実行されているアプリケーションを変更する必要はありません。エンドポイント名は変わりませんが、DynamoDB へのルートは Amazon ネットワーク内に完全にとどまります。パブリックインターネットにアクセスすることはありません。

次の図は、VPC 内の EC2 インスタンスが VPC エンドポイントを使用して DynamoDB にアクセスする様子を示しています。



詳細については、「the section called “チュートリアル: DynamoDB 用の VPC エンドポイントを使用する”」を参照してください。

Amazon VPC エンドポイントと DynamoDB の共有

VPC サブネットのゲートウェイエンドポイントから DynamoDB サービスにアクセスできるようにするには、その VPC サブネットに対する所有者アカウントのアクセス許可が必要です。

VPC サブネットのゲートウェイエンドポイントに DynamoDB へのアクセスを許可すると、そのサブネットへのアクセス権を持つすべての AWS アカウントが DynamoDB を使用できます。つまり、VPC サブネット内のすべてのアカウントユーザーは、アクセス権が付与されている対象のすべての DynamoDB テーブルを使用できます。これには、VPC サブネットとは異なるアカウントに関連付けられた DynamoDB テーブルが含まれます。ただし、VPC サブネットの所有者は、独自の裁量により、サブネット内の特定のユーザーに対して、ゲートウェイエンドポイント経由での DynamoDB サービスの使用を制限できます。

チュートリアル: DynamoDB 用の VPC エンドポイントを使用する

このセクションでは、DynamoDB 用の VPC エンドポイントの設定および使用について説明します。

トピック

ステップ 1: Amazon EC2 インスタンスを起動する

このステップでは、デフォルトの Amazon VPC で Amazon EC2 インスタンスを起動します。その後、DynamoDB 用の VPC エンドポイントを作成して使用できます。

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. [Launch Instance] (インスタンスを起動) を選択して、以下を実行します。

   ステップ 1: Amazon マシンイメージ (AMI) を選択する

   - AMI のリストの上部で [Amazon Linux AMI] に移動し、[選択] を選びます。

   ステップ 2: インスタンスタイプを選択する

   - インスタンスタイプのリストの上部で、[t2.micro] を選択します。
   - [Next: Configure Instance Details] (次のステップ: インスタンスの詳細の設定) を選択します。

   ステップ 3: インスタンスの詳細を設定する

   - [ネットワーク] に移動し、デフォルトの VPC を選択します。

[Next: Add Storage] (次の手順: ストレージの追加) をクリックします。

ステップ 4: ストレージを追加する

- [Next: Tag Instance] を選択してこのステップをスキップします。

ステップ 5: インスタンスをタグ付けする

- [Next: Configure Security Group] (次のステップ: セキュリティグループの設定) を選択してこのステップをスキップします。

ステップ 6: セキュリティグループを設定する

- [Select an existing security group] (既存のセキュリティグループの選択) を選択します。
- セキュリティグループのリストで、[default (デフォルト)] を選択します。これは VPC のデフォルトのセキュリティグループです。
- [Next: Review and Launch] (次のステップ: 確認と起動) を選択します。

ステップ 7: インスタンス起動の確認

- [Launch] (起動する) を選択します。

3. [Select an existing key pair or create a new key pair] (既存のキーペアを選択するか、新しいキーペアを作成する) ウィンドウで、次のいずれかを実行します。

   - Amazon EC2 キーペアがない場合は、[Create a new key pair] (新しいキーペアの作成) を選択して指示に従います。プライベートキーファイル (.pem ファイル) をダウンロードするよう求められます。このファイルは、後で Amazon EC2 インスタンスにログインする際に必要になります。
   - 既存の Amazon EC2 キーペアがすでにある場合は、[Select a key pair] (キーペアの選択) を選択して、リストからキーペアを選択します。Amazon EC2 インスタンスにログインするには、既にプライベートキーファイル (.pem ファイル) が利用可能になっている必要があります。

4. キーペアを設定してある場合は、[Launch Instances] (インスタンスの起動) を選択します。

5. Amazon EC2 コンソールのホームページに戻り、起動したインスタンスを選択します。下のペインの [Description] (説明) タブで、インスタンスの [Public DNS] (パブリック DNS) を見つけます。例: `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`。

   このパブリック DNS 名をメモします。パブリック DNS 名は、このチュートリアル (ステップ 2: Amazon EC2 インスタンスを設定する) の次のステップで必要になります。

> Note
>
> Amazon EC2 インスタンスが使用できるようになるまで数分かかります。次のステップに行く前に、[Instance State] (インスタンスの状態) が `running` で、その [Status Checks] (ステータスチェック) がすべてパスしていることを確認します。

ステップ 2: Amazon EC2 インスタンスを設定する

Amazon EC2 インスタンスが使用できるようになったら、そのインスタンスにログインして、最初に使用できるように準備できます。

> Note
>
> 次の手順は、Linux を実行するコンピュータから Amazon EC2 インスタンスに接続していることを想定しています。その他の接続方法については、「Amazon EC2 の Linux インスタンス用ユーザーガイド」の「Linux インスタンスへの接続」を参照してください。

1. Amazon EC2 インスタンスへのインバウンド SSH トラフィックを認証する必要があります。これを行うには、新しい EC2 セキュリティグループを作成し、そのセキュリティグループを EC2 インスタンスに割り当てます。

   a. ナビゲーションペインで、セキュリティグループ] を選択します。

   b. [Create Security Group (セキュリティグループの作成)] を選択します。[セキュリティグループの作成] ウィンドウで、以下を行います。

      - [Security group name] (セキュリティグループ名) — セキュリティグループの名前を入力します。例: `my-ssh-access`
      - [Description] (説明) — セキュリティグループの簡単な説明を入力します。
      - VPC — デフォルトの VPC を選択します。

- [Security group rules] (セキュリティグループのルール) セクションで、[Add Rule] (ルールの追加) を選択して、次の操作を行います
  - [Type] (タイプ) — SSH を選択します。
  - [Source] (ソース) — My IP を選択します。

  すべての設定が正しいことを確認したら、[作成] を選択します。

c. ナビゲーションペインで、[インスタンス] を選択します。

d. ステップ 1: Amazon EC2 インスタンスを起動する で起動した Amazon EC2 インスタンスを選択します。

e. [Actions] (アクション)、[Networking] (ネットワーキング)、[Change Security Groups] (セキュリティグループの変更) の順に選択します。

f. [Change Security Groups] (セキュリティグループの変更) で、この手順で先に作成したセキュリティグループを選択します (例: my-ssh-access)。既存の default のセキュリティグループも選択する必要があります。すべての設定が正しいことを確認したら、[Assign Security Groups] (セキュリティグループの割り当て) を選択します。

2. 次の例のように、ssh コマンドを使用して Amazon EC2 インスタンスにログインします。

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

   プライベートキーファイル (.pem ファイル) とインスタンスのパブリック DNS 名を指定する必要があります。(「ステップ 1: Amazon EC2 インスタンスを起動する」を参照してください)。

   ログイン ID は ec2-user です。パスワードは不要です。

3. 次に示すように、AWS 認証情報を設定します。プロンプトが表示されたら、AWS アクセスキー ID、シークレットキー、デフォルトのリージョン名を入力します。

```
aws configure

AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]:
```

これで、DynamoDB 用の VPC エンドポイントを作成する準備ができました。

ステップ 3: DynamoDB 用の VPC エンドポイントを作成する

このステップでは、DynamoDB 用の VPC エンドポイントを作成し、テストを行い正常に動作することを確認します。

1. 開始する前に、パブリックエンドポイントを使用して DynamoDB と通信できることを確認します。

   ```
   aws dynamodb list-tables
   ```

   出力では、現在所有している DynamoDB テーブルのリストが表示されます。(テーブルがない場合、リストは空になります。)。

2. DynamoDB が、現在の AWS リージョンで VPC エンドポイントを作成するために利用可能なサービスであることを確認します。(コマンドは太字で示され、その後に出力例が続きます。)

   ```
   aws ec2 describe-vpc-endpoint-services

   {
       "ServiceNames": [
           "com.amazonaws.us-east-1.s3",
           "com.amazonaws.us-east-1.dynamodb"
       ]
   }
   ```

   出力例では、DynamoDB が利用可能なサービスの 1 つであるため、VPC エンドポイントの作成を続行できます。

3. VPC 識別子を決定します。

   ```
   aws ec2 describe-vpcs

   {
       "Vpcs": [
           {
               "VpcId": "vpc-0bbc736e",
               "InstanceTenancy": "default",
               "State": "available",
               "DhcpOptionsId": "dopt-8454b7e1",
               "CidrBlock": "172.31.0.0/16",
               "IsDefault": true
           }
   ```

```
    ]
}
```

出力例では、VPC ID は `vpc-0bbc736e` です。

4. VPC エンドポイントを作成します。`--vpc-id` パラメータで、前のステップの VPC ID を指定します。`--route-table-ids` パラメータを使用して、エンドポイントをルートテーブルに関連付けます。

```
aws ec2 create-vpc-endpoint --vpc-id vpc-0bbc736e --service-name com.amazonaws.us-east-1.dynamodb --route-table-ids rtb-11aa22bb

{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [
            "rtb-11aa22bb"
        ],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

5. VPC エンドポイント経由で DynamoDB にアクセスできることを確認します。

```
aws dynamodb list-tables
```

必要に応じて、DynamoDB 用の他の AWS CLI コマンドを試すことができます。詳細については、AWS CLI コマンドリファァレンスを参照してください。

ステップ 4: (オプション) クリーンアップする

このチュートリアルで作成したリソースを削除する場合は、次の手順に従ってください。

DynamoDB 用の VPC エンドポイントを削除するには

1. Amazon EC2 インスタンスにログインします。

2. VPC エンドポイント ID を決定します。

```
aws ec2 describe-vpc-endpoints

{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

   出力例では、VPC エンドポイント ID は vpce-9b15e2f2 です。

3. VPC エンドポイントを削除します。

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2

{
    "Unsuccessful": []
}
```

   空の配列 [] は成功を示します (失敗したリクエストはありませんでした)。

Amazon EC2 インスタンスを終了するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. Amazon EC2 インスタンスを選択します。
4. [Actions]、[Instance State]、[Terminate] の順に選択します。
5. 確認ウィンドウで、[Yes, Terminate] (はい、終了します) を選択します。

AWS PrivateLink for DynamoDB

AWS PrivateLink for DynamoDB を使うと、仮想プライベートクラウド (Amazon VPC) でインターフェイス Amazon VPC エンドポイント (インターフェイスエンドポイント) をプロビジョニングできます。これらのエンドポイントには、オンプレミスにあるアプリケーションから VPN および AWS Direct Connect 経由で、または別の AWS リージョンにあるアプリケーションから Amazon VPC ピアリング経由で直接アクセスできます。AWS PrivateLink とインターフェイスエンドポイントを使用することで、アプリケーションから DynamoDB へのプライベートネットワーク接続を簡素化できます。

VPC 内のアプリケーションは、パブリック IP アドレスがなくても、DynamoDB インターフェイス VPC エンドポイントと通信して、DynamoDB の操作を実行できます。インターフェイスエンドポイントは、Amazon VPC 内のサブネットからプライベート IP アドレスが割り当てられた 1 つ以上の Elastic Network Interface (ENI) で表されます。インターフェイスエンドポイントを介した DynamoDB へのリクエストが Amazon ネットワーク外に出ることはありません。AWS Direct Connect または AWS Virtual Private Network (AWS VPN) を介して、オンプレミスのアプリケーションから Amazon VPC 内のインターフェイスエンドポイントにアクセスすることもできます。Amazon VPC をオンプレミスネットワークに接続する方法の詳細については、「AWS Direct Connect ユーザーガイド」および「AWS Site-to-Site VPN ユーザーガイド」を参照してください。

インターフェイスエンドポイントの一般的な情報については、「AWS PrivateLink ガイド」のインターフェイス Amazon VPC エンドポイント (AWS PrivateLink) に関するセクションを参照してください。

トピック

- Amazon DynamoDB で使用される Amazon VPC エンドポイントのタイプ
- AWS PrivateLink for Amazon DynamoDB を使用する場合の考慮事項
- Amazon VPC エンドポイントの作成
- Amazon DynamoDB インターフェイスエンドポイントへのアクセス
- DynamoDB インターフェイスエンドポイントから DynamoDB テーブルおよびコントロール API オペレーションへのアクセス
- オンプレミスの DNS 設定の更新
- DynamoDB 用 Amazon VPC エンドポイントポリシーの作成

Amazon DynamoDB で使用される Amazon VPC エンドポイントのタイプ

Amazon DynamoDB へのアクセスには、ゲートウェイエンドポイントとインターフェイスエンドポイント (AWS PrivateLink を使用) の 2 つのタイプの Amazon VPC エンドポイントを使用できます。ゲートウェイエンドポイントは、AWS ネットワーク経由で Amazon VPC から DynamoDB にアクセスするために、ルートテーブルで指定するゲートウェイです。インターフェイスエンドポイントは、プライベート IP アドレスを使用して、Amazon VPC 内、オンプレミス、または Amazon VPC ピアリングや AWS Transit Gateway を使用する別の AWS リージョンにある Amazon VPC から DynamoDB にリクエストをルーティングすることにより、ゲートウェイエンドポイントの機能を拡張します。詳細については、「VPC ピア機能とは」および「Transit Gateway と VPC ピアリング」を参照してください。

インターフェイスエンドポイントは、ゲートウェイエンドポイントと互換性があります。Amazon VPC 内に既存のゲートウェイエンドポイントがある場合は、同じ Amazon VPC で両方のタイプのエンドポイントを使用できます。

DynamoDB のゲートウェイエンドポイント	DynamoDB のインターフェイスエンドポイント
いずれの場合も、ネットワークトラフィックは AWS ネットワーク上に残ります。	
Amazon DynamoDB パブリック IP アドレスを使用する	Amazon VPC のプライベート IP アドレスを使用して Amazon DynamoDB にアクセスする
オンプレミスからのアクセスを許可しない	オンプレミスからのアクセスを許可する
別の AWS リージョン からのアクセスを許可しない	Amazon VPC ピアリングまたは AWS Transit Gateway を使用する別の AWS リージョンにある Amazon VPC エンドポイントからのアクセスを許可する
課金されない	請求される

ゲートウェイエンドポイントの詳細については、「AWS PrivateLink ガイド」の「Gateway Amazon VPC endpoints」を参照してください。

AWS PrivateLink for Amazon DynamoDB を使用する場合の考慮事項

Amazon VPC に関する考慮事項が AWS PrivateLink for Amazon DynamoDB に適用されます。詳細については、AWS PrivateLink ガイドの「インターフェイスエンドポイントの考慮事項」と「AWS PrivateLink クォータ」を参照してください。また、以下の制約も適用されます。

AWS PrivateLink for Amazon DynamoDB では、以下はサポートされていません。

- 連邦情報処理規格 (FIPS) エンドポイント
- Transport Layer Security (TLS) 1.1
- プライベートおよびハイブリッドドメインネームシステム (DNS) サービス

AWS PrivateLink は現在、Amazon DynamoDB Streams エンドポイントではサポートされていません。

有効にする AWS PrivateLink エンドポイントごとに、1 秒あたり最大 5 万件のリクエストを送信できます。

> Note
>
> AWS PrivateLink エンドポイントへのネットワーク接続タイムアウトは DynamoDB エラーレスポンスの範囲ではないため、PrivateLink エンドポイントに接続するアプリケーションで適切に処理する必要があります。

Amazon VPC エンドポイントの作成

Amazon VPC インターフェイスエンドポイントを作成するには、「AWS PrivateLink ガイド」の「Create an Amazon VPC endpoint」を参照してください。

Amazon DynamoDB インターフェイスエンドポイントへのアクセス

インターフェイスエンドポイントを作成すると、DynamoDB はエンドポイント固有の 2 つのタイプの DynamoDB DNS 名 (Regional および Zonal) を生成します。

- Regional DNS 名では、一意の Amazon VPC エンドポイント ID、サービス識別子、AWS リージョン、および `vpce.amazonaws.com` が名前に含まれます。例えば、Amazon VPC エンドポイント

ID が *vpce-1a2b3c4d* の場合、生成される DNS 名は *vpce-1a2b3c4d-5e6f*.dynamodb.us-east-1.vpce.amazonaws.com のようになります。

- Zonal DNS 名には、アベイラビリティーゾーンが含まれます (*vpce-1a2b3c4d-5e6f***-us-east-1a**.dynamodb.us-east-1.vpce.amazonaws.com など)。このオプションは、アーキテクチャがアベイラビリティーゾーンを分離する場合に使用できます。例えば、障害を隔離し、リージョン間のデータ転送コストを削減するために使用できます。

エンドポイント固有の DynamoDB DNS 名は、DynamoDB パブリック DNS ドメインから解決できます。

DynamoDB インターフェイスエンドポイントから DynamoDB テーブルおよびコントロール API オペレーションへのアクセス

DynamoDB インターフェイスエンドポイントを介して DynamoDB テーブルおよびコントロール API オペレーションにアクセスするには、AWS CLI または AWS SDK を使用します。

AWS CLI の例

AWS CLI コマンドを使って DynamoDB インターフェイスエンドポイントを介して DynamoDB テーブルまたは DynamoDB コントロール API オペレーションにアクセスするには、--region および --endpoint-url パラメータを使用します。

例: VPC エンドポイントの作成

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name dynamodb-service-name \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

例: VPC エンドポイントの変更

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
```

```
# any additional parameters needed, see Privatelink documentation for more details
```

例: エンドポイント URL を使ったテーブルの一覧表示

次の例では、リージョン us-east-1、VPC エンドポイント ID の DNS 名 vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com をユーザー自身の情報に置き換えます。

```
aws dynamodb --region us-east-1 –endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

AWS SDK の例

AWS SDK を使って DynamoDB インターフェイスエンドポイントを介して DynamoDB テーブルまたは DynamoDB コントロール API オペレーションにアクセスするには、SDK を最新バージョンに更新します。次に、エンドポイント URL を使用して DynamoDB インターフェイスエンドポイントを介してテーブルまたは DynamoDB コントロール API オペレーションにアクセスするように、クライアントを設定します。

SDK for Python (Boto3)

例: エンドポイント URL を使用して DynamoDB テーブルにアクセスする

次の例では、リージョン us-east-1 および VPC エンドポイント ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com をユーザー自身の情報に置き換えます。

```
ddb_client = session.client(
service_name='dynamodb',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

例: エンドポイント URL を使用して DynamoDB テーブルにアクセスする

次の例では、リージョン us-east-1 および VPC エンドポイント ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com をユーザー自身の情報に置き換えます。

```
//client build with endpoint config
final AmazonDynamoDB dynamodb =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

例: エンドポイント URL を使用して S3 バケットにアクセスする

次の例では、リージョン us-east-1 と VPC エンドポイント ID https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com をユーザー自身の情報に置き換えます。

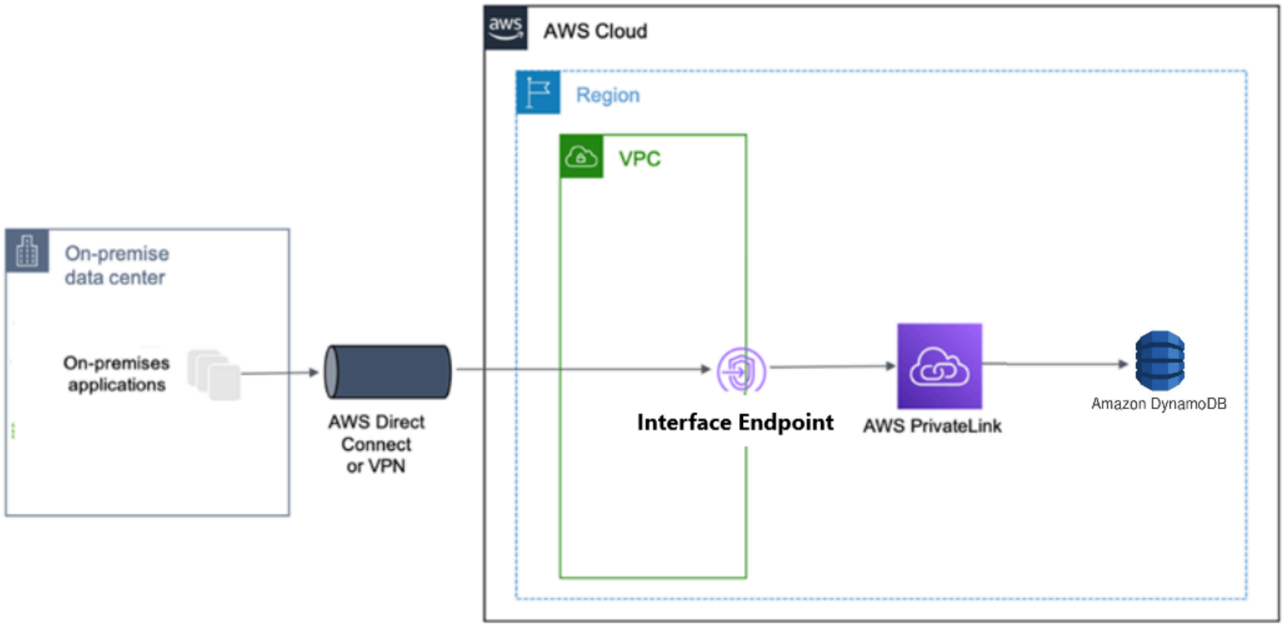
```
Region region = Region.US_EAST_1;
dynamoDbClient = DynamoDbClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

オンプレミスの DNS 設定の更新

エンドポイント固有の DNS 名を使用して DynamoDB のインターフェイスエンドポイントにアクセスする場合、オンプレミス DNS リゾルバーを更新する必要はありません。パブリック DynamoDB DNS ドメインから、インターフェイスエンドポイントのプライベート IP アドレスを使用してエンドポイント固有の DNS 名を解決できます。

Amazon VPC 内のゲートウェイエンドポイントまたはインターネットゲートウェイは使用せず、インターフェイスエンドポイントを使用して DynamoDB にアクセスする

次の図に示すように、Amazon VPC 内のインターフェイスエンドポイントは、Amazon VPC 内のアプリケーションとオンプレミスアプリケーションの両方を Amazon ネットワーク経由で DynamoDB にルーティングできます。



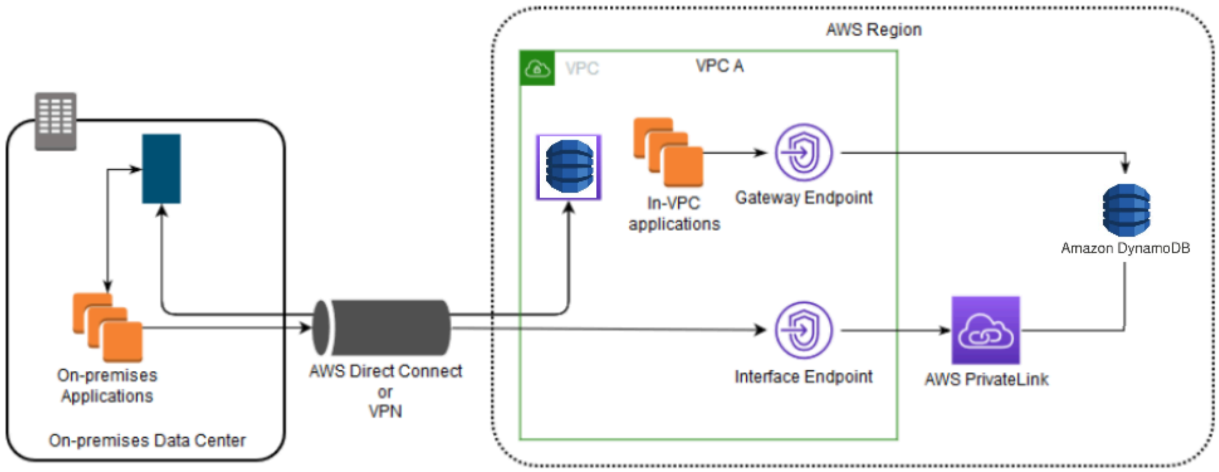
この図表は、以下を示すものです：

- オンプレミスネットワークでは、AWS Direct Connect または AWS VPN を使用して Amazon VPC A に接続します。
- オンプレミスと Amazon VPC A 内のアプリケーションでは、エンドポイント固有の DNS 名を使用して DynamoDB インターフェイスエンドポイントを介して DynamoDB にアクセスします。
- オンプレミスのアプリケーションは、AWS Direct Connect (または AWS VPN) を介して Amazon VPC 内のインターフェイスエンドポイントにデータを送信します。AWS PrivateLink は、AWS ネットワークを経由してデータをインターフェイスエンドポイントから DynamoDB に移動します。
- Amazon VPC 内のアプリケーションも、インターフェイスエンドポイントにトラフィックを送信します。AWS PrivateLink は、AWS ネットワークを経由してデータをインターフェイスエンドポイントから DynamoDB に移動します。

ゲートウェイエンドポイントとインターフェイスエンドポイントを同じ Amazon VPC で併用して DynamoDB にアクセスする

次の図に示すように、インターフェイスエンドポイントを作成し、同じ Amazon VPC 内に既存のゲートウェイエンドポイントも保持できます。このアプローチにより、Amazon VPC 内のアプリケーションが引き続きゲートウェイエンドポイントを介して DynamoDB にアクセスすることが許可されます。これについての請求はありません。インターフェイスエンドポイントは、DynamoDB

にアクセスするオンプレミスのアプリケーションだけが使用することになります。この方法で DynamoDB にアクセスするには、DynamoDB のエンドポイント固有の DNS 名を使用するようにオンプレミスのアプリケーションを更新する必要があります。



この図表は、以下を示すものです：

- オンプレミスのアプリケーションは、エンドポイント固有の DNS 名を使用して、AWS Direct Connect (または AWS VPN) を介して Amazon VPC 内のインターフェイスエンドポイントにデータを送信します。AWS PrivateLink は、AWS ネットワークを経由してデータをインターフェイスエンドポイントから DynamoDB に移動します。
- Amazon VPC 内のアプリケーションは、デフォルトのリージョン DynamoDB 名を使用し、AWS ネットワークを経由して DynamoDB に接続するゲートウェイエンドポイントにデータを送信します。

ゲートウェイエンドポイントの詳細については、Amazon VPC ユーザーガイドの「Gateway Amazon VPC endpoints」を参照してください。

DynamoDB 用 Amazon VPC エンドポイントポリシーの作成

Amazon VPC エンドポイントに DynamoDB へのアクセスをコントロールするエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できる AWS Identity and Access Management (IAM) プリンシパル
- 実行可能なアクション
- アクションを実行できるリソース

トピック

- 例: Amazon VPC エンドポイントから特定のテーブルへのアクセスの制限

例: Amazon VPC エンドポイントから特定のテーブルへのアクセスの制限

特定の DynamoDB テーブルへのアクセスのみを制限するエンドポイントポリシーを作成できます。このタイプのポリシーは、テーブルを使用する他の AWS のサービスが Amazon VPC にある場合に便利です。次のテーブルポリシーは、*DOC-EXAMPLE-TABLE* へのアクセスのみを制限します。このエンドポイントポリシーを使用するには、*DOC-EXAMPLE-TABLE* をテーブルの名前に置き換えます。

```
{
"Version": "2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-table-only",
      "Principal": "*",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:::DOC-EXAMPLE-TABLE",
                   "arn:aws:dynamodb:::DOC-EXAMPLE-TABLE/*"]
    }
  ]
}
```

Amazon DynamoDB での設定と脆弱性の分析

AWS は、ゲストオペレーティングシステム (OS) やデータベースへのパッチ適用、ファイアウォール設定、災害対策などの基本的なセキュリティタスクを処理します。これらの手順は適切な第三者によって確認され、証明されています。詳細については、以下のリソースを参照してください。

- Amazon DynamoDB のコンプライアンス検証
- 責任共有モデル
- アマゾン ウェブ サービス: セキュリティプロセスの概要 (ホワイトペーパー)

以下のセキュリティのベストプラクティスも Amazon DynamoDB での設定と脆弱性の分析に対処します。

- AWS Config ルール を使用した DynamoDB コンプライアンスのモニタリング
- AWS Config を使用した DynamoDB 設定のモニタリング

Amazon DynamoDB のセキュリティベストプラクティス

Amazon DynamoDB には、独自のセキュリティポリシーを策定および実装する際に考慮すべきさまざまなセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションに相当するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、これらは処方箋ではなく、有用な考慮事項と考えてください。

トピック

- DynamoDB での予防的セキュリティのベストプラクティス
- DynamoDB Detective によるセキュリティのベストプラクティス

DynamoDB での予防的セキュリティのベストプラクティス

以下のベストプラクティスは、Amazon DynamoDB のセキュリティインシデントの予防に役立ちます。

保管時の暗号化

> DynamoDB は、AWS Key Management Service (AWS KMS) に保存されているテーブル、インデックス、ストリーミング、Backup に保存されているすべてのユーザーデータを保存時に暗号化します。この機能は、基になるストレージへの不正アクセスからデータを保護することによって、データ保護の追加レイヤーを提供します。
>
> DynamoDB でユーザーデータを暗号化するために AWS 所有のキー (デフォルトの暗号化タイプ)、AWS マネージドキー を使用するか、カスタマーマネージドキーを使用するかを指定できます。詳細については、「保管時の Amazon DynamoDB 暗号化」を参照してください。

IAM ロールを使用した DynamoDB へのアクセス認証

> ユーザー、アプリケーション、およびその他の AWS サービスが DynamoDB にアクセスするには、有効な AWS 認証情報が AWS API リクエストに含まれている必要があります。アプリケー

ションまたは EC2 インスタンスに AWS 認証情報を直接保存しないでください。自動更新されない長期認証情報条件のため、漏洩すると業務に深刻な悪影響が及ぶ可能性があります。IAM ロールでは、AWS サービスおよびリソースにアクセスするために使用できる一時的なアクセスキーを有効にすることができます。

詳細については、「Amazon DynamoDB の Identity and Access Management」を参照してください。

IAM ポリシーを使用した DynamoDB ベース認可

許可を付与する場合、許可を取得するユーザー、取得する許可の対象となる DynamoDB API、およびそれらのリソースに対して許可される特定のアクションを決定します。最小限の特権の実装は、セキュリティリスクはもちろん、エラーや悪意ある行動によってもたらされる可能性のある影響を減らす上での鍵となります。

IAM アイデンティティ (ユーザー、グループ、ロール) にアクセス権限ポリシーをアタッチし、DynamoDB リソースでオペレーションを実行する許可を付与します。

これを行うには、次を使用します。

- AWS 管理 (事前定義) ポリシー
- カスタマー管理ポリシー

詳細に設定されたアクセスコントロールのための IAM ポリシー条件を使用する

DynamoDB でアクセス権限を付与するときは、アクセス権限ポリシーを有効にする方法を決める条件を指定できます。最小限の特権の実装は、セキュリティリスクはもちろん、エラーや悪意ある行動によってもたらされる可能性のある影響を減らす上での鍵となります。

IAM ポリシーを使用して、アクセス許可を付与するときに条件を指定できます。例えば、次の操作を実行できます。

- テーブルまたはセカンダリインデックス内の項目と属性に対する読み込み専用アクセスをユーザーに許可する許可を付与します。
- ユーザーのアイデンティティに基づいて、テーブルの特定の属性への書き込み専用アクセスのアクセス権限をユーザーに付与します。

詳細については、「詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用」を参照してください。

VPC エンドポイントとポリシーを使用した DynamoDB へのアクセス

Virtual Private Cloud (VPC) 内からのみ DynamoDB にアクセスする必要がある場合は、VPC エンドポイントを使用して、必要な VPC からのみのアクセスに制限する必要があります。これによ

り、トラフィックがオープンインターネットを通過してその環境に晒されるのを防ぐことができます。

DynamoDB の VPC エンドポイントを使用すると、以下を使用してアクセスを制御および制限できます。

- VPC エンドポイントポリシー — これらのポリシーは DynamoDB VPC エンドポイントに適用されます。DynamoDB テーブルへの API アクセスを制御および制限が許可されます。
- IAM ポリシー — ユーザー、グループ、またはロールにアタッチされたポリシーで `aws:sourceVpce` 条件を使用すると、DynamoDB テーブルへのすべてのアクセスが指定の VPC エンドポイントを経由するよう強制できます。

詳細については、「Amazon DynamoDB のエンドポイント」を参照してください。

クライアント側の暗号化を考慮する

DynamoDB にテーブルを実装する前に、暗号化戦略を計画することをお勧めします。機密データまたは機密データを DynamoDB に保存する場合は、クライアント側の暗号化をプランに含めることを検討してください。これにより、データをできるだけ送信元に近い状態で暗号化し、ライフサイクル全体にわたってデータを確実に保護できます。伝送中および保管時の機密データを暗号化することで、サードパーティーがお客様のプレーンテキストデータを使用することはできません。

DynamoDB 用の AWS Database Encryption SDK は、DynamoDB に送信する前にテーブルデータを保護するのに役立つソフトウェアライブラリです。DynamoDB テーブル項目を暗号化、署名、検証、復号します。どの属性を暗号化して署名するかを制御できます。

プライマリキーに関する考慮事項

テーブルやグローバルセカンダリインデックスでは、プライマリキーに機密性の高い名前やプレーンテキストデータを使用しないでください。キー名はテーブル定義に表示されます。例えば、プライマリキー名は、DescribeTable を呼び出すアクセス許可を持つすべてのユーザーがアクセスできます。キー値は、AWS CloudTrail やその他のログに表示される場合があります。さらに、DynamoDB はデータの配信とリクエストのルーティングにキー値を使用します。AWS 管理者は、この値を観察してサービスのヘルスを維持する場合があります。

テーブルや GSI のキー値に機密データを使用する必要がある場合は、エンドツーエンドのクライアント暗号化を使用することをお勧めします。これにより、データへのキー値の参照を実行する際に、データが暗号化されないまま DynamoDB 関連ログに表示されることを回避できます。これを実現する 1 つの方法は、AWS Database Encryption SDK for DynamoDB を使用することです

が、これは必須ではありません。独自のソリューションを使用する場合は、十分に安全な暗号化アルゴリズムを常に使用する必要があります。ハッシュなどの非暗号化オプションは、ほとんどの状況で十分に安全とは見なされないため、使用しないでください。

プライマリキーの名前が機密である場合は、代わりに `pk` と `sk` を使用することをお勧めします。これは、パーティションキーの設計を柔軟に保つ一般的なベストプラクティスです。

どれが適切な選択肢であるか不安な場合は、必ずセキュリティエキスパートまたは AWS アカウントチームに相談してください。

DynamoDB Detective によるセキュリティのベストプラクティス

以下は、潜在的なセキュリティ上の弱点とインシデントを検出するために役立つ Amazon DynamoDB でのベストプラクティスです。

AWS CloudTrail を使用した AWS マネージド KMS キーの使用状況モニタリング

保管時の暗号化に AWS マネージドキー を使用している場合、このキーの使用状況が AWS CloudTrail に記録されます。CloudTrail は、アカウントで実行されたアクションをレコードすることで、ユーザーのアクティビティを可視化します。CloudTrail は、リクエストを行ったユーザー、使用されたサービス、実行されたアクション、アクションのパラメータ、AWS のサービスから返されたレスポンス要素など、各アクションに関する重要な情報をレコードします。この情報は、AWS リソースに加えられた変更を追跡し、オペレーション上の問題をトラブルシューティングするサポートになります。CloudTrail を使用すると、社内ポリシーや規制スタンダードへのコンプライアンスが容易になります。

CloudTrail を使用して、キーの使用状況を監査できます。CloudTrail は、アカウントの AWS API コールおよび関連イベントの履歴を含むログファイルを作成します。これらのログファイルには、統合された AWS サービスを通じて行われたものに加えて、AWS Management Console、AWS SDK、およびコマンドラインツールを使用して行われたすべての AWS KMS API リクエストが含まれます。これらのログファイルを使って、KMS が使用された日時、リクエストされたオペレーション、リクエスタの ID、リクエスト元の IP アドレスなどについての情報を取得できます。詳細については、「AWS CloudTrail を使用した AWS KMS API 呼び出しのログ記録」と「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail を使用した DynamoDB オペレーションのモニタリング

CloudTrail は、コントロールプレーンイベントとデータプレーンイベントの両方をモニタリングできます。コントロールプレーンのオペレーションでは、DynamoDB テーブルを作成および管

理できます。また、インデックス、ストリーム、およびテーブルに依存する他のオブジェクトを操作できます。データプレーンオペレーションでは、テーブルのデータで、作成、読み込み、更新、および削除 (CRUD とも呼ばれる) アクションを実行できます。一部のデータプレーンオペレーションでも、セカンダリインデックスからデータを読み込むことができます。CloudTrail でデータプレーンイベントのログを有効にするには、CloudTrail でデータプレーン API アクティビティのログを有効にする必要があります。詳細については、「トレイルのデータイベントのログ記録」を参照してください。

DynamoDB でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS のサービスのイベントと共に CloudTrail イベントにレコードされます。詳細については、「AWS CloudTrail を使用した DynamoDB オペレーションのログ記録」を参照してください。最近のイベントは、AWSアカウントで表示、検索、ダウンロードできます。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail イベント履歴でのイベントの表示」を参照してください。

DynamoDB のイベントなど、AWS アカウントのイベントの継続的なレコードについては、追跡を作成します。証跡により、CloudTrail はログファイルを Amazon Simple Storage Service (Amazon S3) バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡がすべての AWS リージョンに適用されます。証跡では、AWS パーティションのすべてのリージョンからのイベントがログに記録され、指定した S3 バケットにログファイルが配信されます。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づいて対応するため、他の AWS サービスを構成できます。

DynamoDB Streams を使用してデータプレーンオペレーションをモニタリングする

DynamoDB は AWS Lambda と統合されているため、トリガー (DynamoDB Streams 内のイベントに自動的に応答するコードの一部) を作成できます。トリガーを使用すると、DynamoDB テーブル内のデータ変更に対応するアプリケーションを構築できます。

テーブルで DynamoDB Streams を有効にした場合、書き込む Lambda 関数にストリームの Amazon リソースネーム (ARN) を関連付けることができます。テーブルの項目が変更されるとすぐに、新しいレコードがテーブルのストリーミングに表示されます。AWS Lambda はストリーミングをポーリングし、新しいストリーミングレコードを検出すると、 Lambda 関数を同期的に呼び出します。Lambda 関数は、通知の送信やワークフローの開始など、指定したアクションを実行できます。

例については、「チュートリアル: Amazon DynamoDB Streams で AWS Lambda を使用する」を参照してください。この例では、DynamoDB イベント入力を受け取り、それに含まれるメッセージを処理して、受信イベントデータの一部を Amazon CloudWatch Logs に書き込みます。

AWS Config を使用した での DynamoDB 設定のモニタリング

AWS Config を使用すると、AWS リソースの設定変更を継続的にモニタリングおよびレコードできます。また、AWS Config を使用して AWS リソースのインベントリも作成できます。以前の状態からの変更が検出されると、Amazon Simple Notification Service (Amazon SNS) 通知が配信されるので、内容を確認してアクションを実行できます。「コンソールを使用して AWS Config を設定する」のガイダンスに従い、DynamoDB リソースタイプが含まれていることを確認します。

設定変更と通知を Amazon SNS トピックにストリーミングするように AWS Config を設定できます。例えば、リソースの更新時に E メールアドレスに通知が送信されれば、変更を確認できます。AWS Config でカスタムルールまたはマネージドルールを適用してリソースを評価したときにも通知を受け取ることができます。

例については、AWS Config デベロッパーガイドの「AWS Config が Amazon SNS トピックに送信する通知」を参照してください。

AWS Config ルール を使用した での DynamoDB コンプライアンスのモニタリング

AWS Config は、リソースで発生する設定変更を継続的に追跡し、これらの変更がルールの条件に違反していないかどうかをチェックします。リソースがルールに違反している場合、AWS Config はリソースとルールに非準拠を示す [noncompliant] のフラグを付けます。

AWS Config を使用してリソースの設定を評価することで、自社プラクティス、業界ガイドライン、および規制に対するリソース設定の準拠状態を確認できます。AWS Config は、AWSマネージドルールを提供しますが、これは事前定義されているカスタマイズ可能なルールで、AWS Config はこのルールを使用して AWS リソースが一般的なベストプラクティスに準拠しているか評価します。

識別とオートメーションのために DynamoDB リソースにタグを付ける

AWS のリソースにメタデータをタグ付け形式で割り当てることができます。各タグは、カスタマー定義のキーと値 (オプション) で構成されるシンプルなラベルです。タグを使用すると、リソースの管理、検索、フィルターが容易になります。

タグ付けを行うと、グループ化されたコントロールを実装できます。タグには固有のタイプはありませんが、 用途、所有者、環境などの基準でリソースを分類できます。次に例をいくつか示します。

- セキュリティ — 暗号化などの要件を決定するために使用されます。
- 機密性 — リソースがサポートするデータ機密性レベルの識別子。

- 環境 — 開発、テスト、本番稼働用インフラストラクチャを区別するために使用されます。

  詳細については、「AWS タグ付け戦略」および「DynamoDB のタグ付け」を参照してください。

AWS Security Hub を使用して、セキュリティのベストプラクティスに関連する Amazon DynamoDB の使用状況をモニタリングします。

  Security Hub は、セキュリティコントロールを使用してリソース設定とセキュリティ標準を評価し、お客様がさまざまなコンプライアンスフレームワークに準拠できるようサポートします。

  Security Hub を使用して DynamoDB リソースを評価する方法の詳細については、「AWS Security Hub ユーザーガイド」の「Amazon DynamoDB コントロール」を参照してください。

DynamoDB でのモニタリングとログ記録

モニタリングは、DynamoDB および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害を簡単にデバッグできるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。

トピック

モニタリング計画

DynamoDB のモニタリングを開始する前に、以下の質問に対する回答を反映したモニタリング計画を作成します。

- どのような目的でモニタリングしますか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

パフォーマンスのベースライン

さまざまなタイミングと負荷条件でパフォーマンスを測定することにより、使用環境における通常の DynamoDB パフォーマンスのベースラインを確定します。DynamoDB をモニタリングするには、モニタリングの履歴データを保存することを検討します。保存データを最新のパフォーマンスデータと比較するベースラインとして使用し、通常のパフォーマンスのパターンやパフォーマンスの異常を検

出して、問題への対応を検討することができます。ベースラインを確立するには、少なくとも、次の項目をモニタリングする必要があります。

- 指定された期間に消費された読み込みまたは書き込み容量ユニットの数。プロビジョニングされたスループットの使用量を追跡できます。
- テーブルにプロビジョニングされた書き込みまたは読み込み容量を指定期間中に超えたリクエストで、テーブルにプロビジョニングされたスループットクォータをどのリクエストが超えたかを判断できます。
- エラー。リクエストがエラーになったかどうかを判断できます。

統合サービス

DynamoDB は、ユーザーに代わってテーブルを自動的にモニタリングし、Amazon CloudWatch を介してメトリクスを報告します。さらに、以下の AWS のサービスとの統合により、DynamoDB リソースのモニタリングとトラブルシューティングも促進します。

- AWS CloudTrail は、AWS アカウント により、またはそのアカウントに代わって行われた API コールおよび関連イベントを取得し、指定した Amazon S3 バケットにログファイルを配信します。詳細については、「AWS CloudTrail を使用して DynamoDB オペレーションをログに記録する」を参照してください。
- Contributor Insights は、テーブルやインデックス内で最も頻繁にアクセスおよびスロットリングされるキーをすばやく特定するための診断ツールです。詳細については、「CloudWatch Contributor Insights for DynamoDB を使用してデータアクセスを分析する」を参照してください。

自動モニタリングツール

AWS は、DynamoDB のモニタリングに使用できるさまざまなツールを提供しています。モニタリングタスクをできるだけ自動化することをお勧めします。以下の自動化されたモニタリングツールを使用して、DynamoDB を監視し、問題が発生したときにレポートできます。

- AWS CloudTrail アラーム – 指定した期間にわたって単一のメトリクスを監視し、複数の期間にわたって特定のしきい値と比較したメトリクス値に基づいて、1 つ以上のアクションを実行します。

  アクションは、Amazon Simple Notification Service (Amazon SNS) トピックまたは Amazon EC2 Auto Scaling ポリシーに通知として送信されます。AWS CloudTrail アラームが特定の状態にあるというだけでは、アクションは呼び出されません。アラームの状態が変わり、指定した数の期間に

わたって変化が持続する必要があります。詳細については、「Amazon CloudWatch によるメトリクスのモニタリング」を参照してください。

- AWS CloudTrail ログのモニタリング - アカウント間でログファイルを共有し、AWS CloudTrail ログファイルを AWS CloudTrail Logs に送信してリアルタイムでモニタリングします。また、ログ処理アプリケーションを Java で作成し、AWS CloudTrail による配信後にログファイルが変更されていないことを検証します。詳細については、「AWS CloudTrail ユーザーガイド」の「Amazon CloudWatch Logs とは」を参照してください。

Amazon CloudWatch によるメトリクスのモニタリング

CloudWatch を使用して DynamoDB をモニタリングすることで、DynamoDB から raw データを収集し、リアルタイムに近い読み込み可能なメトリクスに加工することができます。これらの統計は一定期間保持されるため、履歴情報にアクセスしてウェブアプリケーションやサービスの動作をより的確に把握できます。デフォルトでは、DynamoDB メトリクスデータは CloudWatch に自動的に送信されます。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch とは」および「メトリクスの保持」を参照してください。

トピック

DynamoDB メトリクスの使用方法

DynamoDB によってレポートされるメトリクスが提供する情報は、さまざまな方法で分析できます。以下のリストは、メトリクスの一般的な利用方法をいくつか示しています。ここで紹介するのは開始するための提案事項です。すべてを網羅しているわけではありません。

DynamoDB メトリクスの使用方法

目的	関連するメトリクス
テーブルの TTL 削除率は、どのようにしてモニタリングできますか?	指定した期間にわたって TimeToLiveDeletedItemCount をモニタリングし、テーブルの TTL 削除率を追跡できます。TimeToLiveDeletedItemCount メトリクスを使用するサーバーレスアプリケーションの例については、「Automatically archive items to S3 using DynamoDB time to live (TTL) with AWS Lambda and Amazon Data Firehose」を参照してください。
プロビジョニングされたスループットの消費量は、どのようにして確認できますか?	指定した期間 ConsumedReadCapacityUnits または ConsumedWriteCapacityUnits をモニタリングし、プロビジョニング済みスループットの使用量を追跡できます。
どのリクエストがテーブルのプロビジョニングされたスループットクォータを超えているかは、どのようにして確認できますか?	リクエスト内で任意のイベントがプロビジョニングされたスループットクォータを超過した場合、ThrottledRequests は 1 ずつ増加します。次に、どのイベントがリクエストをスロットリングしているかについてのインサイトを取得するには、ThrottledRequests とテーブルおよびそのインデックスの ReadThrottleEvents メトリクスと WriteThrottleEvents メトリクスを比較します。
どうすればシステムエラーの発生の有無を判断できますか。	SystemErrors をモニタリングして HTTP 500 (サーバーエラー) コードを発生させたリクエストがあるかどうかを判断できます。通常、このメトリクスはゼロであるべきです。そうでない場合は、調査することをお勧めします。
テーブルオペレーションのレイテンシー値は、どのようにしてモニタリングできますか?	SuccessfulRequestLatency をモニタリングして平均レイテンシーを追跡できます。時折レイテンシーが急上昇しても問題ありません。ただし、平均レイテンシーが高い場合は、解決すべき根本的な問題が存在する可能性があります。詳細については、「Amazon DynamoDB でのレイテンシーの問題のトラブルシューティング」を参照してください。

CloudWatch コンソールでのメトリクスの表示

メトリクスは、まずサービスの名前空間ごとにグループ化されます。次に各名前空間内でさまざまなディメンションの組み合わせごとにグループ化されます。

CloudWatch コンソールでメトリクスを表示する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[メトリクス]、[すべてのメトリクス] の順に選択します。
3. DynamoDB 名前空間を選択します。また、Usage (使用状況) 名前空間を選択して、DynamoDB の使用状況メトリクスを表示できます。使用状況メトリクスの詳細については、「AWS 使用状況メトリクス」を参照してください。
4. [参照] タブには、名前空間内のすべてのメトリクスが表示されます。
5. (オプション) メトリクスグラフを CloudWatch ダッシュボードに追加するには、[アクション]、[ダッシュボードに追加] の順に選択します。

AWS CLI でのメトリクスの表示

AWS CLI を使用してメトリクス情報を取得するには、CloudWatch の `list-metrics` コマンドを使用します。次の例では、`AWS/DynamoDB` 名前空間にすべてのメトリクスがリストされています。

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

メトリクスの統計情報を取得するには、`get-metric-statistics` コマンドを使用します。次のコマンドは、特定の 24 時間にわたるテーブル `ProductCatalog` の `ConsumedReadCapacityUnits` 統計情報を、5 分間隔で取得します。

```
aws cloudwatch get-metric-statistics —namespace AWS/DynamoDB \
     —metric-name ConsumedReadCapacityUnits \
     —start-time 2023-11-01T00:00:00Z \
     —end-time 2023-11-02T00:00:00Z \
     —period 360 \
     —statistics Average \
     —dimensions Name=TableName,Value=ProductCatalog
```

出力例は次のとおりです。

```
{
    "Datapoints": [
        {
            "Timestamp": "2023-11-01T 09:18:00+00:00",
            "Average": 20,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 04:36:00+00:00",
            "Average": 22.5,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 15:12:00+00:00",
            "Average": 20,
            "Unit": "Count"
        }, ...
        {
            "Timestamp": "2023-11-01T 17:30:00+00:00",
            "Average": 25,
            "Unit": "Count"
        }
    ],
    "Label": " ConsumedReadCapacityUnits "
}
```

DynamoDB のメトリクスとディメンション

DynamoDB を操作すると、メトリクスとディメンションが CloudWatch に送信されます。

メトリクスおよびディメンションの表示

CloudWatch は、以下の DynamoDB のメトリクスを表示します。

DynamoDB のメトリクス

> **Note**
>
> Amazon CloudWatch は 1 分間隔でこれらのメトリクスを集計します。
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`

- `ConsumedWriteCapacityUnits`
- `ReadThrottleEvents`
- `ReturnedBytes`
- `ReturnedItemCount`
- `ReturnedRecordsCount`
- `SuccessfulRequestLatency`
- `SystemErrors`
- `TimeToLiveDeletedItemCount`
- `ThrottledRequests`
- `TransactionConflict`
- `UserErrors`
- `WriteThrottleEvents`

その他のすべての DynamoDB メトリクスでは、集計の間隔は 5 分です。

Average や Sum など、すべての統計が必ずしも常にすべてのメトリクスに適用可能であるとは限りません。ただし、これらの値はすべて Amazon DynamoDB コンソール経由で利用できます。また、すべてのメトリクスで CloudWatch コンソール、AWS CLI、AWS SDK を使用することによっても利用できます。

次のリストは、各メトリクスに適用可能な有効な統計のセットを示します。

利用可能なメトリクスのリスト

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- 条件チェックが失敗したリクエスト
- ConsumedChangeDataCaptureUnits

- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount
- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization
- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress
- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteThrottleEvents

AccountMaxReads

アカウントで使用できる読み込み容量ユニットの最大数。この制限は、オンデマンドテーブルやグローバルセカンダリインデックスには適用されません。

単位: `Count`

有効な統計:

- `Maximum` — アカウントで使用できる読み込み容量ユニットの最大数。

AccountMaxTableLevelReads

アカウントのテーブルまたはグローバルセカンダリインデックスで使用できる読み込み容量ユニットの最大数。オンデマンドテーブルの場合、この制限は、テーブルやグローバルセカンダリインデックスで使用できる読み込みリクエストユニットの上限数を示します。

単位: `Count`

有効な統計:

- `Maximum` — アカウントのテーブルまたはグローバルセカンダリインデックスで使用できる読み込み容量ユニットの最大数。

AccountMaxTableLevelWrites

アカウントのテーブルまたはグローバルセカンダリインデックスで使用できる書き込み容量ユニットの最大数。オンデマンドテーブルの場合、この制限は、テーブルやグローバルセカンダリインデックスで使用できる書き込みリクエストユニットの上限数を示します。

単位: `Count`

有効な統計:

- `Maximum` — アカウントのテーブルまたはグローバルセカンダリインデックスで使用できる書き込み容量ユニットの最大数。

AccountMaxWrites

アカウントで使用できる書き込み容量ユニットの最大数。この制限は、オンデマンドテーブルやグローバルセカンダリインデックスには適用されません。

単位: `Count`

有効な統計:

- `Maximum` - アカウントで使用できる書き込み容量ユニットの最大数。

AccountProvisionedReadCapacityUtilization

アカウントで利用されるプロビジョニング済み読み込み容量ユニットの割合。

単位: `Percent`

有効な統計:

- `Maximum` — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの最大割合。
- `Minimum` — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの最小割合。
- `Average` — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの平均割合。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み読み込み容量ユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

AccountProvisionedWriteCapacityUtilization

アカウントで利用されるプロビジョニング済み書き込み容量ユニットの割合。

単位: `Percent`

有効な統計:

- `Maximum` - アカウントで利用されるプロビジョニング済み書き込み容量ユニットの最大割合。
- `Minimum` — アカウントで使用されるプロビジョニング済み読み込み容量ユニットの最小割合。
- `Average` — アカウントで使用されるプロビジョニング済み書き込み容量ユニットの平均割合。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み書き込み容量ユニットをすばやく調整すると、この統計には実際の平均値が反映されないことがあります。

AgeOfOldestUnreplicatedRecord

Kinesis データストリームにレプリケートされていないレコードからの経過時間が DynamoDB テーブルに最初に出現してからの経過時間。

単位: `Milliseconds`

ディメンション: `TableName, DelegatedOperation`

有効な統計:

- Maximum.
- Minimum.
- Average.

条件チェックが失敗したリクエスト

条件付き書き込みの実行に失敗した回数。PutItem、UpdateItem、および DeleteItem オペレーションを使用すると、オペレーションを続行する前に true と評価される必要がある論理条件を指定できます。この条件が false に評価される場合は、ConditionalCheckFailedRequests は 1 つ増加します。ConditionalCheckFailedRequests も、論理条件が提供され、その条件が false に評価される PartiQL Update および Delete ステートメントで 1 つ増加します。

> Note
>
> 条件付き書き込みに失敗すると HTTP 400 エラー (Bad Request) が発生します。これらのイベントは ConditionalCheckFailedRequests メトリクスに反映されますが、UserErrors メトリクスには反映されません。

単位: Count

ディメンション: TableName

有効な統計:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ConsumedChangeDataCaptureUnits

消費された変更データキャプチャユニットの数。

単位: Count

ディメンション: `TableName, DelegatedOperation`

有効な統計:

- `Minimum`
- `Maximum`
- `Average`

ConsumedReadCapacityUnits

プロビジョンドキャパシティとオンデマンドキャパシティの両方について、指定された期間内に消費された読み取りキャパシティユニットの数。これにより、使用されたスループットの量を追跡できます。テーブルとそのすべてのグローバルセカンダリインデックス、または特定のグローバルセカンダリインデックスの消費された読み込み容量の合計を取得できます。詳細については、「読み込み/書き込み容量モード」を参照してください。

`TableName` ディメンションは、グローバルセカンダリインデックスではなくテーブルの `ConsumedReadCapacityUnits` を返します。グローバルセカンダリインデックスの `ConsumedReadCapacityUnits` を表示するには、`TableName` と `GlobalSecondaryIndexName` の両方を指定する必要があります。

> **Note**
>
> Amazon DynamoDB では、消費されたキャパシティのメトリクスが平均値として 1 分間隔で CloudWatch にレポートされます。つまり、キャパシティ消費量の急増が 1 秒間だけであれば、CloudWatch グラフには正確に反映されず、該当する 1 分間の消費率は低く見える可能性があります。
> Sum 統計を使用して、消費されたスループットを計算します。例えば、1 分間の Sum 値を取得し、1 分間の秒数 (60) で除算して 1 秒あたりの平均 `ConsumedReadCapacityUnits` を計算します。計算された値と、DynamoDB が提供するプロビジョニング済みスループット値を比較できます。

単位: `Count`

ディメンション: `TableName, GlobalSecondaryIndexName`

有効な統計:

- Minimum — テーブルまたはインデックスへの個々のリクエストによって消費される読み込み容量ユニットの最小数。
- Maximum — テーブルまたはインデックスへの個々のリクエストによって消費される読み込み容量ユニットの最大数。
- Average — 消費されたリクエストごとの平均読み込み容量。

> Note
>
> Average 値は、サンプル値がゼロになる非活動期間によって影響を受けます。

- Sum — 消費された読み込み容量ユニットの合計。これは、ConsumedReadCapacityUnits メトリクスの最も有用な統計です。
- SampleCount — DynamoDB への読み込みリクエストの数。読み込みキャパシティーの消費がなかった場合は 0 を返します。

> Note
>
> SampleCount 値は、サンプル値がゼロになる非活動期間によって影響を受けます。

ConsumedWriteCapacityUnits

プロビジョンドキャパシティとオンデマンドキャパシティの両方について、指定された期間内に消費された書き込みキャパシティユニットの数。これにより、使用されたスループットの量を追跡できます。テーブルとそのすべてのグローバルセカンダリインデックス、または特定のグローバルセカンダリインデックスの消費された書き込み容量の合計を取得できます。詳細については、「読み込み/書き込み容量モード」を参照してください。

TableName ディメンションは、グローバルセカンダリインデックスではなくテーブルの ConsumedWriteCapacityUnits を返します。グローバルセカンダリインデックスの ConsumedWriteCapacityUnits を表示するには、TableName と GlobalSecondaryIndexName の両方を指定する必要があります。

> Note
>
> Sum 統計を使用して、消費されたスループットを計算します。例えば、Sum 値を 1 分にわたって取得し、1 分間の秒数 (60) で除算して 1 秒あたりの平均 ConsumedWriteCapacityUnits を計算します (この平均では 1 分の間に書き込みアク

ティビティで発生した大きくて短いスパイクは強調されないことを認識します)。計算された値と、DynamoDB が提供するプロビジョニング済みスループット値を比較できます。

単位: `Count`

ディメンション: `TableName, GlobalSecondaryIndexName`

有効な統計:

- `Minimum` — テーブルまたはインデックスへの個々のリクエストによって消費される書き込み容量ユニットの最小数。
- `Maximum` — テーブルまたはインデックスへの個々のリクエストによって消費される書き込み容量ユニットの最大数。
- `Average` — 消費されたリクエストごとの平均書き込み容量。

> Note
>
> `Average` 値は、サンプル値がゼロになる非活動期間によって影響を受けます。

- `Sum` — 消費された書き込み容量ユニットの合計。これは、`ConsumedWriteCapacityUnits` メトリクスの最も有用な統計です。
- `SampleCount` — DynamoDB への書き込みリクエストの数 (書き込み容量が消費されなかった場合も含まれます)。

> Note
>
> `SampleCount` - 値は、サンプル値がゼロになる非活動期間によって影響を受けます。

FailedToReplicateRecordCount

DynamoDB が Kinesis Data Streams にレプリケートできなかったレコードの数。

単位: `Count`

ディメンション: `TableName`、`DelegatedOperation`

有効な統計:

- `Sum`

MaxProvisionedTableReadCapacityUtilization

プロビジョニング済み読み込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み読み込みテーブルまたはグローバルセカンダリインデックスが使用している割合。

単位: `Percent`

有効な統計:

- `Maximum` – プロビジョニング済み読み込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み読み込みテーブルまたはグローバルセカンダリインデックスが使用している割合の最大値。
- `Minimum` – プロビジョニング済み読み込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み読み込みテーブルまたはグローバルセカンダリインデックスが使用している割合の最小値。
- `Average` — プロビジョニング済み書き込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み書き込みテーブルまたはグローバルセカンダリインデックスが使用している割合の平均。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み読み込み容量ユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

MaxProvisionedTableWriteCapacityUtilization

プロビジョニング済み書き込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み書き込みテーブルまたはグローバルセカンダリインデックスが使用している割合。

単位: `Percent`

有効な統計:

- `Maximum` — プロビジョニング済み書き込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み書き込みテーブルまたはグローバルセカンダリインデックスが使用している最大割合。
- `Minimum` — プロビジョニング済み書き込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み書き込みテーブルまたはグローバルセカンダリインデックスが使用している最小割合。

- Average — プロビジョニング済み書き込み容量ユニットのうち、アカウントの最も高いプロビジョニング済み書き込みテーブルまたはグローバルセカンダリインデックスが使用している平均割合。メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み書き込み容量ユニットをすばやく調整すると、この統計には実際の平均値が反映されないことがあります。

OnDemandMaxReadRequestUnits

テーブルまたはグローバルセカンダリインデックスに指定されたオンデマンド読み込みリクエストユニットの数。

テーブルの OnDemandMaxReadRequestUnits を表示するには、TableName を指定する必要があります。グローバルセカンダリインデックスの OnDemandMaxReadRequestUnits を表示するには、TableName と GlobalSecondaryIndexName の両方を指定する必要があります。

単位: カウント

ディメンション: TableName、GlobalSecondaryIndexName

有効な統計:

- Minimum – オンデマンド読み込みリクエストユニットの最小設定。UpdateTable を使用して読み込みリクエストユニットを増やす場合、このメトリクスは、この期間中のオンデマンド ReadRequestUnits の最小値を示します。
- Maximum – オンデマンド読み込みリクエストユニットの最大設定。UpdateTable を使用して読み込みリクエストユニットを減らす場合、このメトリクスは、この期間中のオンデマンド ReadRequestUnits の最大値を示します。
- Average – オンデマンド読み込みリクエストユニットの平均。OnDemandMaxReadRequestUnits メトリクスは 5 分間隔で発行されます。したがって、オンデマンド読み込みリクエストユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

OnDemandMaxWriteRequestUnits

テーブルまたはグローバルセカンダリインデックスに指定されたオンデマンド書き込みリクエストユニットの数。

テーブルの OnDemandMaxWriteRequestUnits を表示するには、TableName を指定する必要があります。グローバルセカンダリインデックスの OnDemandMaxWriteRequestUnits を表示するには、TableName と GlobalSecondaryIndexName の両方を指定する必要があります。

単位: Count

ディメンション: TableName、GlobalSecondaryIndexName

有効な統計:

- Minimum – オンデマンド書き込みリクエストユニットの最小設定。UpdateTable を使用して書き込みリクエストユニットを増やす場合、このメトリクスは、この期間中のオンデマンド WriteRequestUnits の最小値を示します。
- Maximum – オンデマンド書き込みリクエストユニットの最大設定。UpdateTable を使用して書き込みリクエストユニットを減らす場合、このメトリクスは、この期間中のオンデマンド WriteRequestUnits の最大値を示します。
- Average – オンデマンド書き込みリクエストユニットの平均。OnDemandMaxWriteRequestUnits メトリクスは 5 分間隔で発行されます。したがって、オンデマンド書き込みリクエストユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

OnlineIndexConsumedWriteCapacity

新しいグローバルセカンダリインデックスをテーブルに追加するときに消費される書き込み容量ユニットの数。インデックスの書き込み容量が低すぎると、バックフィルフェーズ中の書き込みアクティビティのスロットリングが発生することがあります。その結果、インデックスの作成に要する時間が長くなります。インデックスの作成中にこの統計を監視して、インデックスの書き込み容量のプロビジョニングが必要な量を下回っていないかどうかを判断する必要があります。

インデックスの構築中でも、UpdateTable オペレーションを使用してインデックスの書き込み容量を調整できます。

インデックスの ConsumedWriteCapacityUnits メトリクスには、インデックスの作成中に消費された書き込みスループットは含まれません。

> **Note**
>
> 新しいグローバルセカンダリインデックスのバックフィルフェーズがすぐに (数分以内に) 完了した場合、このメトリックは生成されない可能性があります。これは、ベーステーブルにインデックスにバックフィルする項目がほとんどないか、またはまったくない場合に発生する可能性があります。

単位: Count

ディメンション: TableName, GlobalSecondaryIndexName

有効な統計:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

OnlineIndexPercentageProgress

新しいグローバルセカンダリインデックスがテーブルに追加されるときの完了率。DynamoDB は、まず新しいインデックスにリソースを割り当てて、次にテーブルの属性でインデックスを埋めていく必要があります。大きなテーブルの場合、この処理には長時間かかる場合があります。DynamoDB がインデックスを構築する際の相対的な進行状況を表示するには、この統計をモニタリングする必要があります。

単位: Count

ディメンション: TableName, GlobalSecondaryIndexName

有効な統計:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

OnlineIndexThrottleEvents

新しいグローバルセカンダリインデックスをテーブルに追加するときに発生する書き込みスロットリングイベントの数。これらのイベントは、受信書き込みアクティビティがインデックスのプロビジョニング済み書き込みスループットを超えていることが原因でインデックスの作成の完了に時間がかかることを示します。

インデックスの構築中でも、UpdateTable オペレーションを使用してインデックスの書き込み容量を調整できます。

インデックスの WriteThrottleEvents メトリクスには、インデックスの作成中に発生したスロットリングイベントは含まれません。

単位: Count

ディメンション: TableName, GlobalSecondaryIndexName

有効な統計:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

PendingReplicationCount

グローバルテーブルバージョン 2017.11.29 (レガシー) のメトリクス (グローバルテーブルのみ)。1 つのレプリカテーブルに書き込まれていても、グローバルテーブル内の別のレプリカにはまだ書き込まれていない項目の更新の数。

単位: Count

ディメンション: TableName, ReceivingRegion

有効な統計:

- Average
- Sample Count
- Sum

ProvisionedReadCapacityUnits

テーブルまたはグローバルセカンダリインデックスのプロビジョニング済み読み込み容量ユニットの数。TableName ディメンションは、グローバルセカンダリインデックスでは

なくテーブルの ProvisionedReadCapacityUnits を返します。グローバルセカンダリインデックスの ProvisionedReadCapacityUnits を表示するには、TableName と GlobalSecondaryIndexName の両方を指定する必要があります。

単位: Count

ディメンション: TableName, GlobalSecondaryIndexName

有効な統計:

- Minimum — プロビジョニング済み読み込み容量の最小設定。UpdateTable を使用して読み込み容量を増やす場合、このメトリクスは、この期間中のプロビジョニング済み ReadCapacityUnits の最小値を示します。
- Maximum — プロビジョニング済み読み込み容量の最大設定。UpdateTable を使用して読み込み容量を減らす場合、このメトリクスは、この期間中のプロビジョニング済み ReadCapacityUnits の最大値を示します。
- Average — プロビジョニング済み読み込み容量の平均。ProvisionedReadCapacityUnits メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み読み込み容量ユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

ProvisionedWriteCapacityUnits

テーブルまたはグローバルセカンダリインデックスのプロビジョニング済み書き込み容量ユニットの数。

TableName ディメンションは、グローバルセカンダリインデックスではなくテーブルの ProvisionedWriteCapacityUnits を返します。グローバルセカンダリインデックスの ProvisionedWriteCapacityUnits を表示するには、TableName と GlobalSecondaryIndexName の両方を指定する必要があります。

単位: Count

ディメンション: TableName, GlobalSecondaryIndexName

有効な統計:

- Minimum — プロビジョニング済み書き込み容量の最小設定。UpdateTable を使用して書き込み容量を増やす場合、このメトリクスは、この期間中のプロビジョニング済み WriteCapacityUnits の最小値を示します。

- Maximum — プロビジョニング済み書き込み容量の最大設定。UpdateTable を使用して書き込み容量を減らす場合、このメトリクスは、この期間中のプロビジョニング済み WriteCapacityUnits の最大値を示します。
- Average — プロビジョニング済み書き込み容量の平均。ProvisionedWriteCapacityUnits メトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み書き込み容量ユニットをすばやく調整すると、この統計には実際の平均値が反映されないことがあります。

ReadThrottleEvents

テーブルまたはグローバルセカンダリインデックス用にプロビジョニングされた読み込み容量ユニットを超える DynamoDB へのリクエスト。

1 つのリクエストで複数のイベントが発生する可能性があります。例えば、10 の項目を読み込む BatchGetItem は、10 個の GetItem イベントとして処理されます。各イベントでは、そのイベントがスロットリングされている場合、ReadThrottleEvents は 1 つ増加します。10 個すべての GetItem がスロットリングされない限り、BatchGetItem 全体の ThrottledRequests メトリクスは増加しません。

TableName ディメンションは、グローバルセカンダリインデックスではなくテーブルの ReadThrottleEvents を返します。グローバルセカンダリインデックスの ReadThrottleEvents を表示するには、TableName と GlobalSecondaryIndexName の両方を指定する必要があります。

単位: Count

ディメンション: TableName, GlobalSecondaryIndexName

有効な統計:

- SampleCount
- Sum

ReplicationLatency

(このメトリクスは DynamoDB グローバルテーブル用です)。更新された項目が 1 つのレプリカテーブルの DynamoDB Streams に表示され、その項目がグローバルテーブルの別のレプリカに表示されるまでの経過時間。

単位: Milliseconds

ディメンション: TableName, ReceivingRegion

有効な統計:

- Average
- Minimum
- Maximum

ReturnedBytes

指定した期間中に GetRecords オペレーション (Amazon DynamoDB Streams) によって返されるバイト数。

単位: Bytes

ディメンション: Operation, StreamLabel, TableName

有効な統計:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ReturnedItemCount

指定した期間中に Query、Scan、または ExecuteStatement (選択) オペレーションによって返される項目の数。

返された項目の数は、評価された項目の数と必ずしも同じではありません。例えば、100 の項目があるテーブルまたはインデックス上の Scan をリクエストし、15 の項目だけが返されるように結果を絞り込む FilterExpression を指定した場合を考えてみます。この場合、Scan からのレスポンスには、100 の ScanCount と 15 の返された項目の Count が含まれます。

単位: Count

ディメンション: TableName, Operation

有効な統計:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

ReturnedRecordsCount

指定した期間中に GetRecords オペレーション (Amazon DynamoDB Streams) によって返されるストリームレコードの数。

単位: Count

ディメンション: Operation, StreamLabel, TableName

有効な統計:

- Minimum
- Maximum
- Average
- SampleCount
- Sum

SuccessfulRequestLatency

指定した期間中に成功した DynamoDB または Amazon DynamoDB Streams へのリクエストのレイテンシー。SuccessfulRequestLatency は、次の 2 種類の異なる情報を提供できます。

- リクエストが成功するまでの経過時間 (Minimum、Maximum、Sum、または Average)。
- 成功したリクエストの数 (SampleCount)。

SuccessfulRequestLatency は DynamoDB または Amazon DynamoDB Streams 内のアクティビティのみを反映し、ネットワークレイテンシーやクライアント側のアクティビティは考慮されません。

単位: Milliseconds

ディメンション: `TableName, Operation, StreamLabel`

有効な統計:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`

SystemErrors

指定された期間に HTTP 500 ステータスコードを生成する DynamoDB または Amazon DynamoDB Streams へのリクエスト。HTTP 500 は通常、内部サービスエラーを示します。

単位: `Count`

ディメンション: `TableName, Operation`

有効な統計:

- `Sum`
- `SampleCount`

TimeToLiveDeletedItemCount

指定した期間中に有効期限 (TTL) によって削除された項目の数。このメトリクスは、テーブルの TTL 削除率を監視するのに役立ちます。

単位: `Count`

ディメンション: TableName

有効な統計:

- `Sum`

ThrottledPutRecordCount

Kinesis Data Streams のキャパシティが不足しているために Kinesis データストリームによってスロットリングされたレコードの数。

単位: `Count`

ディメンション: TableName, DelegatedOperation

有効な統計:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`

ThrottledRequests

リソース (テーブルやインデックスなど) のプロビジョニング済みスループット制限を超える DynamoDB へのリクエスト。

リクエスト内でいずれかのイベントがプロビジョニング済みスループットクォータを超過した場合、`ThrottledRequests` が 1 つ増加します。例えば、グローバルセカンダリインデックスを持つテーブル内の項目を更新する場合、テーブルへの書き込みと各インデックスへの書き込みという複数のイベントが発生します。これらのイベントの 1 つまたは複数がスロットリングされている場合、`ThrottledRequests` が 1 つ増加します。

> Note
>
> バッチリクエスト (`BatchGetItem` または `BatchWriteItem`) では、`ThrottledRequests` は、バッチ内の各リクエストがスロットリングされた場合にのみ増加します。
> バッチ内の個々のリクエストがスロットリングされると、次のいずれかのメトリクスが増加します。
>
> - `ReadThrottleEvents` — `BatchGetItem` 内のスロットリングされた `GetItem` イベント。
> - `WriteThrottleEvents` — `BatchWriteItem` 内のスロットリングされた `PutItem` または `DeleteItem`。

どのイベントがリクエストをスロットリングしているかについてのインサイトを取得するには、ThrottledRequests とテーブルおよびそのインデックスの ReadThrottleEvents と WriteThrottleEvents を比較します。

> Note
>
> スロットリングされたリクエストは HTTP 400 ステータスコードになります。このようなイベントはすべて、ThrottledRequests メトリクスに反映されますが、UserErrors メトリクスには反映されません。

単位: Count

ディメンション: TableName, Operation

有効な統計:

- Sum
- SampleCount

TransactionConflict

同じ項目に対する同時になされた要求のトランザクション競合が原因で、品目レベルの要求が拒否されました。詳細については、「DynamoDB でのトランザクション競合の処理」を参照してください。

単位: Count

ディメンション: TableName

有効な統計:

- Sum — トランザクションの競合により拒否された項目レベルのリクエストの数。

  > Note
  >
  > TransactWriteItems または TransactGetItems への呼び出し内の項目レベルの複数のリクエストが拒否された場合、Sum は、各項目レベルの Put、Update、Delete、または Get リクエストに対して 1 つ増加します。

- SampleCount — トランザクションの競合により拒否されたリクエストの数。

> Note
>
> TransactWriteItems または TransactGetItems への呼び出し内の項目レベルの複数のリクエストが拒否された場合、SampleCount が 1 つだけ増加します。

- Min — TransactWriteItems、TransactGetItems、PutItem、UpdateItem、または DeleteItem に対する呼び出し内で拒否された項目レベルのリクエストの最小数。
- Max — TransactWriteItems、TransactGetItems、PutItem、UpdateItem、または DeleteItem に対する呼び出し内で拒否された項目レベルのリクエストの最大数。
- Average — TransactWriteItems、TransactGetItems、PutItem、UpdateItem、または DeleteItem に対する呼び出し内で拒否された項目レベルのリクエストの平均数。

UserErrors

指定された期間に HTTP 400 ステータスコードを生成する DynamoDB または Amazon DynamoDB Streams へのリクエスト。HTTP 400 は通常、無効なパラメータの組み合わせ、存在しないテーブルへの更新の試み、不正なリクエスト署名など、クライアント側のエラーを示します。

UserErrors に関連するメトリクスをログに記録する例外の例は次のようになります。

- ResourceNotFoundException
- ValidationException
- TransactionConflict

以下を除き、このようなイベントはすべて UserErrors メトリクスに反映されます。

- ProvisionedThroughputExceededException — このセクションの ThrottledRequests メトリクスを参照してください。
- ConditionalCheckFailedException —このセクションの ConditionalCheckFailedRequests メトリクスを参照してください。

UserErrors は、現在の AWS リージョンおよび現在の AWS アカウントの DynamoDB または Amazon DynamoDB Streams リクエストの HTTP 400 エラーの集計を表します。

単位: Count

有効な統計:

- `Sum`
- `SampleCount`

WriteThrottleEvents

テーブルまたはグローバルセカンダリインデックス用にプロビジョニングされた書き込み容量ユニットを超える DynamoDB へのリクエスト。

1 つのリクエストで複数のイベントが発生する可能性があります。例えば、3 つのグローバルセカンダリインデックスを持つテーブル上の `PutItem` リクエストでは、テーブル書き込みと 3 つのインデックス書き込みという 4 つのイベントが発生します。各イベントでは、そのイベントがスロットリングされている場合、`WriteThrottleEvents` メトリクスは 1 つ増加します。単一の `PutItem` リクエストでは、いずれかのイベントがスロットリングされている場合、`ThrottledRequests` も 1 つ増加します。`BatchWriteItem` では、個々の `PutItem` または `DeleteItem` イベントがすべてスロットリングされた場合を除き、`BatchWriteItem` 全体の `ThrottledRequests` メトリクスは増加しません。

`TableName` ディメンションは、グローバルセカンダリインデックスではなくテーブルの `WriteThrottleEvents` を返します。グローバルセカンダリインデックスの `WriteThrottleEvents` を表示するには、`TableName` と `GlobalSecondaryIndexName` の両方を指定する必要があります。

単位: `Count`

ディメンション: `TableName, GlobalSecondaryIndexName`

有効な統計:

- `Sum`
- `SampleCount`

使用状況メトリクス

CloudWatch の使用状況メトリクスを使用して、使用状況をプロアクティブに管理することができます。これは、CloudWatch コンソールでのメトリクスの可視化、カスタムダッシュボードの作成、CloudWatch 異常検出によるアクティビティの変化の検出、使用量がしきい値に近づいたときに警告するアラームの設定などによって実現します。

DynamoDB では、これらの使用状況メトリクスをサービスクォータと統合します。CloudWatch を使用して、アカウントのサービスクォータの使用を管理できます。詳細については、「サービスクォータの視覚化とアラームの設定」を参照してください。

利用可能な使用状況メトリクスのリスト

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

アカウントのすべてのテーブルおよびグローバルセカンダリインデックスのプロビジョニング済み書き込みキャパシティ単位の数。

単位: `Count`

有効な統計:

- `Minimum` - 一定期間中のプロビジョニング済み書き込みキャパシティ単位の最小数。
- `Maximum` - 一定期間中のプロビジョニング済み書き込みキャパシティ単位の最大数。
- `Average` - 一定期間中のプロビジョニング済み書き込みキャパシティ単位の平均数

このメトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み書き込み容量ユニットをすばやく調整すると、この統計には実際の平均値が反映されないことがあります。

AccountProvisionedReadCapacityUnits

アカウントのすべてのテーブルおよびグローバルセカンダリインデックスのプロビジョニング済み読み込みキャパシティ単位の数。

単位: `Count`

有効な統計:

- `Minimum` - 一定期間中のプロビジョニング済み読み込みキャパシティ単位の最小数。
- `Maximum` - 一定期間中のプロビジョニング済み読み込みキャパシティ単位の最大数。
- `Average` - 一定期間中のプロビジョニング済み読み込みキャパシティ単位の平均数。

このメトリクスは 5 分間隔で発行されます。したがって、プロビジョニング済み読み込み容量ユニットをすばやく調整すると、この統計には実際の平均が反映されないことがあります。

TableCount

アカウントのアクティブなテーブルの数。

単位: `Count`

有効な統計:

- `Minimum` - 一定期間中のテーブルの最小数。
- `Maximum` - 一定期間中のテーブルの最大数。
- `Average` - 一定期間中のテーブルの平均数。

DynamoDB のメトリクスとディメンションについて

DynamoDB のメトリクスは、アカウント、テーブル名、グローバルセカンダリインデックス名、オペレーションなどの値で修飾されます。CloudWatch コンソールを使用して、DynamoDB データ、および以下の表に示すいずれかのディメンションを取得できます。

使用できるディメンションのリスト

- DelegatedOperation
- GlobalSecondaryIndexName
- Operation
- OperationType
- Verb
- ReceivingRegion
- StreamLabel
- TableName

DelegatedOperation

このディメンションは、 DynamoDB がユーザーに代わって実行するオペレーションにデータを制限します。以下のオペレーションがキャプチャされます。

- Kinesis Data Streams でのデータキャプチャの変更

GlobalSecondaryIndexName

このディメンションは、テーブルのグローバルセカンダリインデックスにデータを制限します。`GlobalSecondaryIndexName` を指定する場合は、`TableName` も指定する必要があります。

Operation

このディメンションは、以下の DynamoDB オペレーションタイプのいずれかにデータを制限します。

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`
- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

さらに、データを次の Amazon DynamoDB Streams オペレーションに制限することもできます。

- `GetRecords`

OperationType

このディメンションは、以下のオペレーションタイプのいずれかにデータを制限します。

- `Read`

- `Write`

このディメンションは `ExecuteTransaction` リクエストと `BatchExecuteStatement` リクエストに発行されます。

Verb

このディメンションは、次の DynamoDB PartiQL 動詞のいずれかにデータを制限します。

- Insert: `PartiQLInsert`
- Select: `PartiQLSelect`
- Update: `PartiQLUpdate`
- Delete: `PartiQLDelete`

このディメンションは、`ExecuteStatement` オペレーションに発行されます。

ReceivingRegion

このディメンションは、特定の AWS リージョン内にデータを制限します。これは、DynamoDB グローバルテーブル内のレプリカテーブルから生成されたメトリクスで使用されます。

StreamLabel

このディメンションは、特定のストリームラベルにデータを制限します。これは、Amazon DynamoDB Streams の `GetRecords` オペレーションから送信されたメトリクスとともに使用されます。

TableName

このディメンションは、特定のテーブルにデータを制限します。この値は、現在のリージョンおよび現在の AWS アカウントの任意のテーブル名にすることができます。

CloudWatch アラームの作成

CloudWatch アラームは、指定した期間にわたって単一のメトリクスを監視し、一定期間にわたってしきい値と比較したメトリクスの値に基づいて、1 つ以上の指定したアクションを実行します。アクションは、Amazon SNS のトピックまたは自動スケーリングのポリシーに送信される通知です。ダッシュボードにアラームを追加すると、複数のリージョンにまたがって AWS リソースやアプリケーションをモニタリングし、アラートを受け取ることもできます。作成できるアラームの数に制限

はありません。CloudWatch のアラームは、メトリクスが特定の状態にあるだけではアクションを呼び出しません。アクションを呼び出すには、指定した期間継続している必要があります。推奨される DynamoDB アラームのリストについては、「推奨アラーム」を参照してください。

> Note
>
> CloudWatch アラームを作成するときは、必要なすべてのディメンションを指定する必要があります。これは、CloudWatch が欠落しているディメンションのメトリクスを集計しないためです。ディメンションが欠落している CloudWatch アラームを作成しても、アラームの作成時にエラーは発生しません。

1 つのプロビジョニングされたテーブル内に、5 つの読み込みキャパシティユニットがあるとします。この場合、プロビジョニングされた読み込みキャパシティ全体を消費する前に通知を受け取るには、テーブルのプロビジョニングされたキャパシティの消費量が 80% に達したときに通知を送信するように CloudWatch アラームを作成します。アラームは、CloudWatch コンソールまたは AWS CLI を使用して作成できます。

CloudWatch コンソールでのアラームの作成

CloudWatch コンソールでアラームを作成するには

1. AWS Management Console にサインインして、CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[アラーム]、[すべてのアラーム] の順に選択します。
3. [アラームの作成] を選択します。
4. [メトリクス名] 列で、モニタリングするテーブルと **`ConsumeReadCapacityUnits`** を含む行を見つけます。この行の横にあるチェックボックスをオンにして、[メトリクスを選択] を選択します。
5. [メトリクスと条件の指定] で、[統計] の [合計] を選択します。[期間] として [1 分] を選択します。
6. [Conditions (条件)] で、次のように指定します。

    a. [Threshold type (しきい値タイプ)] で [静的] を選択します。

    b. [**`ConsumedReadCapacityUnits`** が以下のとき] で、[より大きい/等しい] を選択し、しきい値を 240 に指定します。
7. [Next] を選択します。

8. [通知] で、[**In alarm**] を選択し、アラームが ALARM 状態のときに通知する SNS トピックを選択します。
9. 完了したら、[次へ] を選択します。
10. 次のページで、アラームの名前と説明を入力し、[次へ] を選択します。
11. [Preview and create (プレビューして作成)] で、情報と条件が正しいことを確認し、[アラームの作成] を選択します。

AWS CLI でのアラームの作成

```
aws cloudwatch put-metric-alarm \
    -\-alarm-name ReadCapacityUnitsLimitAlarm \
    -\-alarm-description "Alarm when read capacity reaches 80% of my provisioned read
 capacity" \
    -\-namespace AWS/DynamoDB \
    -\-metric-name ConsumedReadCapacityUnits \
    -\-dimensions Name=TableName,Value=myTable \
    -\-statistic Sum \
    -\-threshold 240 \
    -\-comparison-operator GreaterThanOrEqualToThreshold \
    -\-period 60 \
    -\-evaluation-periods 1 \
    -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

アラームのテストを行います。

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value OK
```

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value ALARM
```

その他の AWS CLI の例

次の手順では、テーブルのプロビジョニングされたスループットクォータを超えるリクエストがある場合に、通知を受け取る方法を示します。

1. Amazon SNS トピック `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput` を作成します。詳細については、「Amazon Simple Notification Service をセットアップする」を参照してください。

2. アラームを作成します。

```
aws cloudwatch put-metric-alarm \
        -\-alarm-name ReadCapacityUnitsLimitAlarm \
        -\-alarm-description "Alarm when read capacity reaches 80% of my
 provisioned read capacity" \
        -\-namespace AWS/DynamoDB \
        -\-metric-name ConsumedReadCapacityUnits \
        -\-dimensions Name=TableName,Value=myTable \
        -\-statistic Sum \
        -\-threshold 240 \
        -\-comparison-operator GreaterThanOrEqualToThreshold \
        -\-period 60 \
        -\-evaluation-periods 1 \
        -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

3. アラームのテストを行います。

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value ALARM
```

次の手順では、システムエラーが発生した場合に通知を受け取る方法を示します。

1. Amazon SNS トピック `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors` を作成します。詳細については、「Amazon Simple Notification Service をセットアップする」を参照してください。

2. アラームを作成します。

```
aws cloudwatch put-metric-alarm \
    --alarm-name SystemErrorsAlarm \
    --alarm-description "Alarm when system errors occur" \
    --namespace AWS/DynamoDB \
    --metric-name SystemErrors \
    --dimensions Name=TableName,Value=myTable
 Name=Operation,Value=aDynamoDBOperation \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --treat-missing-data breaching \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. アラームのテストを行います。

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value ALARM
```

AWS CloudTrail を使用して DynamoDB オペレーションをログに記録する

DynamoDB は、DynamoDB のユーザー、ロール、または AWS のサービスによって実行されたアクションをレコードするサービスである AWS CloudTrail と統合されています。CloudTrail は、DynamoDB のすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、DynamoDB コンソールからの呼び出しと PartiQL とクラシック API の両方を使用した DynamoDB API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、DynamoDB のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。追跡を設定しない場合でも、CloudTrail コンソールの [Event history] (イベント履歴) で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、DynamoDB に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

堅牢なモニタリングとアラートのために、CloudTrail イベントを Amazon CloudWatch Logs と統合することもできます。DynamoDB サービスアクティビティの分析を強化し、AWS アカウントのアクティビティの変更を特定するには、Amazon Athena を使用して、AWS CloudTrail をクエリできます。例えば、クエリを使用して傾向を識別したり、アクティビティを属性 (ソース IP アドレスやユーザーなど) でさらに分離したりできます。

設定や有効化の方法など、CloudTrail の詳細については、「 AWS CloudTrail ユーザーガイド」を参照してください。

トピック

- CloudTrail 内の DynamoDB 情報
- DynamoDB ログファイルのエントリについて

CloudTrail 内の DynamoDB 情報

AWS アカウントを作成すると、そのアカウントに対して CloudTrail が有効になります。DynamoDB でサポートされているイベントアクティビティが発生すると、そのアクティビティは [イベント履歴] の他の AWS のサービスのイベントとともに CloudTrail イベントにレコードされます。AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「CloudTrail イベント履歴の操作」を参照してください。

DynamoDB のイベントなど、AWS アカウントのイベントの継続的なレコードについては、証跡を作成します。追跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての AWS リージョンに証跡が適用されます。証跡は、AWSパーティションのすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づいて対応するため、他の AWS サービスを構成できます。詳細については、次を参照してください:

- 「証跡作成の概要」
- 「CloudTrail がサポートされているサービスと統合」
- CloudTrail の Amazon SNS 通知の設定
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

CloudTrail のコントロールプレーンイベント

デフォルトで、以下の API アクションはイベントとして CloudTrail ファイルに記録されます。

Amazon DynamoDB

- CreateBackup
- CreateGlobalTable
- CreateTable
- DeleteBackup

- DeleteTable
- DescribeBackup
- DescribeContinuousBackups
- DescribeGlobalTable
- DescribeLimits
- DescribeTable
- DescribeTimeToLive
- ListBackups
- ListTables
- ListTagsOfResource
- ListGlobalTables
- RestoreTableFromBackup
- RestoreTableToPointInTime
- TagResource
- UntagResource
- UpdateGlobalTable
- UpdateTable
- UpdateTimeToLive
- DescribeReservedCapacity
- DescribeReservedCapacityOfferings
- PurchaseReservedCapacityOfferings
- DescribeScalableTargets
- RegisterScalableTarget

DynamoDB Streams

- DescribeStream
- ListStreams

DynamoDB Accelerator (DAX)

- CreateCluster

- CreateParameterGroup
- CreateSubnetGroup
- DecreaseReplicationFactor
- DeleteCluster
- DeleteParameterGroup
- DeleteSubnetGroup
- DescribeClusters
- DescribeDefaultParameters
- DescribeEvents
- DescribeParameterGroups
- DescribeParameters
- DescribeSubnetGroups
- IncreaseReplicationFactor
- ListTags
- RebootNode
- TagResource
- UntagResource
- UpdateCluster
- UpdateParameterGroup
- UpdateSubnetGroup

CloudTrail の DynamoDB データプレーンイベント

CloudTrail ファイルで以下の API アクションのロギングを有効にするには、CloudTrail でデータプレーン API アクティビティのログ記録を有効にする必要があります。詳細については、「トレイルのデータイベントのログ記録」を参照してください。

データプレーンイベントは、リソースタイプ別にフィルタリングして、CloudTrail でログと支払いを選択的に実行する DynamoDB API 呼び出しをきめ細かくコントロールできます。例えば、リソースタイプに `AWS::DynamoDB::Stream` を指定することで、DynamoDB ストリーム API への呼び出しのみをログに記録できます。ストリームが有効になっているテーブルの場合、データプレーンイベントのリソースフィールドには `AWS::DynamoDB::Stream` と `AWS::DynamoDB::Table` 両方が含まれます。リソースタイプに `AWS::DynamoDB::Table` を指定すると、デフォルトで DynamoDB

テーブルと DynamoDB ストリームイベントの両方がログに記録されます。ストリームイベントをログに記録しない場合、ストリームイベントを除外するフィルターを追加できます。詳細については、「AWS CloudTrail API リファレンス」の「DataResource」を参照してください。

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Query
- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> Note
>
> CloudTrail では DynamoDB の有効期限 (TTL) データプレーンのアクションはログに記録されません。

DynamoDB Streams

- GetRecords
- GetShardIterator

DynamoDB ログファイルのエントリについて

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意の発生元

からの 1 つのリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどに関する情報が含まれます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。ID 情報は次の判断に役立ちます。

- リクエストが、ルートとユーザー認証情報のどちらを使用して送信されたか。
- リクエストが、ロールとフェデレーションユーザーの一時的なセキュリティ認証情報のどちらを使用して送信されたか。
- リクエストが別の AWS サービスによって行われたかどうか。

> Note
>
> 非キー属性値は、PartiQL API を使用してアクションの CloudTrail ログで編集され、クラシック API を使用するアクションのログには表示されません。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

以下の例は、これらのイベントタイプの CloudTrail ログを示しています。

Amazon DynamoDB

- `UpdateTable`
- `DeleteTable`
- `CreateCluster`
- `PutItem` (成功)
- `UpdateItem` (失敗)
- `TransactWriteItems` (成功)
- `TransactWriteItems` (`TransactionCanceledException` を含む)
- `ExecuteStatement`
- `BatchExecuteStatement`

DynamoDB Streams

- `GetRecords`

UpdateTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2015-05-04T02:14:52Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "provisionedThroughput": {
                    "writeCapacityUnits": 25,
                    "readCapacityUnits": 25
                }
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "attributeDefinitions": [
                        {
                            "attributeType": "S",
                            "attributeName": "Artist"
                        },
                        {
                            "attributeType": "S",
                            "attributeName": "SongTitle"
                        }
                    ],
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 10,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 10,
                        "lastIncreaseDateTime": "May 3, 2015 11:34:14 PM"
                    },
                    "creationDateTime": "May 3, 2015 11:34:14 PM",
                    "keySchema": [
                        {
                            "attributeName": "Artist",
                            "keyType": "HASH"
                        },
                        {
                            "attributeName": "SongTitle",
                            "keyType": "RANGE"
                        }
                    ],
                    "tableStatus": "UPDATING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "AALNP0J2L244N5O15PKISJ1KUFVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "eb834e01-f168-435f-92c0-c36278378b6e",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

DeleteTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2015-05-04T13:38:20Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "DeleteTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "tableName": "Music"
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 25,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 25
                    },
                    "tableStatus": "DELETING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

CreateCluster

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "bob"
            },
            "eventTime": "2019-12-17T23:17:34Z",
            "eventSource": "dax.amazonaws.com",
            "eventName": "CreateCluster",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.304 Python/3.6.9
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.13.40",
            "requestParameters": {
                "sSESpecification": {
                    "enabled": true
                },
                "clusterName": "daxcluster",
                "nodeType": "dax.r4.large",
                "replicationFactor": 3,
                "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess"
            },
            "responseElements": {
                "cluster": {
                    "securityGroups": [
                        {
                            "securityGroupIdentifier": "sg-1af6e36e",
                            "status": "active"
                        }
                    ],
                    "parameterGroup": {
                        "nodeIdsToReboot": [],
                        "parameterGroupName": "default.dax1.0",
                        "parameterApplyStatus": "in-sync"
                    },
                    "clusterDiscoveryEndpoint": {
                        "port": 8111
                    },
                    "clusterArn": "arn:aws:dax:us-west-2:111122223333:cache/
daxcluster",
                    "status": "creating",
                    "subnetGroup": "default",
                    "sSEDescription": {
                        "status": "ENABLED",
                        "kMSMasterKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/764898e4-adb1-46d6-a762-e2f4225b4fc4"
                    },
                    "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess",
                    "clusterName": "daxcluster",
                    "activeNodes": 0,
                    "totalNodes": 3,
                    "preferredMaintenanceWindow": "thu:13:00-thu:14:00",
                    "nodeType": "dax.r4.large"
                }
            },
            "requestID": "585adc5f-ad05-4e27-8804-70ba1315f8fd",
            "eventID": "29158945-28da-4e32-88e1-56d1b90c1a0c",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

PutItem (成功)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2019-01-19T15:41:54Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "PutItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Scared of My Shadow"
                },
                "item": [
                    "Artist",
                    "SongTitle",
                    "AlbumTitle"
                ],
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

UpdateItem (失敗)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T22:27:15Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "errorCode": "ConditionalCheckFailedException",
            "errorMessage": "The conditional request failed",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Today"
                },
                "updateExpression": "SET #Y = :y, #AT = :t",
                "expressionAttributeNames": {
                    "#Y": "Year",
                    "#AT": "AlbumTitle"
                },
                "conditionExpression": "attribute_not_exists(#Y)",
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

TransactWriteItems (成功)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T21:48:12Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
                        "ConditionExpression": "attribute_not_exists(Rating)",
                        "ExpressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Delete",
                        "TableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Yesterday"
                        },
                        "conditionExpression": "#P between :lo and :hi",
                        "expressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "ConditionCheck",
                        "TableName": "Music",
                        "Key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Now"
                        },
                        "ConditionExpression": "#P between :lo and :hi",
                        "ExpressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    }
                ],
                "returnConsumedCapacity": "TOTAL",
                "returnItemCollectionMetrics": "SIZE"
            },
            "responseElements": null,
            "requestID": "45EN32OM6TQSMV2MI65O4L5TNFVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "4f1cc78b-5c94-4174-a6ad-3ee78605381c",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

TransactWriteItems (TransactionCanceledException を含む)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2019-02-01T00:42:34Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.93 Python/3.4.7
Linux/4.9.119-0.1.ac.277.71.329.metal1.x86_64 botocore/1.12.83",
            "errorCode": "TransactionCanceledException",
            "errorMessage": "Transaction cancelled, please refer cancellation reasons
for specific reasons [ConditionalCheckFailed, None]",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
                        "ConditionExpression": "attribute_not_exists(Rating)",
                        "ExpressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Delete",
                        "TableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Yesterday"
                        },
                        "conditionExpression": "#P between :lo and :hi",
                        "expressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "ConditionCheck",
                        "TableName": "Music",
                        "Key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Now"
                        },
                        "ConditionExpression": "#P between :lo and :hi",
                        "ExpressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    }
                ],
                "returnConsumedCapacity": "TOTAL",
                "returnItemCollectionMetrics": "SIZE"
            },
            "responseElements": null,
            "requestID": "A0GTQEKLBB9VD8E05REA5A3E1VVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "43e437b5-908a-46af-84e6-e27fffb9c5cd",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

ExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:06:45Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "ExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "statement": "SELECT * FROM Music WHERE Artist = 'No One You Know' AND
 SongTitle = 'Call Me Today' AND nonKeyAttr = ***(Redacted)"
            },
            "responseElements": null,
            "requestID": "V7G2KCSFLP83ORB7MMFG6RIAD3VV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "0b5c4779-e169-4227-a1de-6ed01dd18ac7",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

BatchExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:24:48Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "BatchExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13 
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "requestItems": [
                    {
                        "statement": "UPDATE Music SET Album = ***(Redacted) WHERE 
 Artist = 'No One You Know' AND SongTitle = 'Call Me Today'"
                    },
                    {
                        "statement": "INSERT INTO Music VALUE {'Artist' : 
 ***(Redacted), 'SongTitle' : ***(Redacted), 'Album' : ***(Redacted)}"
                    }
                ]
            },
            "responseElements": null,
            "requestID": "23PE7ED291UD65P9SMS6TISNVBVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "f863f966-b741-4c36-b15e-f867e829035a",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
```

```
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

GetRecords

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-04-15T04:15:02Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "GetRecords",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.50 Python/3.6.13
 Linux/4.9.230-0.1.ac.224.84.332.metal1.x86_64 botocore/1.20.50",
            "requestParameters": {
                "shardIterator": "arn:aws:dynamodb:us-west-2:123456789012:table/
Music/stream/2021-04-15T04:02:47.428|1|AAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGV1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpluWsHyCW3PPbKsMlJSruVRWoitRhSd3S6slEWEPB0bDC7+
+ISH5mXrCHOnvyezQKlqNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bpOUI2cuRTchgUpPSe3ur2sQrRj3KlbmIyCz7P
+H3CYlugafi8fQ5kipDSkESkIWS6O5ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAk3ab68aUXGSFEZLONntgNIhQkcMo00/
mJlaIgkEdBUyqvZO1vtKUBH5YonIrZqSUhv8Coc+mh24vOg1YI+SPIXlr
+Lnl54BG6AjrmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cskcX2ISaLHwYzJPaMBSftxOgeRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXmOgay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMhlBVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrzOEzeB4uconEXAMPLE=="
            },
            "responseElements": null,
            "requestID": "1M0U1Q80P4LDPT7A7N1A758N2VVV4KQNSO5AEMVJF66Q9EXAMPLE",
            "eventID": "09a634f2-da7d-4c9e-a259-54aceexample",
            "readOnly": true,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

CloudWatch Contributor Insights for DynamoDB を使用してデータアクセスを分析する

Amazon CloudWatch Contributor Insights for Amazon DynamoDB は、テーブルやインデックス内で最も頻繁にアクセスおよび調整されるキーをすばやく特定する診断ツールです。このツールでは、CloudWatch Contributor Insights を使用します。

テーブルまたはグローバルセカンダリインデックスで CloudWatch Contributor Insights for DynamoDB を有効にすることで、これらのリソースで最もアクセスが多く調整された項目を表示できます。

> **Note**
>
> DynamoDB の Contributor Insights には CloudWatch の料金が適用されます。料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

トピック

CloudWatch Contributor Insights for DynamoDB: 仕組み

Amazon DynamoDB は、Amazon CloudWatch Contributor Insights との統合により、テーブルやグローバルセカンダリインデックスでアクセスとスロットリングが最も多い項目に関する情報を提供します。DynamoDB は、CloudWatch Contributor Insights のルール、レポート、レポートデータのグラフを通じて、この情報を提供します。

詳細については、「Amazon CloudWatch ユーザーガイド」の「Contributor Insights を使用して高カーディナリティデータを分析する」を参照してください。

次のセクションでは、CloudWatch Contributor Insights for DynamoDB の基本概念と動作について説明します。

トピック

- CloudWatch Contributor Insights for DynamoDB のルール
- CloudWatch Contributor Insights for DynamoDB のグラフを理解する
- 他の DynamoDB 機能との相互作用
- CloudWatch Contributor Insights for DynamoDB の請求

CloudWatch Contributor Insights for DynamoDB のルール

テーブルまたはグローバルセカンダリインデックスで DynamoDB の CloudWatch Contributor Insights を有効にすると、DynamoDB はユーザーに代わって次のルールを作成します。

- 最もアクセスされた項目 (パーティションキー) - テーブルまたはグローバルセカンダリインデックスで最もアクセスされた項目のパーティションキーを識別します。

  CloudWatch ルール名の形式: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`
- 最もスロットリングされたキー (パーティションキー) - テーブルまたはグローバルセカンダリインデックスで最もスロットリングされた項目のパーティションキーを識別します。

  CloudWatch ルール名の形式: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

> **Note**
>
> DynamoDB テーブルで Contributor Insights を有効にする場合でも、Contributor Insights ルールによる制限は引き続き適用されます。詳細については、「CloudWatch サービスのクォータ」を参照してください。

テーブルまたはグローバルセカンダリインデックスにソートキーがある場合、DynamoDB はソートキーに固有の次のルールも作成します。

- 最もアクセスされたキー (パーティションキーとソートキー) - テーブルまたはグローバルセカンダリインデックスで最もアクセスされた項目のパーティションキーとソートキーを識別します。

  CloudWatch ルール名の形式: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`

- 最もスロットリングされたキー (パーティションキーとソートキー) - テーブルまたはグローバルセカンダリインデックスで最もスロットリングされた項目のパーティションキーとソートキーを識別します。

  CloudWatch ルール名の形式: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

> **Note**
>
> - CloudWatch コンソールまたは API を使用して、CloudWatch Contributor Insights for DynamoDB によって作成されたルールを直接変更または削除することはできません。テーブルまたはグローバルセカンダリインデックスで CloudWatch Contributor Insights for DynamoDB を無効にすると、そのテーブルまたはグローバルセカンダリインデックス用に作成されたルールが自動的に削除されます
> - DynamoDB によって作成された CloudWatch Contributor Insights ルールで GetInsightRuleReport オペレーションを使用すると、`MaxContributorValue` と `Maximum` のみが有用な統計を返します。このリストの他の統計は、意味のある値を返しません。
> - CloudWatch Contributor Insights for DynamoDB の寄稿者の上限は 25 です。25 人を超える寄稿者をリクエストすると、エラーが返されます。

CloudWatch アラームは、CloudWatch Contributor Insights for DynamoDB ルールを使用して作成できます。これにより、項目が `ConsumedThroughputUnits` または `ThrottleCount` の特定のしきい値を超えたとき、または満たすときに通知を受け取ることができます。詳細については、「Contributor Insights メトリクスデータでのアラームの設定」を参照してください。

CloudWatch Contributor Insights for DynamoDB のグラフを理解する

Contributor Insights for DynamoDB は、DynamoDB コンソールと CloudWatch コンソールの両方で 2 種類のグラフを表示します ([最もアクセスされた項目] および [最もスロットリングされた項目])。

最もアクセスされた項目

このグラフを使用して、テーブルまたはグローバルセカンダリインデックスで最もアクセスの多い項目を特定します。グラフの Y 軸に `ConsumedThroughputUnits` が、X 軸に時間が表示されます。上位 N 個のキーはそれぞれ独自の色で表示され、X 軸の下に凡例が表示されます。

DynamoDB は、ConsumedThroughputUnits を使用してキーアクセス頻度を測定します。これは、読み取りと書き込みのトラフィックを組み合わせて測定します。ConsumedThroughputUnits は次のように定義されます。

- プロビジョンド - (3 x 消費された書き込み容量の単位) + 消費された読み込み容量の単位
- オンデマンド - (3 x 書き込みリクエスト単位) + 読み込みリクエスト単位

DynamoDB コンソールで、グラフの各データポイントは 1 分間にわたる ConsumedThroughputUnits の最大値を表します。例えば、ConsumedThroughputUnits のグラフ値が 180,000 である場合は、1 分間以内の 60 秒間にわたり、項目別の最大スループットを 1,000 の書き込みリクエスト単位または 3,000 の読み取りリクエスト単位として、項目が継続的にアクセスされたことを示します (3,000 x 60 秒)。つまり、グラフ化された値は、各 1 分間以内で最もトラフィックの多い 1 分間を表します。ConsumedThroughputUnits メトリクスの時間粒度は CloudWatch コンソールで変更できます (1 分間ではなく 5 分間のメトリクスを表示するなど)。

明確な異常値がなく、いくつかの密接にクラスター化された行が表示されている場合、指定された時間枠内で項目間でワークロードのバランスが比較的取れていることを示しています。グラフに接続線ではなく孤立したポイントが表示されている場合は、項目が短時間だけ頻繁にアクセスされたこと示します。

テーブルまたはグローバルセカンダリインデックスにソートキーがある場合、DynamoDB は 2 つのグラフを作成します。1 つは最もアクセス頻度の高いパーティションキー用、もう 1 つは最もアクセス頻度の高いパーティションキーとソートキーのペア用です。パーティションキーレベルのトラフィックは、パーティションキー専用のグラフでのみ確認できます。項目レベルのトラフィックは、パーティションキーとソートキーのペア用グラフで確認できます。

最もスロットリングされた項目

このグラフを使用して、テーブルまたはグローバルセカンダリインデックスで最もスロットリングされた項目を特定します。グラフの Y 軸に ThrottleCount が、X 軸に時間が表示されます。上位 N 個の各キーは独自の色で表示され、X 軸の下に凡例が表示されます。

DynamoDB は ThrottleCount を使用してスロットル頻度を測定します。これは、ProvisionedThroughputExceededException、ThrottlingException、および RequestLimitExceeded エラーのカウントです。

グローバルセカンダリインデックスの書き込み容量が不十分なため発生した書き込みスロットルは測定されません。グローバルセカンダリインデックスの [最もアクセスされた項目] グラフを使用して、書き込みスロットリングを引き起こす可能性のある不均衡なアクセスパターンを特定できます。

詳細については、「グローバルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項」を参照してください。

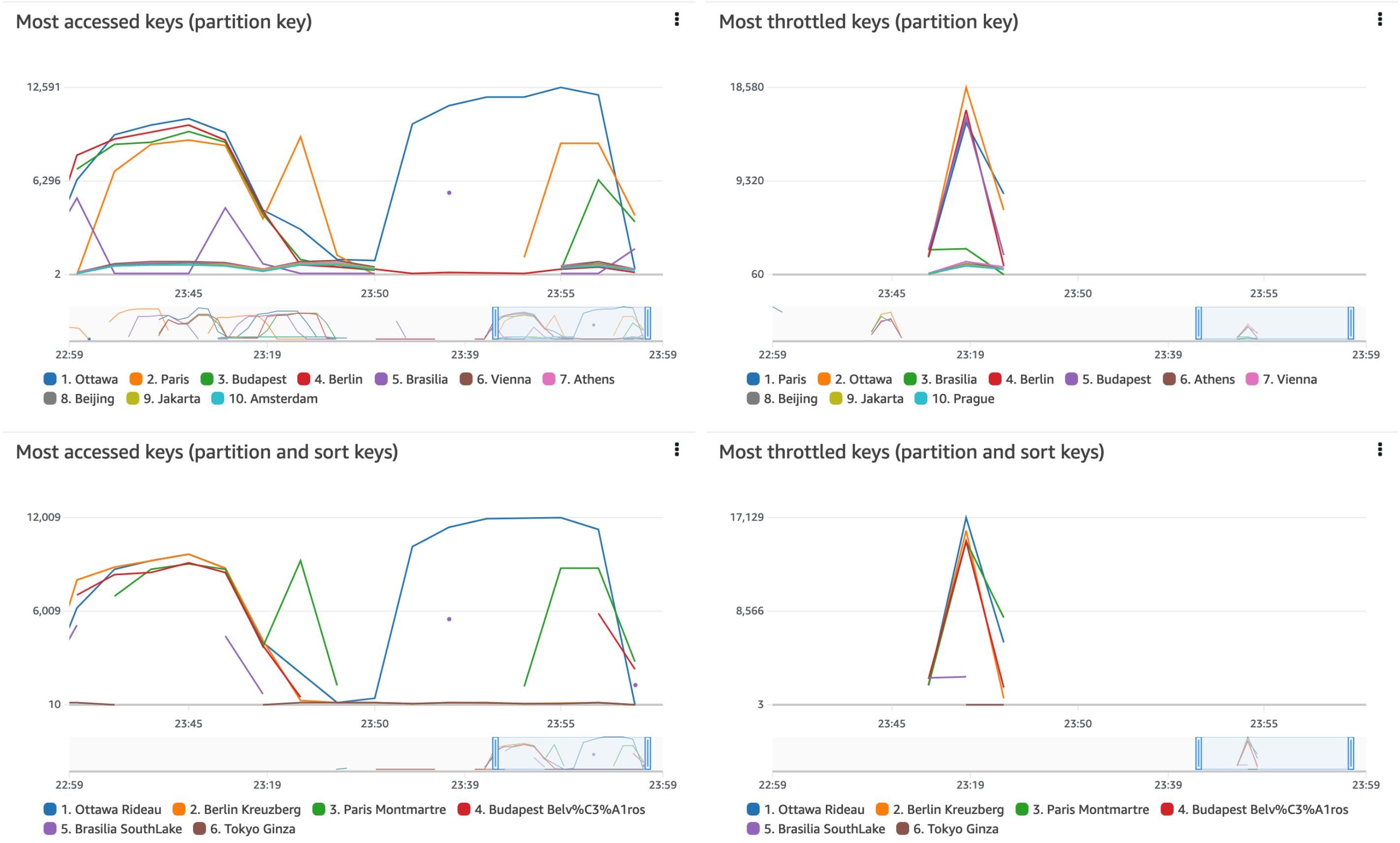
DynamoDB コンソールの場合、グラフの各データポイントは 1 分間のスロットルイベント数を表します。

このグラフにデータが表示されない場合は、リクエストがスロットリングされていないことを示しています。グラフに接続線ではなく孤立したポイントが表示されている場合は、項目が短時間に頻繁にスロットリングされたことを示します。

テーブルまたはグローバルセカンダリインデックスにソートキーがある場合、DynamoDB は 2 つのグラフを作成します。1 つは最もスロットリングされたパーティションキー用、もう 1 つは最もスロットリングされたパーティションキーとソートキーのペア用です。パーティションキーのみのグラフではパーティションキーレベルのスロットルカウントを、パーティションキーとソートキーのグラフでは項目レベルのスロットルカウントを確認できます。

レポートの例

パーティションキーとソートキーの両方があるテーブルに対して生成されたレポートの例を以下に示します。



他の DynamoDB 機能との相互作用

次のセクションでは、CloudWatch Contributor Insights for DynamoDB がどのように動作し、DynamoDB の他のいくつかの機能とどのようにやり取りするかについて説明します。

グローバルテーブル

CloudWatch Contributor Insights for DynamoDB は、グローバルテーブルのレプリカを個別のテーブルとしてモニタリングします。Contributor Insights のグラフに表示されるレプリカのパターンは、AWS リージョン間で異なる場合があります。これは、書き込みデータはグローバルテーブル内のすべてのレプリカにレプリケートされますが、リージョンにバインドされた読み込みトラフィックは各レプリカで処理できるためです。

DynamoDB Accelerator (DAX)

CloudWatch Contributor Insights for DynamoDB は、DAX のキャッシュ応答を表示しません。テーブルまたはグローバルセカンダリインデックスへのアクセスに対する応答のみを表示します。

> **Note**
>
> DynamoDB CCI は PartiQL リクエストをサポートしていません。

保管中の暗号化

CloudWatch Contributor Insights for DynamoDB は、DynamoDB での暗号化の動作には影響しません。CloudWatch で発行されるプライマリキーデータは、AWS 所有のキー で暗号化されます。ただし、DynamoDB は AWS マネージドキー やカスタマーマネージドキーもサポートしています。

CloudWatch Contributor Insights for DynamoDB グラフには、頻繁にアクセスされる項目と頻繁にスロットリングされる項目のパーティションキーとソートキー（該当する場合）がプレーンテキストで表示されます。AWS Key Management Service (KMS) を使用してこのテーブルのパーティションキーを暗号化し、AWS マネージドキー またはカスタマー管理のキーでキーデータをソートする必要がある場合は、このテーブルに対して [CloudWatch Contributor Insights for DynamoDB] を有効にしないでください。

プライマリキーデータを AWS マネージドキー またはカスタマーマネージドキーで暗号化する必要がある場合、そのテーブルで CloudWatch Contributor Insights for DynamoDB を有効にしないでください。

きめ細かなアクセスコントロール

CloudWatch Contributor Insights for DynamoDB は、きめ細かなアクセスコントロール (FGAC) を使用してテーブル別に動作を調整することはありません。つまり、適切な CloudWatch 許可を持つユーザーは、CloudWatch Contributor Insights のグラフで FGAC で保護されたプライマリキーを表示できます。

テーブルのプライマリキーに、CloudWatch に公開したくない FGAC で保護されたデータが含まれている場合、そのテーブルに対して CloudWatch Contributor Insights for DynamoDB を有効にしないでください。

アクセスコントロール

DynamoDB コントロールプレーンの許可と CloudWatch データプレーンの許可を制限することにより、AWS Identity and Access Management (IAM) を使用して、CloudWatch Contributor Insights for DynamoDB へのアクセスを制御します。詳細については、「CloudWatch Contributor Insights for DynamoDB での IAM の使用」を参照してください。

CloudWatch Contributor Insights for DynamoDB の請求

Contributor Insights for DynamoDB の料金は、毎月の請求書の CloudWatch セクションに表示されます。これらの料金は、処理された DynamoDB イベント数に基づいて計算されます。CloudWatch Contributor Insights for DynamoDB が有効になっているテーブルとグローバルセカンダリインデックスの場合、データプレーンオペレーションを介して書き込みまたは読み込みが実行される各項目は 1 つのイベントを表します。

テーブルまたはグローバルセカンダリインデックスにソートキーが含まれている場合、読み込みまたは書き込みされる各項目は 2 つのイベントを表します。これは、DynamoDB が別々の時系列から上位の貢献者を識別するためです。1 つはパーティションキー専用の時系列、もう 1 つはパーティションキーとソートキーのペア用の時系列です。

例えば、5 つの項目を配置する `GetItem`、`PutItem`、`BatchWriteItem` の DynamoDB オペレーションがアプリケーションによって実行されたとします。

- テーブルまたはグローバルセカンダリインデックスにパーティションキーのみがある場合、イベント数は 7 になります (`GetItem` に 1、`PutItem` に 1、`BatchWriteItem` に 5)。
- テーブルまたはグローバルセカンダリインデックスにパーティションキーとソートキーがある場合、イベント数は 14 になります (`GetItem` に 2、`PutItem` に 2、`BatchWriteItem` に 10)。
- `Query` オペレーションでは、返される項目の数に関係なく、イベント数は常に 1 です。

DynamoDB の他の機能とは異なり、CloudWatch Contributor Insights for DynamoDB の請求は以下に基づいて変動しません。

- 容量モード (プロビジョンドとオンデマンド)
- 読み込みリクエストと書き込みリクエストのどちらを実行するか
- 読み込みまたは書き込みされた項目のサイズ (KB)

CloudWatch Contributor Insights for DynamoDB の開始方法

このセクションでは、Amazon DynamoDB コンソールまたは AWS Command Line Interface (AWS CLI) で Amazon CloudWatch Contributor Insights を使用する方法について説明します。

次の例では、「DynamoDB の使用開始」チュートリアルで定義された DynamoDB テーブルを使用します。

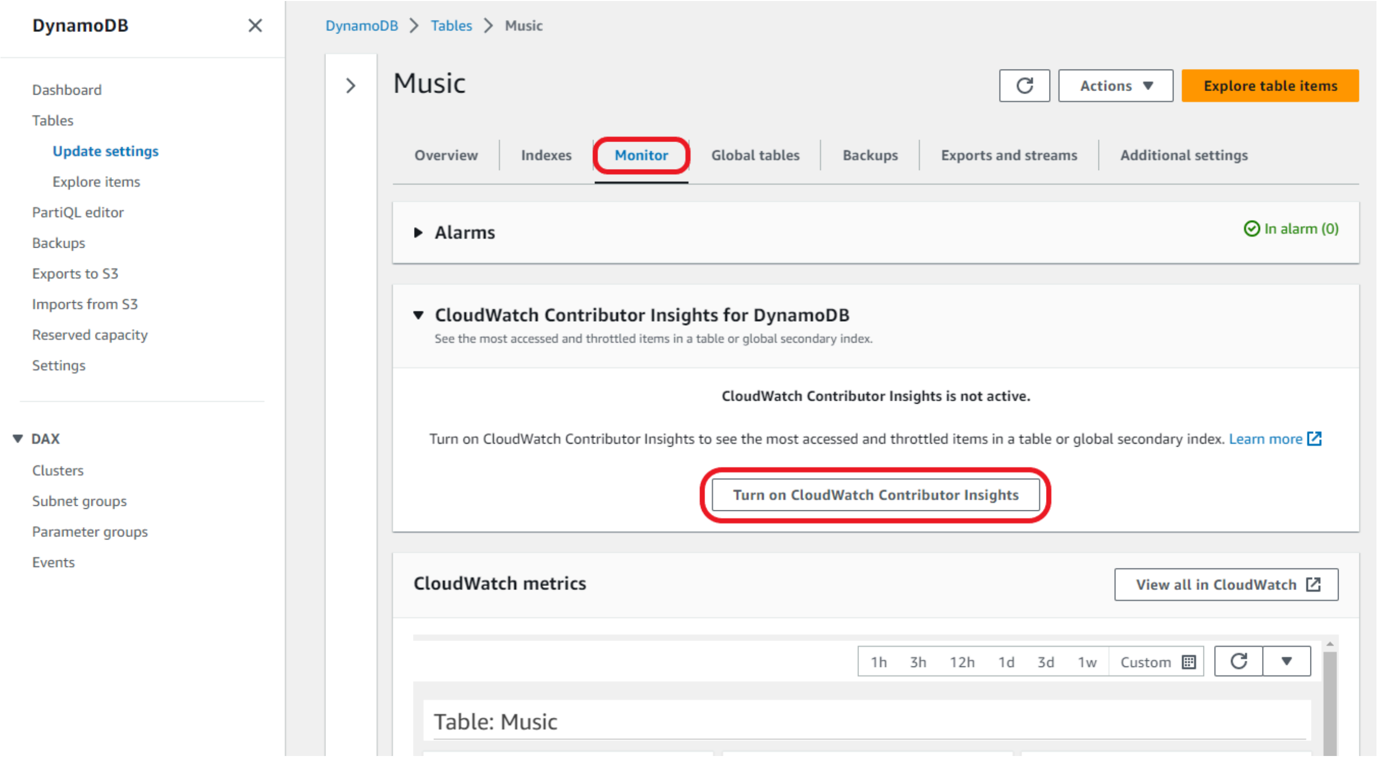
トピック

- Contributor Insights の使用 (コンソール)
- Contributor Insights の使用 (AWS CLI)

Contributor Insights の使用 (コンソール)

コンソールで Contributor Insights を使用するには

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
3. `Music` テーブルを選択します。
4. [Monitor] (モニタリング) タブを選択します。
5. [Turn on CloudWatch Contributor Insights] (CloudWatch Contributor Insights を有効にする) を選択します。



CloudWatch Contributor Insights for DynamoDB

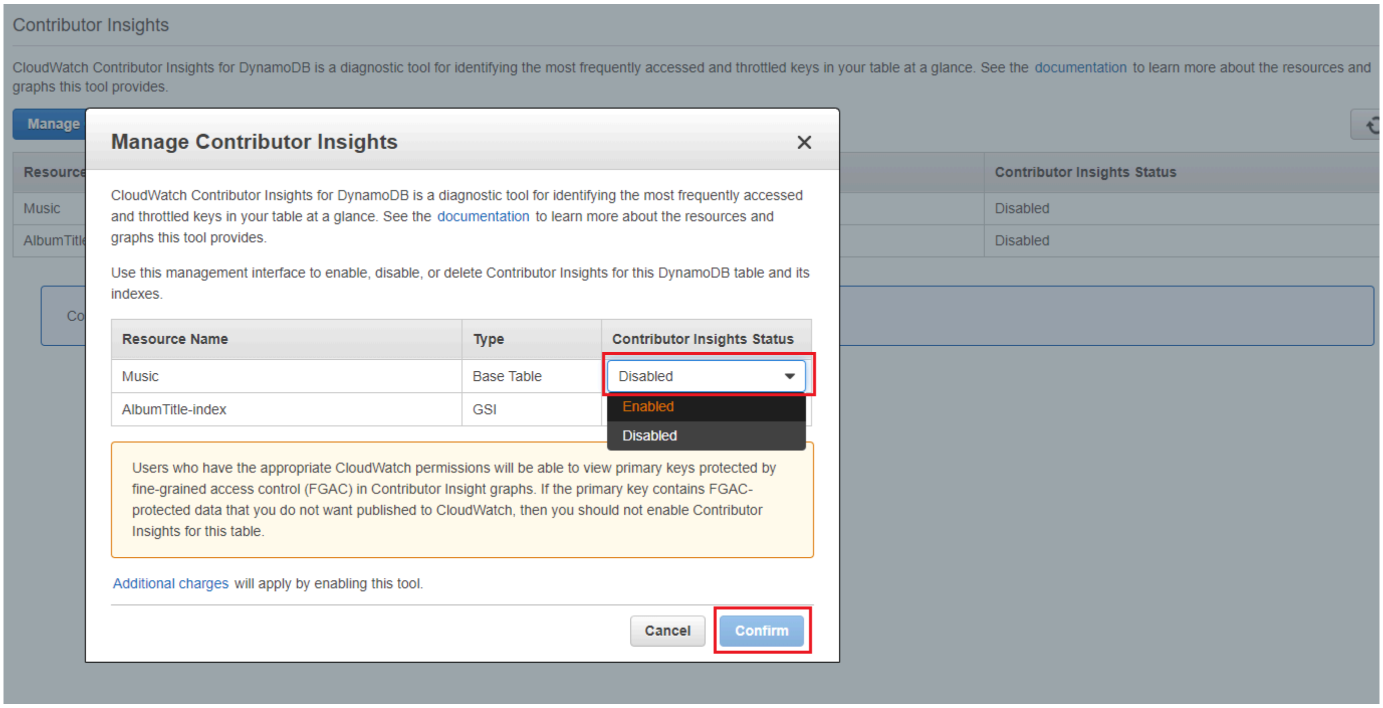
See the most accessed and throttled items in a table or global secondary index.

CloudWatch Contributor Insights is not active.

Turn on CloudWatch Contributor Insights to see the most accessed and throttled items in a table or global secondary index. Learn more

Turn on CloudWatch Contributor Insights

6. [Manage Contributor Insights] (投稿者のインサイトを管理) ダイアログボックスの [Contributor Insights Status] (Contributor Insights のステータス) で、`Music` 基本テーブルと `AlbumTitle-index` グローバルセカンダリインデックスの両方に対して [Enabled] (有効) を選択します。[Confirm] (確認) を選択します。


Contributor Insights
CloudWatch Contributor Insights for DynamoDB is a diagnostic tool for identifying the most frequently accessed and throttled keys in your table at a glance. See the documentation to learn more about the resources and graphs this tool provides.
Manage
Resource
Music
AlbumTitl
Contributor Insights Status
Disabled
Disabled
Co

Manage Contributor Insights

CloudWatch Contributor Insights for DynamoDB is a diagnostic tool for identifying the most frequently accessed and throttled keys in your table at a glance. See the documentation to learn more about the resources and graphs this tool provides.

Use this management interface to enable, disable, or delete Contributor Insights for this DynamoDB table and its indexes.

Resource Name	Type	Contributor Insights Status
Music	Base Table	Disabled
AlbumTitle-index	GSI	

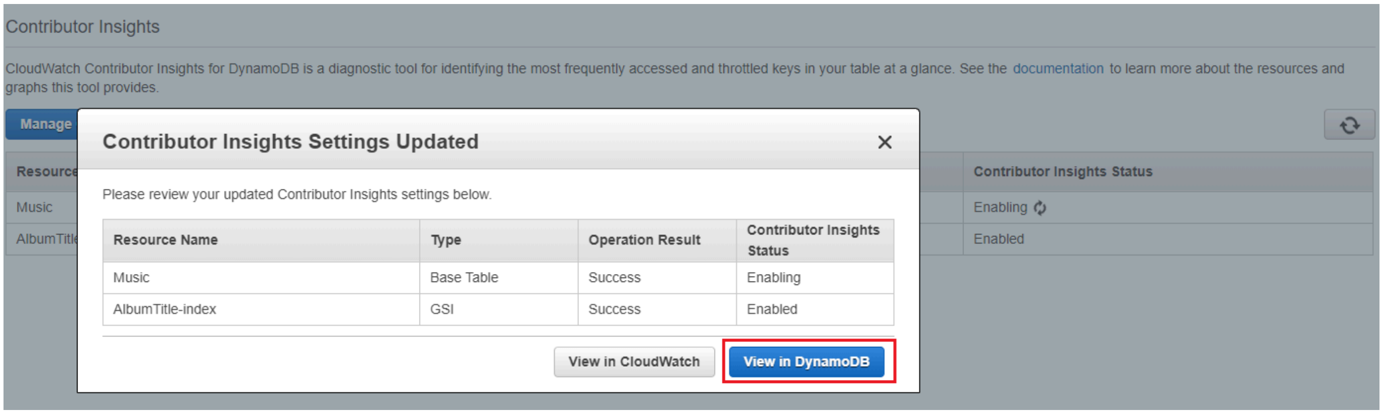
Users who have the appropriate CloudWatch permissions will be able to view primary keys protected by fine-grained access control (FGAC) in Contributor Insight graphs. If the primary key contains FGAC-protected data that you do not want published to CloudWatch, then you should not enable Contributor Insights for this table.

Additional charges will apply by enabling this tool.

Cancel Confirm

このオペレーションが失敗した場合は、Amazon DynamoDB API リファレンスの「DescribeContributorInsights FailureException」を参照して考えられる原因を検討してください。

7. [View in DynamoDB] (DynamoDB で表示) を選択します。



Contributor Insights

CloudWatch Contributor Insights for DynamoDB is a diagnostic tool for identifying the most frequently accessed and throttled keys in your table at a glance. See the documentation to learn more about the resources and graphs this tool provides.

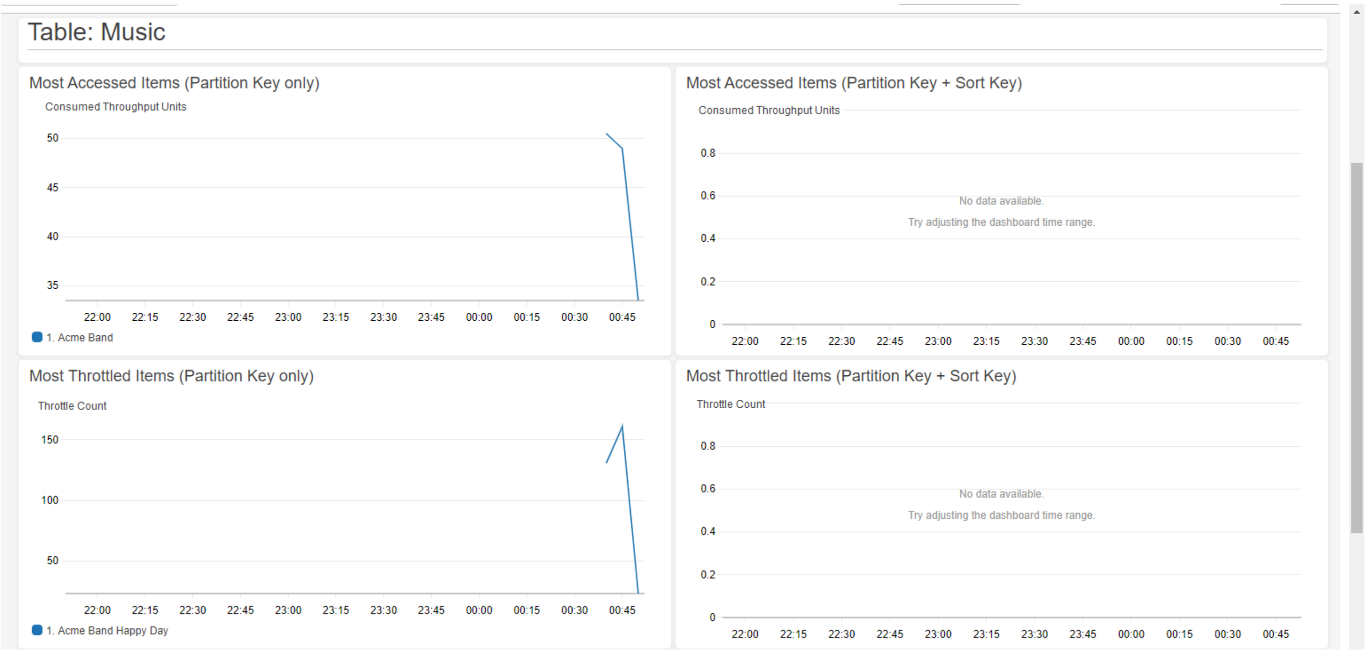
Resource	Contributor Insights Status
Music	Enabling
AlbumTitle	Enabled

Contributor Insights Settings Updated ✕

Please review your updated Contributor Insights settings below.

Resource Name	Type	Operation Result	Contributor Insights Status
Music	Base Table	Success	Enabling
AlbumTitle-index	GSI	Success	Enabled

8. Contributor Insights グラフが `Music` テーブルの [Contributor Insights] (投稿者のインサイト) タブに表示されます。

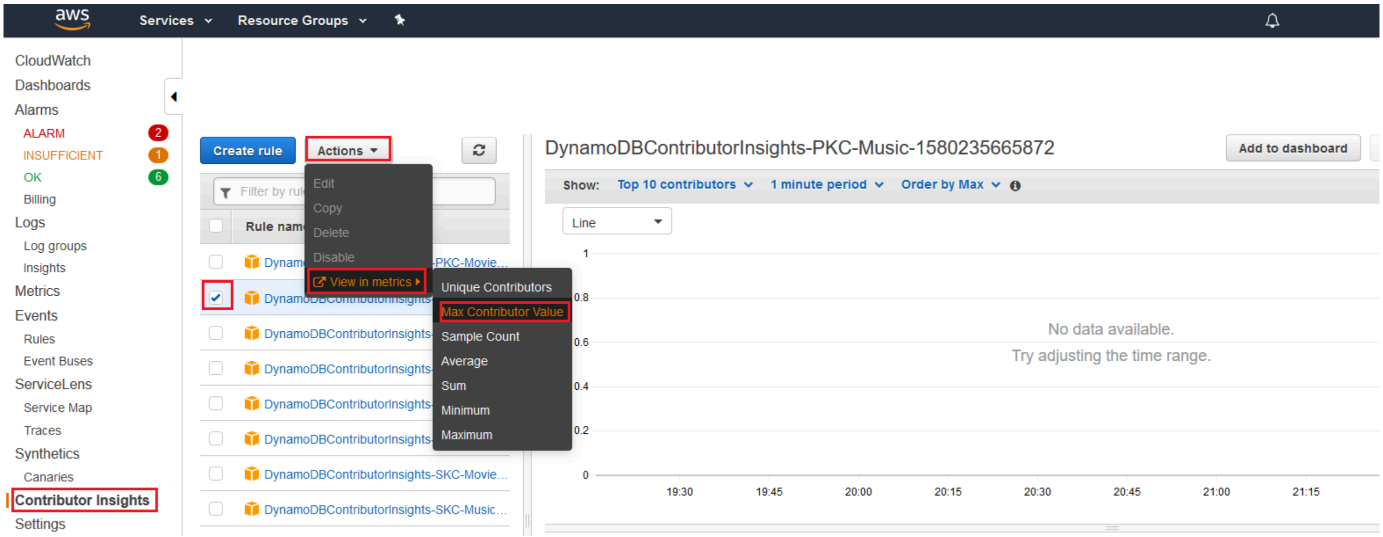


CloudWatch アラームの作成

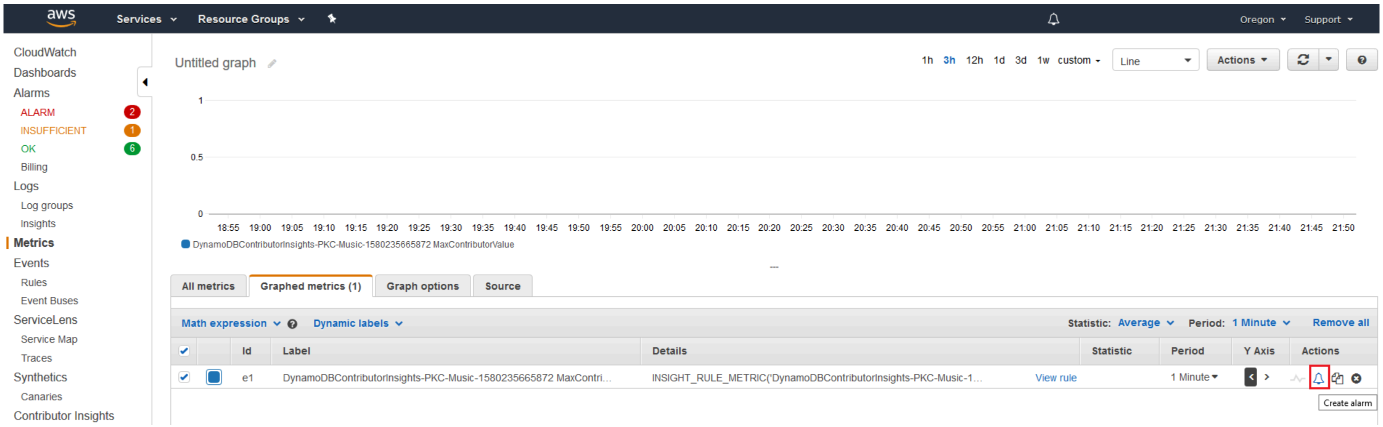
以下のステップに従って、CloudWatch アラームを作成し、パーティションキーが 50,000 件の ConsumedThroughputUnits を超えたときに通知を受け取ります。

1. AWS Management Console にサインインして、CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. コンソールの左側のナビゲーションペインで、[Contributor Insights] を選択します。
3. [DynamoDBContributorInsights-PKC-Music] ルールを選択します。
4. [アクション] ドロップダウンを選択します。
5. [View in metrics (メトリクスで表示)] を選択します。
6. [Max Contributor Value (寄稿者の最大値)] を選択します。

> **Note**
>
> `Max Contributor Value` と `Maximum` のみが有用な統計を返します。このリストの他の統計は、意味のある値を返しません。

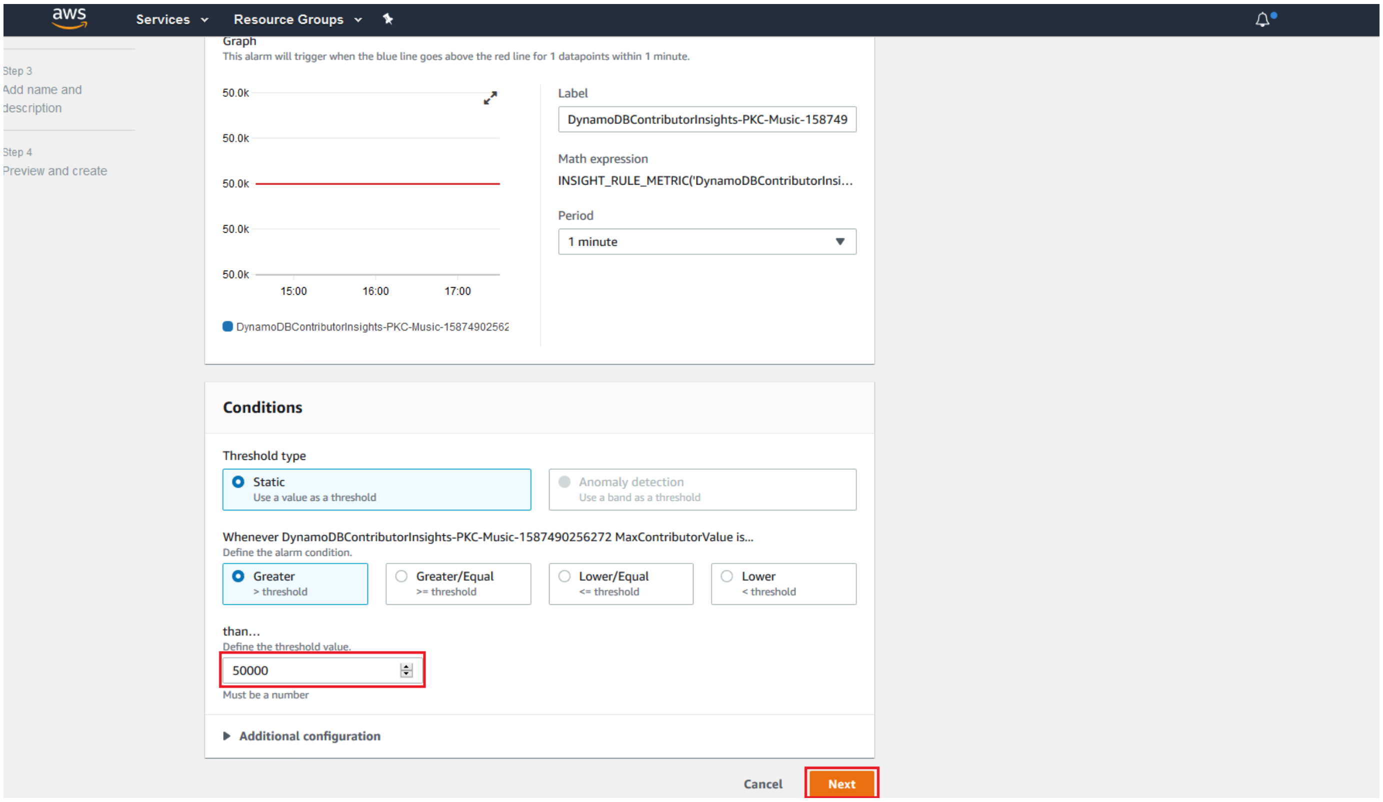


7. [アクション] 列で、[Create Alarm (アラームの作成)] を選択します。



☑		Id	Label	Details		Statistic	Period	Y Axis	Actions
☑	🟦	e1	DynamoDBContributorInsights-PKC-Music-1580235665872 MaxContri...	INSIGHT_RULE_METRIC('DynamoDBContributorInsights-PKC-Music-1...	View rule		1 Minute	< >	

8. [しきい値] として 50000 の値を入力し、[次へ] を選択します。



Graph
This alarm will trigger when the blue line goes above the red line for 1 datapoints within 1 minute.

■ DynamoDBContributorInsights-PKC-Music-15874902562

Label
DynamoDBContributorInsights-PKC-Music-158749

Math expression
INSIGHT_RULE_METRIC('DynamoDBContributorInsi...

Period
1 minute ▼

Conditions

Threshold type
● Static
Use a value as a threshold
○ Anomaly detection
Use a band as a threshold

Whenever DynamoDBContributorInsights-PKC-Music-1587490256272 MaxContributorValue is...
Define the alarm condition.
● Greater
> threshold
○ Greater/Equal
>= threshold
○ Lower/Equal
<= threshold
○ Lower
< threshold

than...
Define the threshold value.
50000
Must be a number

▶ Additional configuration

9. アラームの通知を設定する方法の詳細については、「Amazon CloudWatch アラームの使用」を参照してください。

Contributor Insights の使用 (AWS CLI)

AWS CLI で Contributor Insights を使用するには

1. Music 基本テーブルで CloudWatch Contributor Insights for DynamoDB を有効化します。

```
aws dynamodb update-contributor-insights --table-name Music --contributor-insights-
action=ENABLE
```

2. AlbumTitle-index グローバルセカンダリインデックスで Contributor Insights for DynamoDB を有効化します。

```
aws dynamodb update-contributor-insights --table-name Music --index-name
 AlbumTitle-index --contributor-insights-action=ENABLE
```

3. Music テーブルとそのすべてのインデックスのステータスとルールを取得します。

```
aws dynamodb describe-contributor-insights --table-name Music
```

4. `AlbumTitle-index` グローバルセカンダリインデックスで CloudWatch Contributor Insights for DynamoDB を無効化します。

   ```
   aws dynamodb update-contributor-insights --table-name Music --index-name
    AlbumTitle-index --contributor-insights-action=DISABLE
   ```

5. `Music` テーブルとそのすべてのインデックスのステータスを取得します。

   ```
   aws dynamodb list-contributor-insights --table-name Music
   ```

IAM と CloudWatch Contributor Insights for DynamoDB の使用

Amazon DynamoDB の Amazon CloudWatch Contributor Insights を初めて有効にすると、DynamoDB は AWS Identity and Access Management (IAM) サービスリンクロールを自動的に作成します。このロール (`AWSServiceRoleForDynamoDBCloudWatchContributorInsights`) により、DynamoDB がユーザーに代わって CloudWatch Contributor Insights ルールを管理できます。このサービスリンクロールは削除しないでください。ロールを削除してしまうと、テーブルまたはグローバルセカンダリインデックスを削除したときに、すべてのユーザー管理ルールがクリーンアップされなくなります。

サービスリンクロールの詳細については、「IAM ユーザーガイド」の「サービスリンクロールの使用」を参照してください。

以下のアクセス権限が必要です。

- CloudWatch Contributor Insights for DynamoDB を有効または無効にするには、テーブルまたはインデックスに対する `dynamodb:UpdateContributorInsights` 許可が必要です。
- DynamoDB グラフの CloudWatch Contributor Insights を表示するには、`cloudwatch:GetInsightRuleReport` の権限が必要です。
- 特定の DynamoDB テーブルまたはインデックスに関して、DynamoDB の CloudWatch Contributor Insights を記述するには、`dynamodb:DescribeContributorInsights` 許可が必要です。
- 各テーブルおよびグローバルセカンダリインデックスに関して、DynamoDB ステータスの CloudWatch Contributor Insights を一覧表示するには、`dynamodb:ListContributorInsights` の権限が必要です。

例: DynamoDB の CloudWatch Contributor Insights を有効または無効にする

以下の IAM ポリシーは、DynamoDB の CloudWatch Contributor Insights を有効または無効にするための許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        }
    ]
}
```

KMS キーで暗号化されたテーブルの場合、Contributor Insights を更新するには、ユーザーが kms:Decrypt アクセス許可を持っている必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:kms:*:*:key/*",
            "Action": [
                "kms:Decrypt"
            ],
        }
    ]
}
```

例: CloudWatch Contributor Insights のルールレポートを取得する

以下の IAM ポリシーは、CloudWatch Contributor Insights のルールレポートを取得するための許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

例: リソースに基づいて DynamoDB の CloudWatch Contributor Insights の許可を選択的に適用する

以下の IAM ポリシーは、`ListContributorInsights` と `DescribeContributorInsights` のアクションを許可するアクセス権限を付与し、特定のグローバルセカンダリインデックスに対する `UpdateContributorInsights` アクションを拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListContributorInsights",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/
Author-index"
        }
    ]
}
```

CloudWatch Contributor Insights for DynamoDB のサービスリンクロールの使用

CloudWatch Contributor Insights for DynamoDB は、AWS Identity and Access Management (IAM) サービスリンクロールを使用します。サービスリンクロールは、CloudWatch Contributor Insights for DynamoDB に直接リンクされた一意のタイプの IAM ロールです。サービスリンクロールは、CloudWatch Contributor Insights for DynamoDB で事前定義され、ユーザーの代わりに該当サービスから他の AWS のサービスを呼び出すために必要なすべての許可が含まれされます。

サービスリンクロールを使用することで、必要なアクセス権限を手動で追加する必要がなくなるため、CloudWatch Contributor Insights for DynamoDB の設定が簡単になります。CloudWatch Contributor Insights for DynamoDB は、サービスリンクロールの許可を定義します。特に定義されている場合を除き、DynamoDB の CloudWatch Contributor Insights のみがそのロールを引き受けることができます。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールをサポートする他のサービスについては、「IAM と連携する AWS のサービス」を参照して、[Service-Linked Role] (サービスリンクロール) 列が [Yes] (はい) になっているサービスを探してください。サービスリンクロールに関するドキュメントをサービスで表示するには、[Yes] (はい) リンクを選択します。

CloudWatch Contributor Insights for DynamoDB のサービスリンクロールの許可

CloudWatch Contributor Insights for DynamoDB は、サービスリンクロール AWSServiceRoleForDynamoDBCloudWatchContributorInsights を使用します。サービスリンクロールの目的は、Amazon DynamoDB が DynamoDB テーブルおよびグローバルセカンダリインデックス用に作成された Amazon CloudWatch Contributor Insights ルールをユーザーに代わって管理できるようにすることです。

`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` サービスリンクロールは、ロールの引き受けについて以下のサービスを信頼します。

- `contributorinsights.dynamodb.amazonaws.com`

このロールの許可ポリシーは、CloudWatch Contributor Insights for DynamoDB が指定されたリソースで以下のアクションを実行できるようにします。

- アクション: `DynamoDBContributorInsights` 上で `Create and manage Insight Rules`

サービスにリンクされたロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、権限を設定する必要があります。詳細については、「IAM ユーザーガイド」の「Service-Linked Role Permissions」(サービスリンクロールのアクセス権限) を参照してください。

CloudWatch Contributor Insights for DynamoDB のサービスリンクロールの作成

サービスリンクロールを手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API でコントリビューターインサイトを有効にすると、CloudWatch Contributor Insights for DynamoDB によってサービスリンクロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。Contributor Insights を有効にすると、CloudWatch Contributor Insights for DynamoDB によって、サービスリンクロールが再度作成されます。

CloudWatch Contributor Insights for DynamoDB のサービスリンクロールの編集

CloudWatch Contributor Insights for DynamoDB では、`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` サービスリンクロールを編集することはできません。サービスリンクロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使

用したロールの説明の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

CloudWatch Contributor Insights for DynamoDB のサービスリンクロールの削除

`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` ロールを手動で削除する必要はありません。AWS Management Console、AWS CLI、または AWS API で Contributor Insights を無効にすると、CloudWatch Contributor Insights for DynamoDB によってリソースがクリーンアップされます。

サービスリンクロールは、IAM コンソール、AWS CLI、または AWS API を使用して手動で削除することもできます。そのためにはまず、サービスリンクロールのリソースをクリーンアップする必要があります。その後で、手動で削除できます。

> **Note**
>
> リソースを削除する際に、CloudWatch Contributor Insights for DynamoDB サービスでロールが使用されている場合、削除が失敗することがあります。失敗した場合は、数分待ってから再度オペレーションを実行してください。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、`AWSServiceRoleForDynamoDBCloudWatchContributorInsights` サービスリンクロールを削除します。詳細については、 IAM ユーザーガイド の「サービスにリンクされたロールの削除」を参照してください。

DynamoDB を使用した設計とアーキテクチャの設計に関するベストプラクティス

このセクションでは、Amazon DynamoDB の使用時にパフォーマンスを最大にしてスループットコストを最小にするための推奨事項をすばやく確認することができます。

トピック

DynamoDB 用の NoSQL

Amazon DynamoDB などの NoSQL データベースシステムでは、キーと値のペアやドキュメントストレージなど、データ管理のための代替モデルを使用します。リレーショナルデータベース管理シス

テム (RDBMS) から DynamoDB のような NoSQL データベースシステムに切り替えるときは、主な相違点と特定の設計アプローチを理解することが重要です。

トピック

- リレーショナルデータ設計と NoSQL の相違点
- NoSQL 設計の 2 つの重要な概念
- NoSQL 設計へのアプローチ
- DynamoDB 用の NoSQL Workbench

リレーショナルデータ設計と NoSQL の相違点

リレーショナルデータベースシステム (RDBMS) と NoSQL データベースにはそれぞれ異なる長所と短所があります。

- RDBMS では、データは柔軟にクエリできますが、クエリは比較的コストが高く、トラフィックが多い状況ではスケールがうまくいかない場合があります (DynamoDB でリレーショナルデータをモデル化するための最初のステップ 参照)。
- 一方、DynamoDB のような NoSQL データベースでは、データは限られた数の方法で効率的にクエリできますが、その範囲外では、クエリは高コストで低速になりがちです。

これらの相違点により、2 つのシステム間でデータベース設計が異なるものになります。

- RDBMS では、実装の詳細やパフォーマンスを気にせずに柔軟に設計できます。クエリの最適化は一般的にスキーマ設計には影響しませんが、正規化は重要です。
- DynamoDB では、最も一般的で重要なクエリをできるだけ速く、安価にするために、具体的にスキーマを設計します。データ構造は、ビジネスユースケースの特定の要件に合わせて調整されています。

NoSQL 設計の 2 つの重要な概念

NoSQL 設計では、RDBMS 設計とは異なる考え方が必要です。RDBMS の場合は、アクセスパターンを考慮せずに正規化されたデータモデルを作成できます。その後、新しい課題とクエリの要件が発生したら、そのデータモデルを拡張することができます。各タイプのデータを独自のテーブルに整理できます。

NoSQL の設計の違い

- 対照的に、DynamoDB の場合は答えが必要な質問が分かるまで、スキーマの設計を開始すべきではありません。ビジネス上の問題とアプリケーションのユースケースを理解することが不可欠です。
- DynamoDB アプリケーションで維持するテーブルはできるだけ少なくする必要があります。テーブルの数が少なくなると、スケーラビリティが向上し、必要なアクセス権限の管理が少なくなり、DynamoDB アプリケーションのオーバーヘッドが削減されます。また、バックアップコストを全体的に低く抑えるのにも役立ちます。

NoSQL 設計へのアプローチ

DynamoDB アプリケーションを設計する最初のステップは、システムが満たす必要がある特定のクエリパターンを見極めることです。

始める前に、特にアプリケーションのアクセスパターンの 3 つの基本的な特性を理解することが重要です。

- データサイズ: 一度に格納され、リクエストされるデータの量を把握することで、データパーティションの最も効果的な方法を決定できます。
- データシェイプ: クエリが処理される際 (RDBMS システムのように) データを再形成するのではなく、データベースの形状がクエリ処理に対応するように、NoSQL データベースでデータを整理します。これは、スピードとスケーラビリティを向上させる重要な要素です。
- データ速度: DynamoDB では、クエリを処理するために使用可能な物理パーティションの数を増やし、それらのパーティション間で効率的にデータを分散させることでスケーリングします。ピーク時のクエリのロードを事前に把握することは、I/O 容量を最大限に活用するためにデータを分割する方法を決定する上で役立ちます。

特定のクエリ要件を特定したら、パフォーマンスを管理する一般的な原則に従ってデータを整理できます。

- 関連するデータをまとめてください。 20 年前のルーティングテーブルの最適化に関する研究では、関連するデータをまとめて 1 つの場所にまとめておく「参照の局所性」が、応答時間を短縮する上で最も重要な要素であることがわかりました。これは、今日の NoSQL システムにも同様に当てはまります。関連するデータを近くに置くことはコストとパフォーマンスに大きな影響を与えます。関連するデータ項目を複数のテーブルに分散するのではなく、NoSQL システム内の関連項目を可能な限り近くにまとめる必要があります。

一般的なルールとして、DynamoDB アプリケーションでは維持するテーブルをできるだけ少なくする必要があります。

例外として、大キャパシティの時系列データが必要な場合や、データセットのアクセスパターンが大きく異なる場合などがあります。反転されたインデックスを含む単一のテーブルでは通常、シンプルなクエリを使用してアプリケーションで必要とされる複雑な階層データ構造を構築および取得できます。

- ソート順を使用します。　キーの設計が原因でソートされている場合は、関連項目をグループ化して、効率的にクエリできます。これは重要な NoSQL 設計方法です。
- クエリを分散します。　大キャパシティのクエリがデータベースの一部に集中しないことも重要です。I/O キャパシティを超えるおそれがあります。その代わり、「ホットスポット」を避け、できるだけ多くのパーティションにトラフィックを均等に分散するように、データキーを設計する必要があります。
- グローバルセカンダリインデックスを使用します。　特定のグローバルセカンダリインデックスを作成することで、メインテーブルでサポートできるクエリとは異なるクエリを有効にすることができ、このクエリは高速で比較的安価です。

このような一般原則は、DynamoDB でデータを効率的にモデル化するために使用できる一般的なデザインパターンに変換されます。

DynamoDB 用の NoSQL Workbench

DynamoDB 用の NoSQL Workbench は、最新のデータベース開発および運用で使用できるクロスプラットフォームのクライアント側 GUI アプリケーションです。Windows、macOS、Linux で使用できます。NoSQL Workbench は、DynamoDB テーブルの設計、作成、クエリ、管理に役立つデータモデリング、データ可視化、サンプルデータ生成、クエリ開発といった特徴を提供する視覚的開発ツールです。DynamoDB 用の NoSQL Workbench を使用すると、アプリケーションのデータアクセスパターンを満たす既存のデータモデルから新しいデータモデルを構築したり、既存のデータモデルに基づいてモデルを設計したりできます。プロセスの最後に、設計されたデータモデルをインポートおよびエクスポートすることもできます。詳細については、「NoSQL Workbench を使用したデータモデルの構築」を参照してください。

削除保護を使用してテーブルを保護する

削除保護を使用すると、テーブルが誤って削除されるのを防ぐことができます。このセクションでは、削除保護を使用するためのいくつかのベストプラクティスについて説明します。

- すべての有効な本番環境のテーブルでは、削除保護設定をオンにして、これらのテーブルが誤って削除されないように保護するのがベストプラクティスです。これはグローバルレプリカにも当てはまります。
- アプリケーション開発ユースケースを提供するときに、テーブル管理ワークフローに開発テーブルとステージングテーブルの頻繁な削除と再作成が含まれる場合は、削除保護設定をオフにできます。これにより、権限のある IAM プリンシパルは、そうしたテーブルを意図的に削除できるようになります。

削除に対する保護についての詳細は、「削除保護の使用」を参照してください。

DynamoDB の Well-Architected レンズを使用して DynamoDB ワークロードを最適化する

このセクションでは、優れたアーキテクチャの DynamoDB ワークロードを設計するための設計原則とガイダンスの集合である Amazon DynamoDB Well-Architected レンズについて説明します。

DynamoDB テーブルのコストの最適化

このセクションでは、既存の DynamoDB テーブルのコストを最適化する方法のベストプラクティスについて説明します。次の戦略を見て、どのコスト最適化戦略がニーズに最も適しているかを確認し、それらに繰り返しアプローチする必要があります。各戦略には、コストに影響する可能性のある要素、注意が必要な兆候、およびその削減方法に関する規範的なガイダンスの概要が示されています。

トピック

テーブルレベルでコストを評価する

AWS Management Console 内にある Cost Explorer ツールを使用すると、読み取り、書き込み、ストレージ、バックアップの料金など、コストをタイプ別に分類して確認できます。また、これらのコストを月や日などの期間別にまとめて表示することもできます。

管理者が直面する課題として、ある特定のテーブルのコストのみを確認する必要がある場合があります。一部のデータは DynamoDB コンソールまたは `DescribeTable` API の呼び出しを介して取得することができます。ただし Cost Explorer では、デフォルトでは特定のテーブルに関連するコストでフィルタリングまたはグループ化することができません。このセクションでは、タグ付けを使用して Cost Explorer で個々のテーブルのコスト分析を実行する方法を説明します。

トピック

単一の DynamoDB テーブルのコストを表示する方法

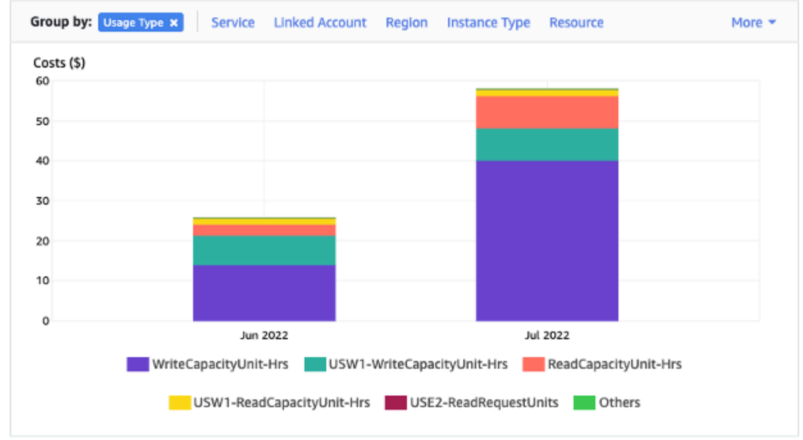
Amazon DynamoDB AWS Management Console と`DescribeTable` API はどちらも、プライマリキースキーマ、テーブルのすべてのインデックス、テーブルとインデックスのサイズと項目数など、単一のテーブルに関する情報を表示します。テーブルのサイズとインデックスのサイズから、テーブルの月々のストレージコストを計算することができます。たとえば、us-east-1 リージョンでは 1 GB あたり 0.25 USD です。

テーブルがプロビジョンドキャパシティーモードの場合、現在の RCU と WCU の設定も返されます。これらを使用してテーブルの現在の読み取りと書き込みのコストを計算することもできますが、特にテーブルが Auto Scaling で設定されている場合、これらのコストは変わる可能性があります。

> Note
>
> テーブルがオンデマンドキャパシティーモードの場合、`DescribeTable` はスループットコストの見積もりには役立ちません。これはスループットコストが、ある期間内のプロビジョニングされた使用量ではなく、実際の使用量に基づいて請求されるためです。

Cost Explorer のデフォルトビュー

Cost Explorer のデフォルトビューには、スループットやストレージなどの消費されたリソースのコストを示すグラフが表示されます。月ごとや日ごとの合計など、期間ごとにコストをグループ化することを選択できます。ストレージ、読み取り、書き込み、その他の機能のコストも分類して比較できます。



Cost Explorer でのテーブルタグの使用方法および適用方法

デフォルトでは、Cost Explorer は複数のテーブルのコストを合計にまとめるため、特定の 1 つのテーブルのコストの概要は表示されません。ただし、AWS リソースへのタグ付けを使用すると、メタデータタグで各テーブルを識別できます。タグとは、プロジェクトや部門に属するすべてのリソースを識別するなど、さまざまな目的に使用できるキーと値のペアです。この例では、MyTable という名前のテーブルがあると仮定します。

1. table_name のキーと MyTable の値を使用してタグを設定します。
2. Cost Explorer 内でタグをアクティブ化し、タグ値でフィルタリングすると、各テーブルのコストをより明確に把握できます。

> **Note**
>
> タグが Cost Explorer に表示され始めるまでには、1～2 日かかる場合があります。

メタデータタグは、コンソールで自分で設定することも、AWS CLI やAWS SDK などのオートメーションを使用して設定することもできます。組織の新しいテーブル作成プロセスの一環として、table_name タグの設定を義務付けることを検討してください。既存のテーブルについては、これらのタグを検索してアカウントの特定のリージョンにあるすべての既存のテーブルに適用す

る Python ユーティリティが用意されています。詳細については、GitHub の「Eponymous Table Tagger」を参照してください。

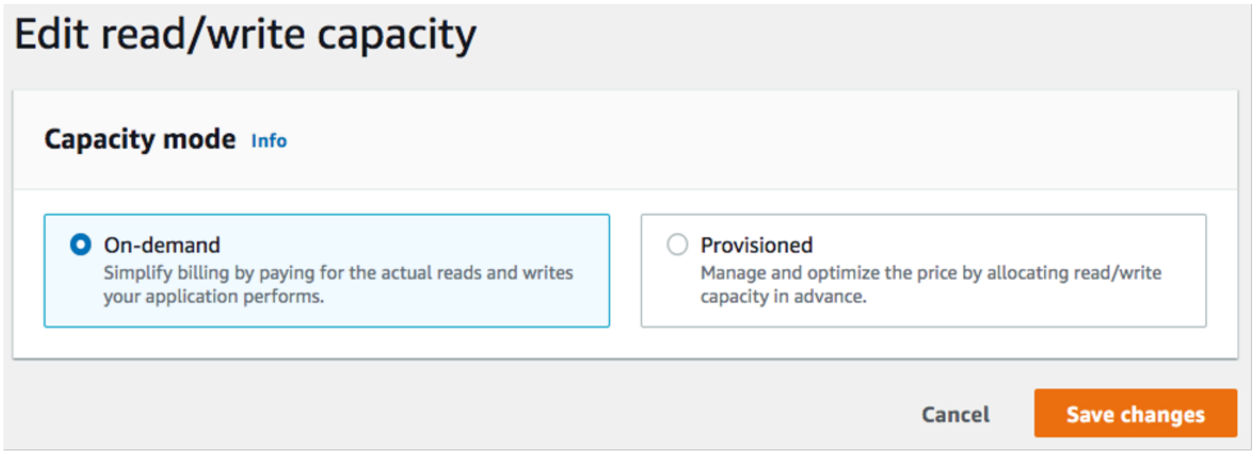
テーブルキャパシティモードの評価

このセクションでは、DynamoDB テーブルで、適切なキャパシティモードを選択する方法の概要を説明します。各モードは、スループットの変化に対する応答性や使用に対する請求方法という点で、さまざまなワークロードのニーズを満たすように調整されています。決定を下す際には、これらの要素のバランスを取る必要があります。

トピック

利用可能なテーブルキャパシティモード

DynamoDB テーブルを作成する場合、オンデマンドまたはプロビジョンドキャパシティモードのいずれかを選択する必要があります。24 時間ごとに 1 回、読み込み/書き込みキャパシティーモードを切り替えることができます。唯一の例外は、プロビジョンドモードテーブルをオンデマンドモードに切り替えた場合、同じ 24 時間以内にプロビジョニングモードに戻すことができることです。



Edit read/write capacity

Capacity mode Info

● On-demand
Simplify billing by paying for the actual reads and writes your application performs.

○ Provisioned
Manage and optimize the price by allocating read/write capacity in advance.

オンデマンドキャパシティモード

オンデマンドキャパシティモードは、DynamoDB テーブルのキャパシティをプランニングまたはプロビジョニングする必要がないように設計されています。このモードでは、テーブルへのリクエスト

にテーブルが即座に対応します。リソースをスケールアップまたはスケールダウンする必要はありません (テーブルの以前のピークスループットの最大 2 倍まで)。

オンデマンドテーブルは、テーブルに対する実際のリクエスト数をカウントして請求されるため、お支払いいただくのは、プロビジョニングされたものではなく、実際に使用した分のみです。

プロビジョンドキャパシティモード

プロビジョンドキャパシティモードは従来型のモデルで、リクエストに対して利用できるテーブルのキャパシティを直接、または自動スケーリングを使って定義できます。特定のキャパシティが指定された時間にテーブルにプロビジョニングされるため、請求はリクエスト数ではなく、プロビジョニングされたキャパシティに基づいて行われます。割り当てられたキャパシティを超えると、テーブルがリクエストを拒否し、アプリケーションユーザーのエクスペリエンスが低下する可能性があります。

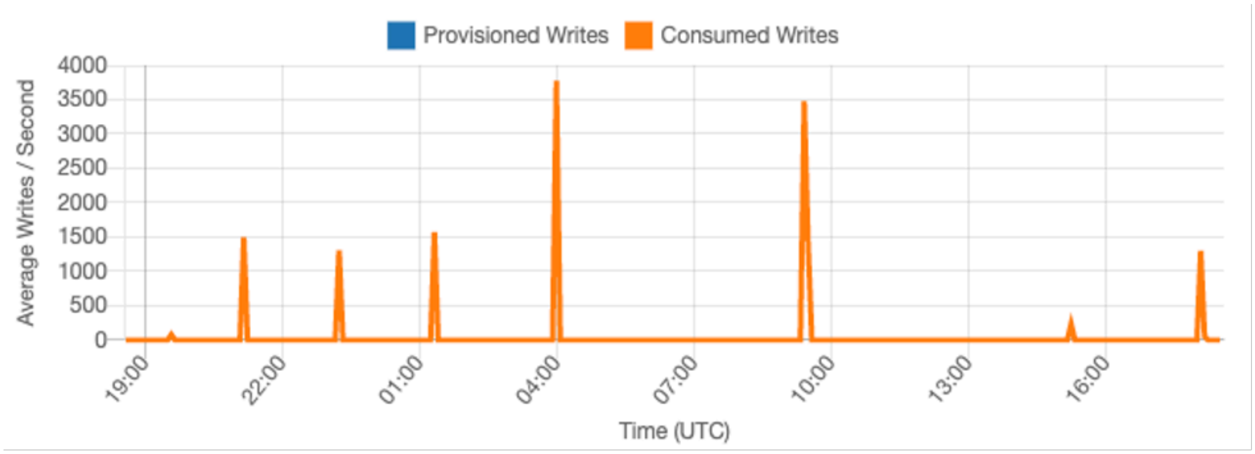
プロビジョンドキャパシティモードでは、スロットリングを低く抑え、コストを調整するために、テーブルのオーバープロビジョニングを避ける、あるいはプロビジョニングを不足させないというバランスが必要になります。

オンデマンドキャパシティモードを選択する場合

コストを最適化する際、次のグラフのようなワークロードを実行する場合は、オンデマンドモードが最適です。

この種のワークロードには、次の要素が影響します。

- リクエストのタイミングが予測できない (トラフィックの急増につながる)
- リクエストの量が変動する (バッチワークロードに起因)
- 一定時間、ゼロまで低下する、またはピーク時の 18% を下回る (開発環境またはテスト環境に起因)



上記の要素があるワークロードでは、トラフィックの急増に対応するために自動スケーリングを使用して十分なキャパシティをテーブルに維持しようとすると、テーブルが過剰にプロビジョニングされて必要以上にコストがかかったり、テーブルがプロビジョニング不足でリクエストが必要以上にスロットリングされたりする可能性があります。

オンデマンドテーブルは、読み込みおよび書き込みリクエストの料金が従量制であるため、使用した分だけを支払います。これにより、コストとパフォーマンスのバランスを簡単に取ることができます。オプションで、個々のオンデマンドテーブルとグローバルセカンダリインデックスに対して 1 秒あたりの読み込みや書き込み (または両方) の最大スループットを設定して、コストと使用量を制限し続けることもできます。詳細については、「オンデマンドテーブルの最大スループット」を参照してください。オンデマンドテーブルを定期的に評価して、ワークロードに上記の要素がいまだに残っていることを確認する必要があります。ワークロードが安定したら、コストをさらに最適化するためにプロビジョンドモードに変更することを検討してください。

プロビジョンドキャパシティモードを選択する場合

プロビジョンドキャパシティモードに最適なワークロードは、以下のグラフのように、使用パターンがより予測可能な場合です。

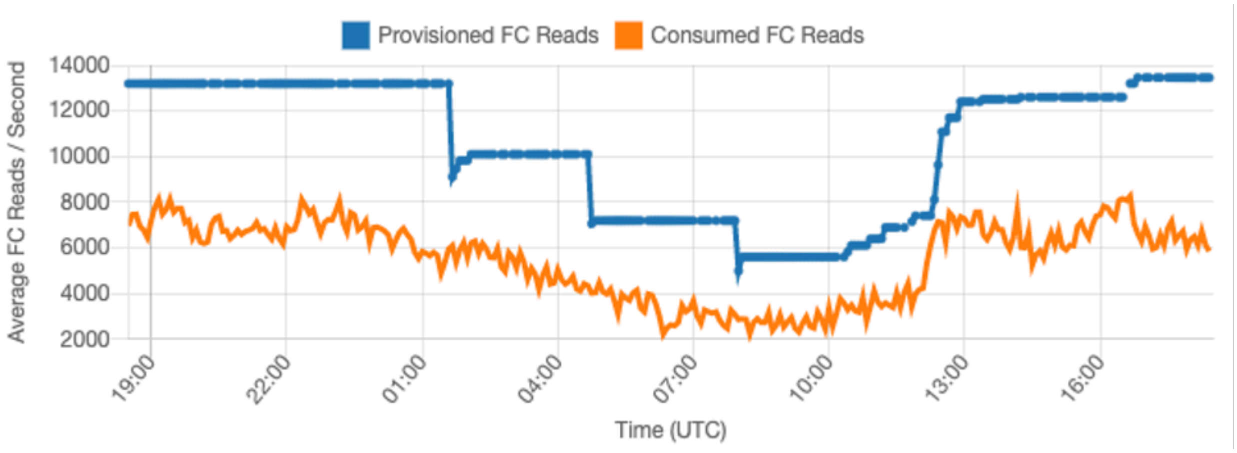
Note

プロビジョンドキャパシティでアクションを実行する前に、14 日や 24 時間などの短い期間でメトリクスを確認することをお勧めします。

この種のワークロードには、次の要素が影響します。

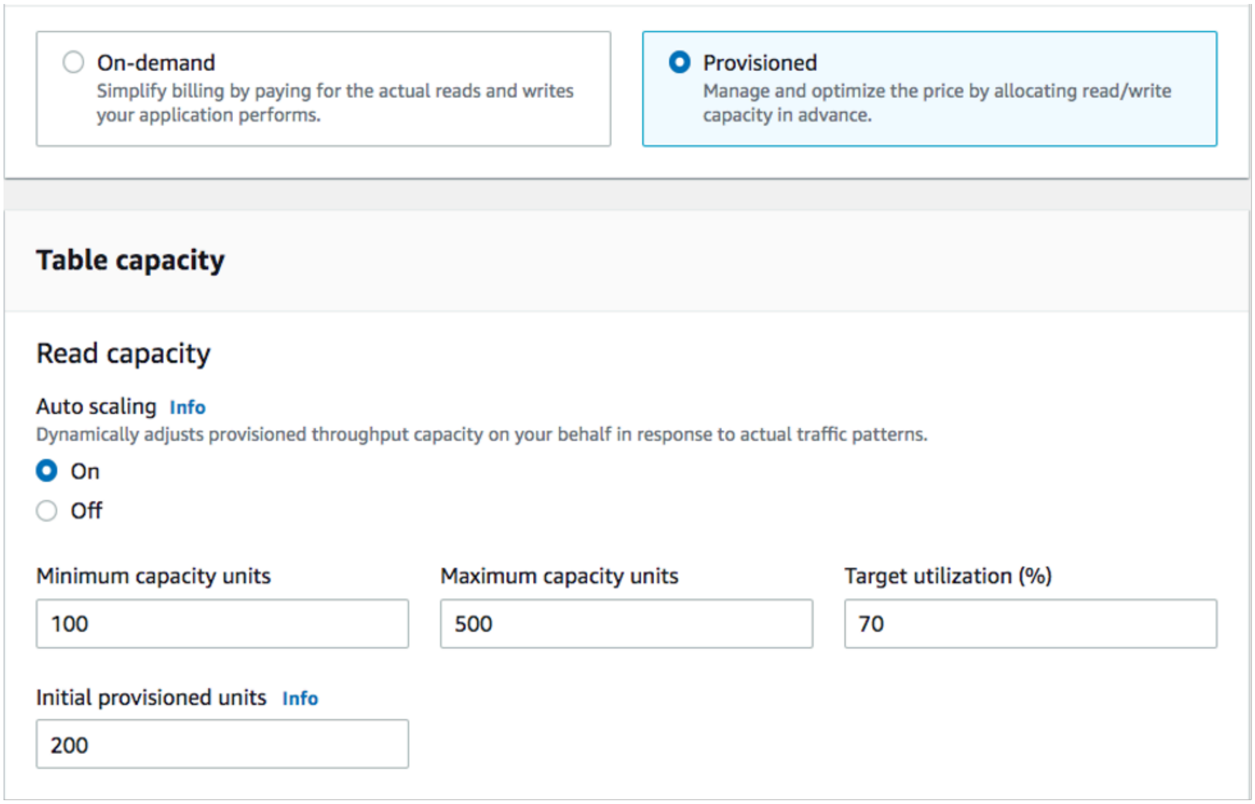
- 特定の時間または 1 日のトラフィックが予測可能/周期的

- トラフィックの急増が短期間で限定的



一定時間内または 1 日のトラフィック量が安定しているため、テーブルのプロビジョンドキャパシティは、テーブルで実際に消費されるキャパシティに比較的近い値に設定できます。プロビジョンドキャパシティテーブルのコストを最適化することは、最終的には、テーブル上で `ThrottledRequests` を増やすことなく、プロビジョンドキャパシティ (青色の線) を消費キャパシティ (オレンジ色の線) にできるだけ近づけるための練習になります。2 つの線の間にある空白は、未使用のキャパシティであると同時に、スロットリングによるユーザーエクスペリエンスの低下に備えた保険でもあります。

DynamoDB は、Auto Scaling を提供しており、ユーザーに代わって自動的にプロビジョンドキャパシティテーブルのバランスを調整します。これにより、1 日を通して消費されたキャパシティを追跡し、いくつかの変数に基づいてテーブルのキャパシティを設定できます。



○ On-demand
Simplify billing by paying for the actual reads and writes your application performs.

● Provisioned
Manage and optimize the price by allocating read/write capacity in advance.

Table capacity

Read capacity

Auto scaling Info
Dynamically adjusts provisioned throughput capacity on your behalf in response to actual traffic patterns.

● On
○ Off

Minimum capacity units: 100

Maximum capacity units: 500

Target utilization (%): 70

Initial provisioned units Info: 200

キャパシティユニットの最小数

テーブルの最小キャパシティを設定してスロットリングを制限することはできますが、これによってテーブルのコストが削減されるわけではありません。テーブルの使用量が少ない期間が続いた後に突然使用量が急増した場合、最小値を設定しておくと、Auto Scaling がテーブルキャパシティを低く設定しすぎるのを防ぐことができます。

キャパシティユニットの最大数

テーブルの最大キャパシティを設定すると、テーブルが意図したより大きくスケールしてしまうことを制限できます。大規模な負荷テストを実施しない開発テーブルまたはテストテーブルには最大値を適用することを検討してください。最大数はどのテーブルにも設定できますが、本番環境で使用する場合は、誤ってスロットリングされないように、この設定をテーブルのベースラインと照らし合わせて定期的に評価してください。

目標使用率

テーブルの目標使用率を設定することは、プロビジョンドキャパシティテーブルのコストを最適化するための主な手段です。ここでパーセント値を低く設定すると、テーブルでオーバープロビジョニングされるキャパシティが増え、コストが増加しますが、スロットリングのリスクは軽減されます。

パーセント値を高く設定すると、テーブルでオーバープロビジョニングされるキャパシティは減少しますが、スロットリングのリスクは増大します。

テーブルキャパシティモードを選択する際に考慮すべきその他の要素

2 つのモードのどちらかを決める際には、考慮すべき点が他にもいくつかあります。

リザーブドキャパシティ

プロビジョンドキャパシティテーブルの場合、DynamoDB では読み込みおよび書き込みキャパシティ用にリザーブドキャパシティを購入できます (レプリケートされた書き込みキャパシティユニット (rWCU) と Standard-IA テーブルは現在対象外です)。このキャパシティの予約を購入すると、テーブルのコストを大幅に削減できます。

2 つのテーブルモードのどちらかを決める際には、この追加の割引がテーブルのコストに与える影響を考慮してください。多くの場合、比較的予測が困難なワークロードでも、リザーブドキャパシティを利用してオーバープロビジョニングされたプロビジョンドキャパシティテーブルで実行する方が安く済む場合があります。

ワークロードの予測可能性の向上

状況によっては、ワークロードに予測可能なパターンと予測不可能なパターンの両方があるように見えることがあります。これはオンデマンドテーブルで簡単にサポートできますが、ワークロードの予測不可能なパターンを改善できれば、コストも改善できる可能性があります。

これらのパターンの最も一般的な原因の 1 つは、バッチインポートです。このタイプのトラフィックでは、ワークロードを実行するとスロットリングが発生してしまうほど、テーブルのベースラインキャパシティを超えることがよくあります。このようなワークロードをプロビジョンドキャパシティテーブルで実行し続ける場合は、次のオプションを検討してください。

- スケジュールされた時間にバッチが行われる場合は、バッチの実行前に Auto Scaling のキャパシティを増やすようにスケジュールできます。
- バッチがランダムに行われる場合は、できるだけ速く実行するよりも、実行する時間を延長することを検討してください。
- 自動スケーリングでテーブルキャパシティの調整を開始できるようになるまで、インポートを低速で開始し、数分間かけてゆっくりと速度を上げていくランプアップ時間をインポートに追加します。

テーブルの Auto Scaling 設定を評価する

このセクションでは、DynamoDB テーブルの Auto Scaling 設定を評価する方法の概要を説明します。Amazon DynamoDB Auto Scaling は、アプリケーショントラフィックとターゲット使用率メトリクスに基づいてテーブルとグローバルセカンダリインデックス (GSI) のスループットを管理する機能です。これにより、テーブルや GSI がアプリケーションパターンに必要なキャパシティを確保できるようになります。

AWS Auto Scaling サービスは現在のテーブル使用率をモニタリングし、ターゲット使用率値である `TargetValue` と比較します。割り当てられたキャパシティを増減する時期になると、通知されます。

トピック

Auto Scaling 設定について

ターゲット使用率、初期ステップ、最終値の正しい値を定義するには、運用チームの関与が必要です。これにより、AWS Auauto Scaling ポリシーのトリガーに使用されるアプリケーションの使用履歴に基づいて値を適切に定義することができます。ターゲット使用率は、Auto Scaling ルールが適用される前の期間に達成する必要がある合計キャパシティの割合です。

高いターゲット使用率 (90% 前後) を設定すると、Auto Scaling が起動する前に、一定期間トラフィックが 90% を超える必要があります。アプリケーションが常に一定で、トラフィックが急増しない場合を除いて、高いターゲット使用率を使用しないでください。

非常に低い使用率 (50% 未満) を設定する場合は、アプリケーションが Auto Scaling ポリシーをトリガーする前に、プロビジョニングされたキャパシティの 50% に達する必要があります。アプリケーショントラフィックが非常に速い速度で増加しない限り、これは通常、キャパシティの未使用およびリソースの浪費につながります。

目標使用率が低い (50% 未満）テーブルを特定する方法

AWS CLI または AWS Management Console を使用して、DynamoDB リソース内の Auto Scaling ポリシーの `TargetValues` をモニタリングおよび識別できます。

AWS CLI

1. リソースのリスト全体を返すには、次のコマンドを実行します。

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb
   ```

   このコマンドは、任意の DynamoDB リソースに発行された Auto Scaling ポリシーのリスト全体を返します。特定のテーブルからのみリソースを取得する場合は、`–resource-id parameter` を追加できます。例:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>”
   ```

2. 特定の GSI の Auto Scaling ポリシーのみを返すには、次のコマンドを実行します。

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
   ```

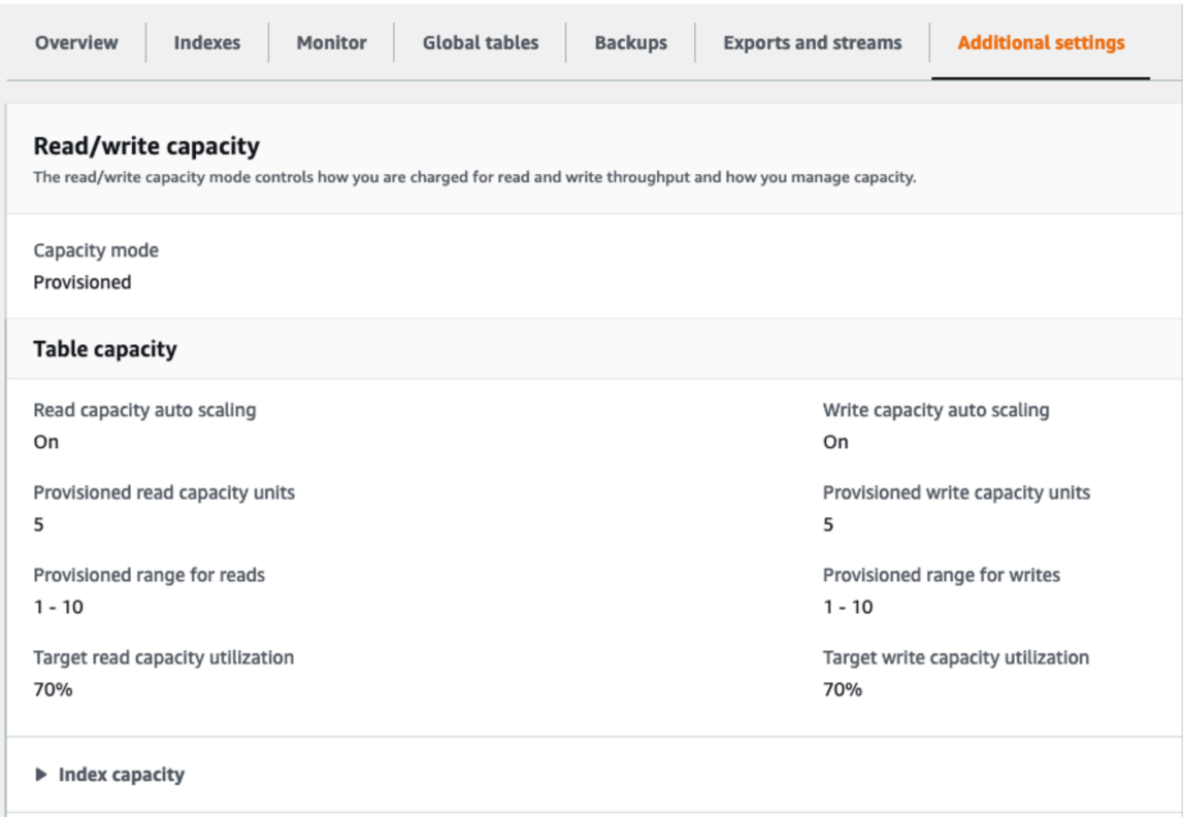
   Auto Scaling ポリシーで注目している値は以下のとおりです。オーバープロビジョニングを避けるため、ターゲット値を 50% 以上に設定したいと考えています。次のような結果を取得します。

   ```
   {
       "ScalingPolicies": [
           {
               "PolicyARN": "arn:aws:autoscaling:<region>:<account-
   id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
   name>:policyName/$<full-gsi-name>-scaling-policy",
               "PolicyName": $<full-gsi-name>”,
               "ServiceNamespace": "dynamodb",
               "ResourceId": "table/<table-name>/index/<index-name>",
               "ScalableDimension": "dynamodb:index:WriteCapacityUnits",
               "PolicyType": "TargetTrackingScaling",
               "TargetTrackingScalingPolicyConfiguration": {
                   "TargetValue": 70.0,
                   "PredefinedMetricSpecification": {
                       "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
                   }
               },
               "Alarms": [
   ```

```
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName":$<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:ReadCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBReadCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

AWS Management Console

1. AWS Management Console にログインし、「AWS Management Console の開始方法」の CloudWatch サービスページに移動します。必要に応じて、適切な AWS リージョンを選択します。
2. 左のナビゲーションバーで、[Tables] (テーブル) を選択します。[Tables] (テーブル) ページで、テーブルの [Name] (名) を選択します。
3. [Additional Settings] (追加の設定) タブの [Table Details] (テーブルの詳細) ページで、テーブルの Auto Scaling 設定を確認します。



Read/write capacity

The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.

Capacity mode
Provisioned

Table capacity

Read capacity auto scaling
On

Provisioned read capacity units
5

Provisioned range for reads
1 - 10

Target read capacity utilization
70%

Write capacity auto scaling
On

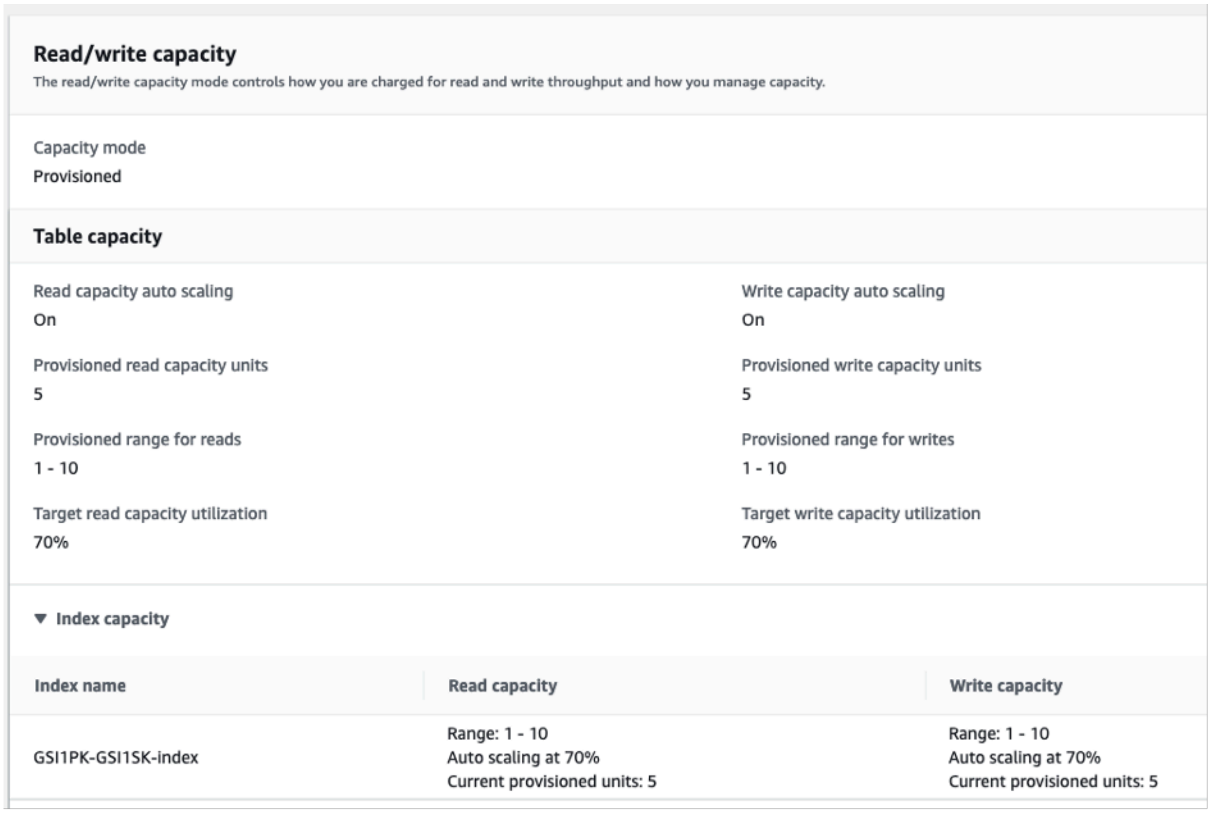
Provisioned write capacity units
5

Provisioned range for writes
1 - 10

Target write capacity utilization
70%

▶ Index capacity

インデックスの場合は、[Index Capacity] (インデックスキャパシティ) タブを展開して、インデックスの Auto Scaling 設定を確認します。



Read/write capacity

The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity.

Capacity mode
Provisioned

Table capacity

Read capacity auto scaling
On

Provisioned read capacity units
5

Provisioned range for reads
1 - 10

Target read capacity utilization
70%

Write capacity auto scaling
On

Provisioned write capacity units
5

Provisioned range for writes
1 - 10

Target write capacity utilization
70%

▼ Index capacity

Index name	Read capacity	Write capacity
GSI1PK-GSI1SK-index	Range: 1 - 10 Auto scaling at 70% Current provisioned units: 5	Range: 1 - 10 Auto scaling at 70% Current provisioned units: 5

ターゲット使用率が 50% 以下の場合は、テーブルの使用率メトリクスを調べて、プロビジョニングが不十分か過剰かを確認する必要があります。

季節変動のあるワークロードに対処する方法

次のシナリオを考えてみましょう。アプリケーションはほとんどの場合最小平均値で動作していますが、目標使用率は低いため、アプリケーションは 1 日の特定の時間に発生するイベントに迅速に対応でき、十分なキャパシティがあり、スロットリングを回避できます。これは、アプリケーションが通常の勤務時間 (午前 9 時 ~ 午後 5 時) には非常にビジーで、勤務時間外には基本的なレベルで動作する場合に一般的なシナリオです。一部のユーザーは午前 9 時前に接続を開始するため、アプリケーションはこの低いしきい値を使用して、ピーク時に必要なキャパシティに達するようにすばやく拡張します。

このシナリオは次のようなものです。

- 午後 5 時から午前 9 時までの間、`ConsumedWriteCapacity` ユニットの単位は 90 から 100 の間にとどまる
- ユーザーが午前 9 時前にアプリケーションに接続し始めると、キャパシティユニットが大幅に増加する (最大値は 1500 WCU)

- 勤務時間中のアプリケーション使用量は、平均して 800～1,200 の間で推移する

前述のシナリオが当てはまる場合は、スケジュールされた Auto Scaling の使用を検討してください。この場合、テーブルにはアプリケーションの Auto Scaling ルールを設定できますが、ターゲット使用率はそれほど高くなく、必要な間隔で追加のキャパシティのみをプロビジョニングできます。

AWS CLI を使用して以下の手順を実行し、時間帯と曜日に基づいて実行されるスケジュールされた Auto Scaling ルールを作成することができます。

1. DynamoDB テーブルまたは GSI をスケーラブルなターゲットとして Application Auto Scaling に登録します。スケーラブルなターゲットは、Application Auto Scaling でスケールアウトまたはスケールインできるリソースです。

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --min-capacity 90 \
    --max-capacity 1500
```

2. 要件に従ってスケジュールされたアクションをセットアップします。

   このシナリオに対応するには、2 つのルールが必要です。1 つはスケールアップ用、もう 1 つはスケールダウン用です。スケジュールされたアクションをスケールアップするための最初のルール:

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-8-5-scheduled-action \
    --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
    --schedule "cron(45 8 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

   スケジュールされたアクションをスケールダウンするための 2 つ目のルール:

```
aws application-autoscaling put-scheduled-action \
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

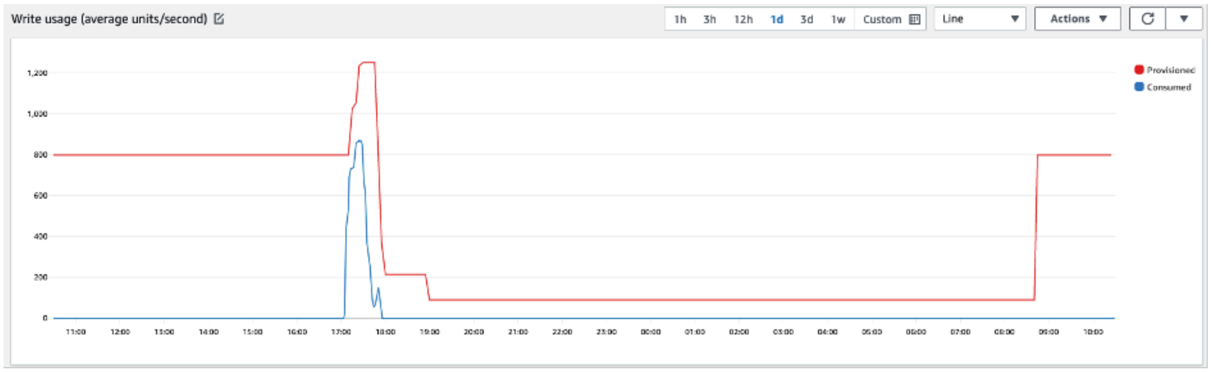
3. 次のコマンドを実行して、両方のルールがアクティブになっていることを確認します。

```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

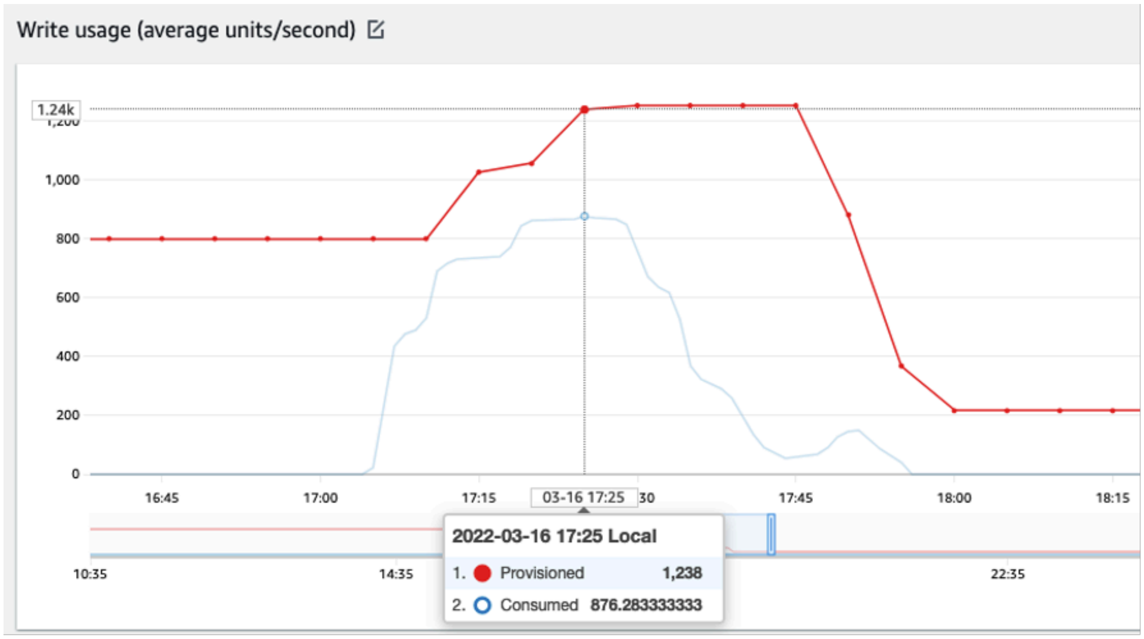
次のような結果が表示されます。

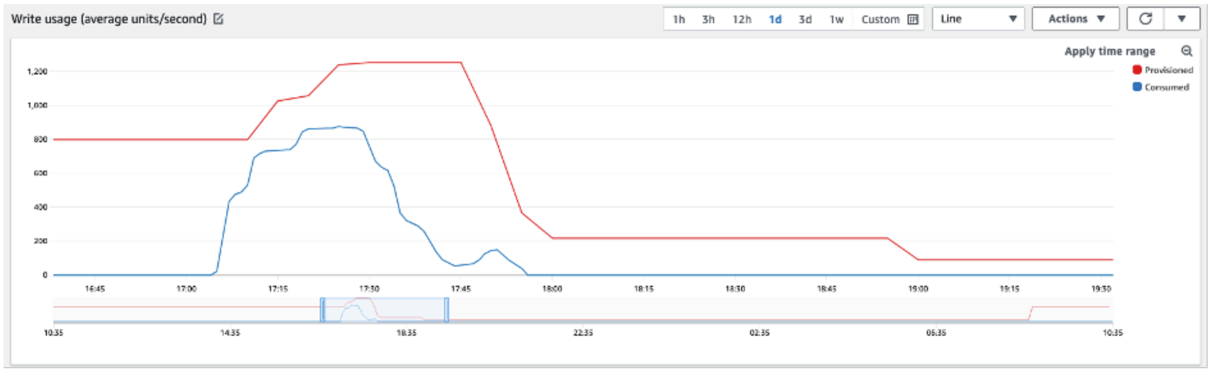
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

次の図は、常に 70% のターゲット使用率を維持するサンプルワークロードを示しています。Auto Scaling ルールが適用され、スループットが低下していないことに注目してください。



拡大すると、アプリケーションにスパイクが発生して Auto Scaling のしきい値が 70% に達し、Auto Scaling が強制的に開始され、テーブルに必要な追加キャパシティが提供されたことがわかります。スケジュールされた Auto Scaling アクションは最大値と最小値に影響するので、設定するのはユーザーの責任です。





パターンが不明な急増するワークロードに対処する方法

このシナリオでは、アプリケーションのパターンがまだわからないため、アプリケーションのターゲット使用率が非常に低く、ワークロードがスロットリングされないようにする必要があります。

代わりにオンデマンドキャパシティモードを使用することを検討してください。オンデマンドテーブルは、トラフィックパターンが不明で急増するワークロードに最適です。オンデマンドキャパシティモードでは、アプリケーションがテーブルに対して実行するデータの読み取りと書き込みに対して、リクエストごとに料金を支払います。DynamoDB はワークロードの増減に即座に対応するため、アプリケーションに期待する読み取りと書き込みのスループットを指定する必要はありません。

リンクされたアプリケーションのワークロードに対応する方法

このシナリオでは、アプリケーションは他のシステムに依存します。例えば、バッチ処理シナリオでは、アプリケーションロジックのイベントに応じてトラフィックが大幅に急増する可能性があります。

特定のニーズに応じてテーブルのキャパシティと `TargetValues` を増やすことができる、これらのイベントに反応するカスタムの Auto Scaling ロジックの開発を検討してください。Amazon EventBridge のメリットを得たり、Lambda や Step Functions のような AWS を組み合わせることで、特定のアプリケーションのニーズに対応することができます。

テーブルクラスの選択を評価する

このセクションでは、DynamoDB テーブルで、適切なテーブルクラスを選択する方法の概要を説明します。Standard-Infrequent Access (Standard-IA) テーブルクラスのリリースにより、テーブルを最適化してストレージコストまたはスループットコストを削減できるようになりました。

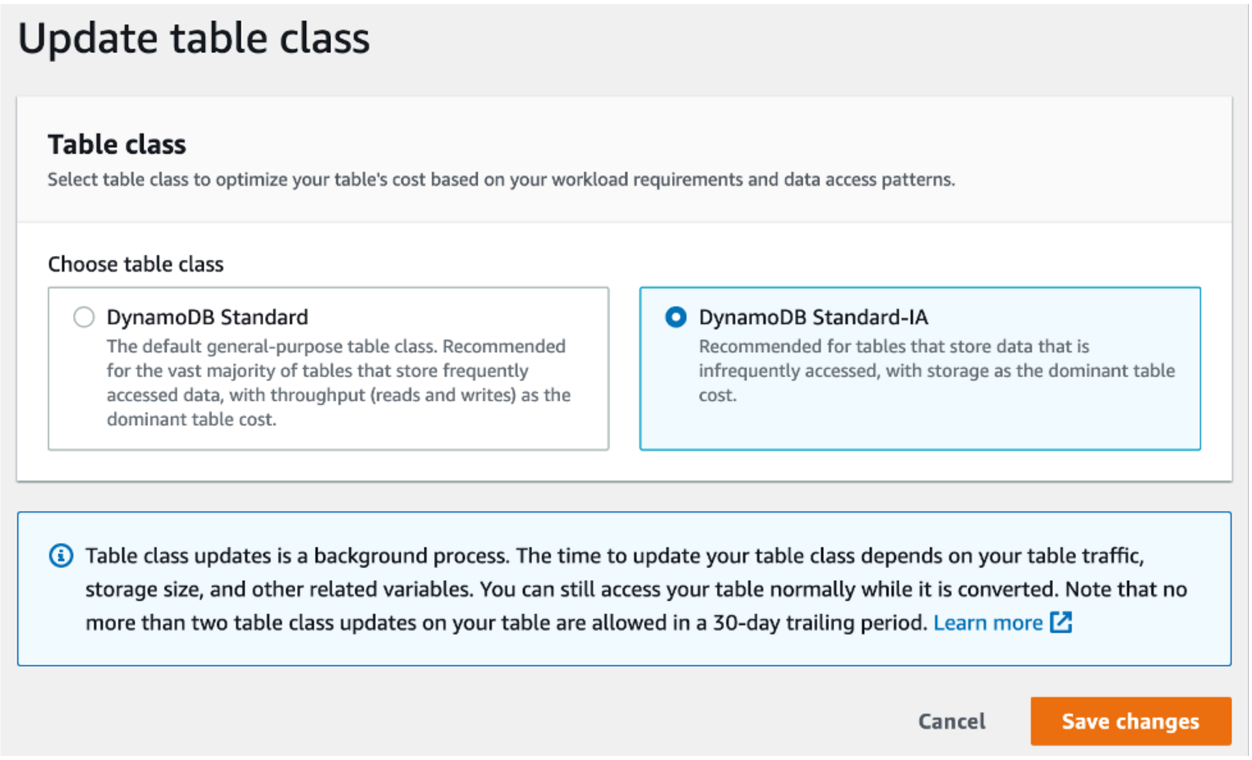
トピック

- 利用可能なテーブルクラス

- DynamoDB Standard テーブルクラスを選択する場合
- DynamoDB Standard-IA テーブルクラスを選択する場合
- テーブルクラスを選択する際に考慮すべきその他の要素

利用可能なテーブルクラス

DynamoDB テーブルを作成する場合は、テーブルクラスに DynamoDB Standard または DynamoDB Standard-IA のいずれかを選択する必要があります。テーブルクラスは 30 日間に 2 回変更できるため、将来いつでも変更できます。どちらのテーブルクラスを選択しても、テーブルのパフォーマンス、可用性、信頼性、耐久性には影響しません。



Update table class

Table class

Select table class to optimize your table's cost based on your workload requirements and data access patterns.

Choose table class

○ DynamoDB Standard
The default general-purpose table class. Recommended for the vast majority of tables that store frequently accessed data, with throughput (reads and writes) as the dominant table cost.

● DynamoDB Standard-IA
Recommended for tables that store data that is infrequently accessed, with storage as the dominant table cost.

Table class updates is a background process. The time to update your table class depends on your table traffic, storage size, and other related variables. You can still access your table normally while it is converted. Note that no more than two table class updates on your table are allowed in a 30-day trailing period. Learn more

Standard テーブルクラス

Standard テーブルクラスは新しいテーブルのデフォルトのオプションです。このオプションでは、スループットと、アクセス頻度の高いデータを含むテーブルのストレージコストのバランスが取れる DynamoDB の元の請求残高が維持されます。

Standard-IA テーブルクラス

Standard-IA テーブルクラスは、更新や読み込みの頻度が低いデータの長期保存を必要とするワークロードのストレージコストを削減 (最大 60% 削減) するのに役立ちます。このクラスはアクセス頻度

の低いクラス向けに最適化されているため、読み込みと書き込みの料金は Standard テーブルクラスよりも若干高く (最大 25% 高い) なります。

DynamoDB Standard テーブルクラスを選択する場合

DynamoDB Standard テーブルクラスは、ストレージコストがテーブルの月額総コストの約 50% 以下のテーブルに最適です。このコストバランスは、DynamoDB 内に既に保存されている項目に定期的にアクセスするか、項目を更新するワークロードを示しています。

DynamoDB Standard-IA テーブルクラスを選択する場合

DynamoDB Standard-IA テーブルクラスは、ストレージコストがテーブルの月額総コストの約 50% 以上であるテーブルに最適です。このコストバランスは、1 か月あたりの項目の作成または読み込みがストレージに保持される量よりも少ないワークロードを示しています。

Standard-IA テーブルクラスの一般的な使用方法は、アクセス頻度の低いデータを個々の Standard-IA テーブルに移動することです。詳細については、「Optimizing the storage costs of your workloads with Amazon DynamoDB Standard-IA table class (Amazon DynamoDB Standard-IA テーブルクラスによるワークロードのストレージコストの最適化)」を参照してください。

テーブルクラスを選択する際に考慮すべきその他の要素

2 つのテーブルクラスのどちらかを決める際には、決定するうえで考慮すべき要素が他にもいくつかあります。

リザーブドキャパシティ

Standard-IA テーブルクラスを使用するテーブルのリザーブドキャパシティの購入は、現在サポートされていません。リザーブドキャパシティを利用している Standard テーブルからリザーブドキャパシティを利用していない Standard-IA テーブルに移行する場合、コスト上のメリットが得られない可能性があります。

未使用のリソースを特定する

このセクションでは、未使用のリソースを定期的に評価する方法の概要を説明します。アプリケーションの要件が変化するにつれて、未使用のリソースによって不要な Amazon DynamoDB のコストが生じないようにする必要があります。以下に説明する手順では、Amazon CloudWatch メトリクスを使用して未使用のリソースを特定します。さらにこの手順は、リソースを特定してコストを削減するための対策を講じるのに役立ちます。

CloudWatch を使用して DynamoDB をモニタリングすることで、DynamoDB から raw データを収集し、リアルタイムに近い読み込み可能なメトリクスに加工することができます。これらの統計は一定

期間保持されるため、履歴情報にアクセスして使用率をより正確に調べることができます。デフォルトでは、DynamoDB メトリクスデータは CloudWatch に自動的に送信されます。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch とは」および「メトリクスの保持」を参照してください。

トピック

- 未使用のリソースを特定する方法
- 未使用のテーブルリソースを特定する
- 未使用のテーブルリソースをクリーンアップする
- 未使用の GSI リソースを特定する
- 未使用の GSI リソースをクリーンアップする
- 未使用のグローバルテーブルをクリーンアップする
- 未使用のバックアップまたはポイントインタイムリカバリ (PITR) をクリーンアップする

未使用のリソースを特定する方法

未使用のテーブルやインデックスを特定するには、30 日間にわたって次の CloudWatch メトリクスを調べ、テーブルに対するアクティブな読み込みまたは書き込み、あるいはグローバルセカンダリインデックス (GSI) の読み込みがあるかどうかを調べます。

ConsumedReadCapacityUnits

指定された期間に消費された読み込みキャパシティユニットの数。消費されたキャパシティの使用量を追跡できます。テーブルとそのすべてのグローバルセカンダリインデックス、または特定のグローバルセカンダリインデックスの消費された読み込み容量の合計を取得できます。

ConsumedWriteCapacityUnits

指定された期間に消費された書き込みキャパシティユニットの数。消費されたキャパシティの使用量を追跡できます。テーブルとそのすべてのグローバルセカンダリインデックス、または特定のグローバルセカンダリインデックスの消費された書き込み容量の合計を取得できます。

未使用のテーブルリソースを特定する

Amazon CloudWatch は、モニタリングおよびオブザーバビリティサービスで、未使用のリソースの特定に使用する DynamoDB テーブルのメトリクスを提供します。CloudWatch メトリクスは、AWS Management Console と AWS Command Line Interface を使って表示できます。

AWS Command Line Interface

AWS Command Line Interface を使ってテーブルのメトリクスを表示する場合は、次のコマンドを使用できます。

1. まず、テーブルの読み込みを評価します。

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

テーブルが未使用と誤って識別されないようにするには、長期間にわたってメトリクスの評価を行います。たとえば、30 日間と、適切な開始時刻と終了時刻の範囲、および 86400 などで適切な期間を選択します。

返されるデータで、[合計] が 0 を超える場合は、評価対象のテーブルがその期間に受信した読み込みトラフィックを示します。

次の結果は、評価期間中に読み込みトラフィックを受信したテーブルを示しています。

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

次の結果は、評価期間中に読み込みトラフィックを受信しなかったテーブルを示しています。

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. 次に、テーブルの書き込みを評価します。

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

   テーブルが未使用と誤って識別されないようにするには、長期間にわたってメトリクスを評価する必要があります。30 日間などの適切な開始時刻と終了時刻の範囲、および 86400 などの適切な期間を選択します。

   返されるデータで、[Sum (合計)] が 0 を超える場合は、評価対象のテーブルがその期間に受信した読み込みトラフィックを示します。

   次の結果は、評価期間中に書き込みトラフィックを受信したテーブルを示しています。

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

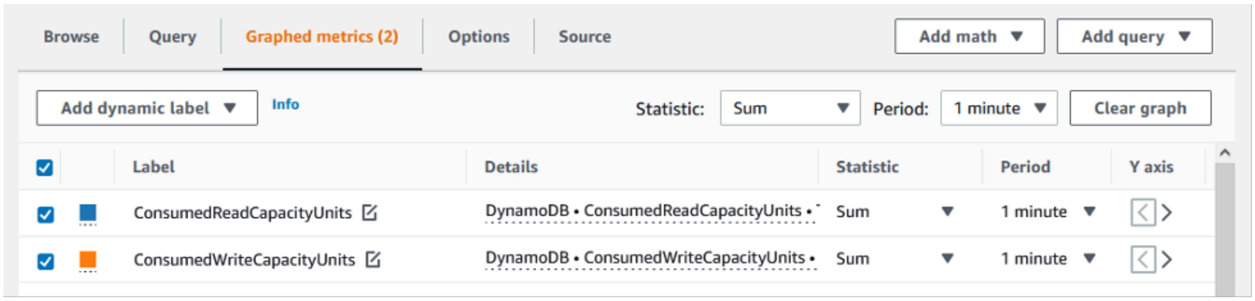
   次の結果は、評価期間中に書き込みトラフィックを受信しなかったテーブルを示しています。

```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

AWS Management Console

次のステップでは、AWS Management Console を使用してリソースの使用率を評価できます。

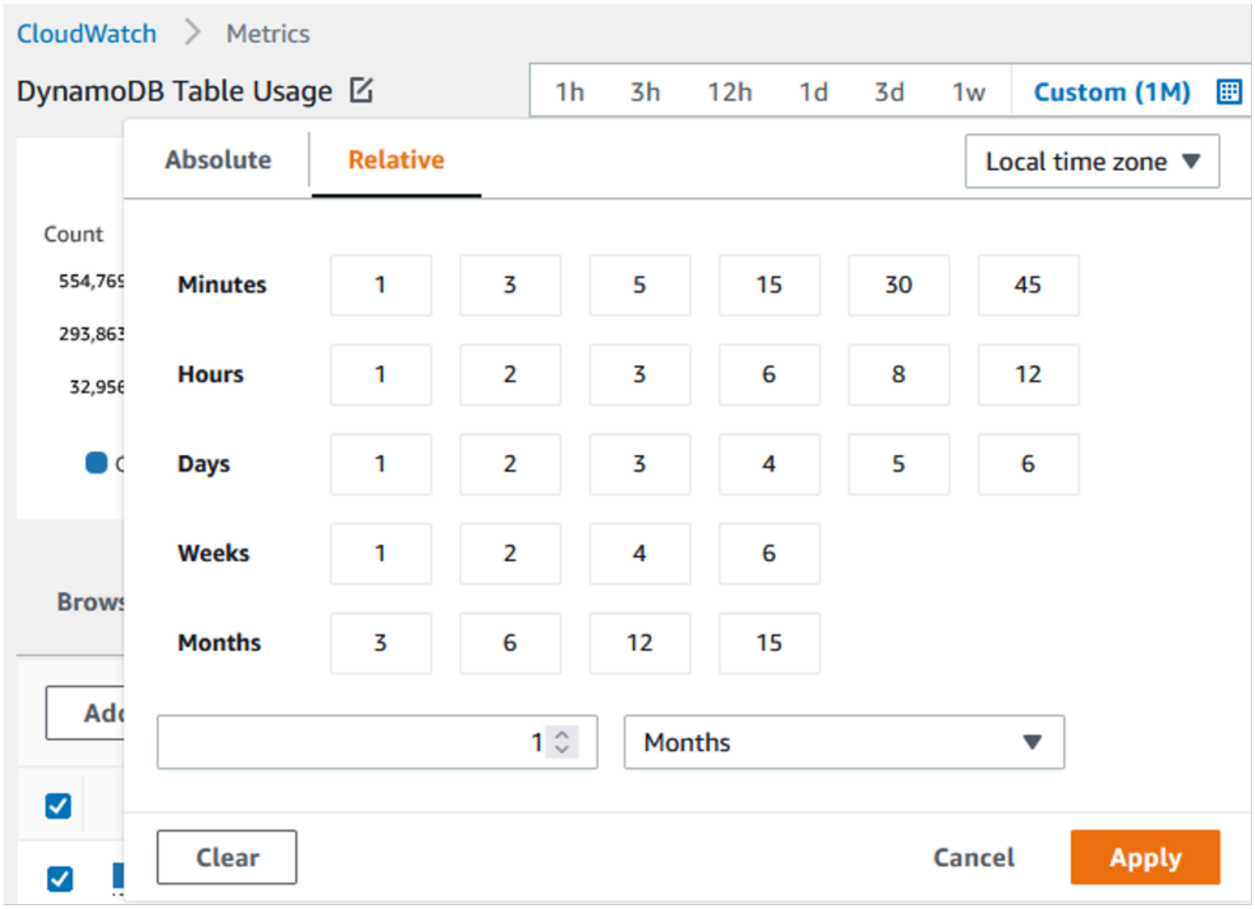
1. AWS コンソールにログインして、CloudWatch サービスページ (https://console.aws.amazon.com/cloudwatch/) を開きます。必要に応じて、コンソールの右上で適切な AWS リージョンを選択します。
2. 左のナビゲーションバーで、[Metrics] (メトリクス) セクションを探し、[All metrics] (すべてのメトリクス) を選択します。
3. これにより、2 つのパネルで構成されるダッシュボードが開きます。上部のパネルには、現在グラフ化されているメトリクスが表示されます。下部のパネルでは、グラフ化できるメトリクスを選択できます。下部のパネルで DynamoDB を選択します。
4. DynamoDB メトリクスの選択パネルで [Table Metrics] (テーブルメトリクス) カテゴリを選択し、現在のリージョンのテーブルのメトリクスを表示します。
5. メニューを下にスクロールしてテーブル名を確認し、テーブルの `ConsumedReadCapacityUnits` および `ConsumedWriteCapacityUnits` メトリクスを選択します。
6. [Graphed metrics (2)] (グラフ化したメトリクス (2)) タブを選択し、[Statistic] (統計) 列を [Sum] (合計) に設定します。



Statistic: Sum ▼ Period: 1 minute ▼

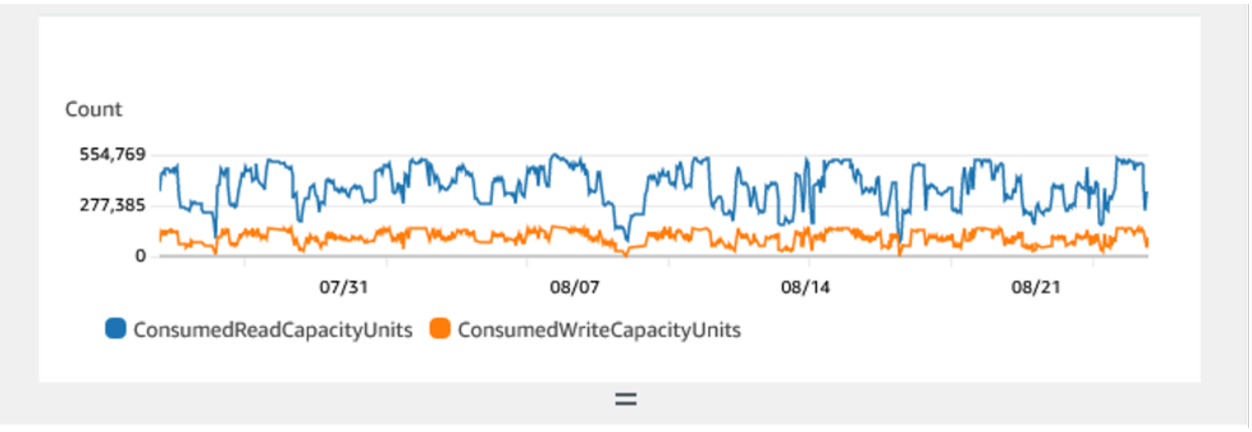
☑		Label	Details	Statistic	Period	Y axis
☑	🟦	ConsumedReadCapacityUnits	DynamoDB • ConsumedReadCapacityUnits •	Sum ▼	1 minute ▼	< >
☑	🟧	ConsumedWriteCapacityUnits	DynamoDB • ConsumedWriteCapacityUnits •	Sum ▼	1 minute ▼	< >

7. テーブルが未使用と誤って識別されないようにするには、長期間にわたってメトリクスを評価する必要があります。グラフパネルの上部で、テーブルを評価するための適切な時間枠 (1 か月など) を選択します。[Custom] (カスタム) を選択し、ドロップダウンで [1 Months] (1 か月間) を選択し、[Apply] (適用) を選択します。

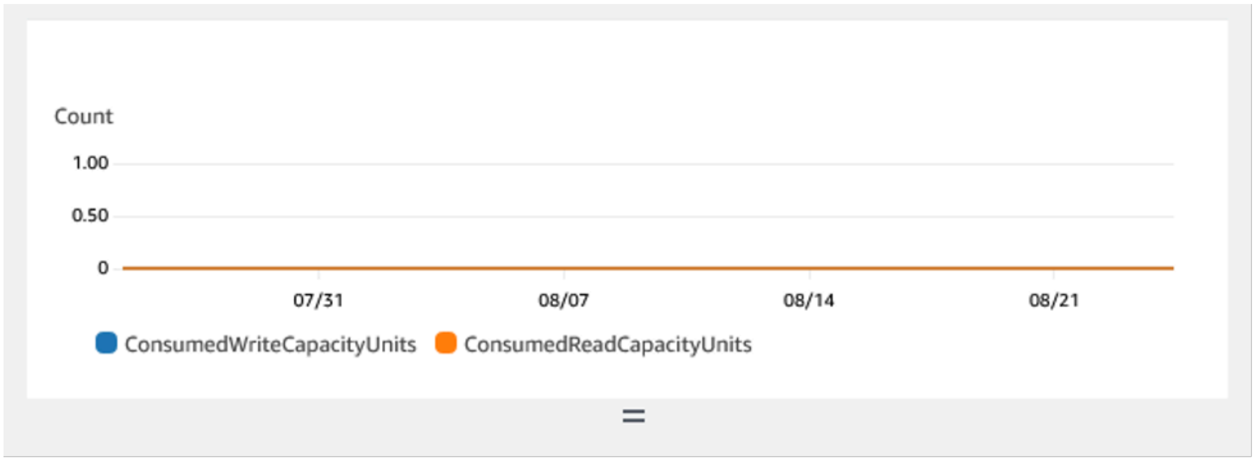


8. テーブルのグラフ化されたメトリクスを評価して、使用されているかどうかを判断します。0 を超えるメトリクスは、評価期間中にテーブルが使用されたことを示します。読み込みと書き込みの両方が 0 の平らなグラフは、テーブルが未使用であることを示します。

   以下の画像は、読み込みトラフィックがあるテーブルを示しています。



   以下の画像は、読み込みトラフィックがないテーブルを示しています。



未使用のテーブルリソースをクリーンアップする

未使用のテーブルリソースを特定したら、次の方法でその継続的なコストを削減できます。

> **Note**
>
> 未使用のテーブルを特定したものの、将来アクセスする必要が生じた場合に備えて使用できるようにしておきたい場合は、オンデマンドモードに切り替えることを検討してください。それ以外の場合、テーブル全体をバックアップして削除することを検討できます。

キャパシティモード

DynamoDB テーブル内のデータの読み込み、書き込み、保存には料金がかかります。

DynamoDB には 2 種類のキャパシティモードがあり、テーブルの読み書き処理について別個の請求オプション (オンデマンドとプロビジョンド) があります。読み取り/書き込みキャパシティモードは、読み取りおよび書き込みスループットの課金方法とキャパシティの管理方法を制御します。

オンデマンドモードのテーブルでは、アプリケーションで実行することが予測される読み込みおよび書き込みスループットを指定する必要はありません。DynamoDB では、読み込みリクエスト単位と書き込みリクエスト単位に関して、アプリケーションがテーブルに対して実行する読み込みと書き込みに対して料金が請求されます。テーブル/インデックスにアクティビティがない場合、スループットに対する支払いは発生しませんが、ストレージ料金は発生します。

テーブルクラス

DynamoDB には、コストの最適化に役立つように設計された 2 つのテーブルクラスも用意されています。デフォルトは DynamoDB Standard テーブルクラスで、大半のワークロードで推奨されていま

す。DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) テーブルクラスは、ストレージが主要なコストとなるテーブル向けに最適化されています。

テーブルまたはインデックスにアクティビティがない場合は、ストレージがコストの大きな割合を占める可能性が高く、テーブルクラスを変更することで大幅な節約が実現します。

テーブルの削除

未使用のテーブルを発見し、それを削除する場合は、まずデータのバックアップまたはエクスポートを行うことをお勧めします。

AWS Backup によるバックアップでは、コールドストレージの階層化を活用できるため、コストをさらに削減できます。AWS Backup を使用してバックアップを有効にする方法については「DynamoDB での AWS Backup の使用」のドキュメントを、ライフサイクルを使用してバックアップをコールドストレージに移動する方法については、「Managing backup plans」(バックアッププランの管理) のドキュメントを参照してください。

または、テーブルのデータを S3 にエクスポートすることもできます。その場合は、「Export to Amazon S3」(Amazon S3 へのエクスポート) のドキュメントを参照してください。データをエクスポートした後に、S3 Glacier Instant Retrieval、S3 Glacier Flexile Retrieval、または S3 Glacier Deep Archive を活用してさらにコストを削減する場合は、「Managing your storage lifecycle」(ストレージのライフサイクルの管理) を参照してください。

テーブルをバックアップしたら、AWS Management Console または AWS Command Line Interface を使用してテーブルを削除できます。

未使用の GSI リソースを特定する

未使用のグローバルセカンダリを特定するステップは、未使用のテーブルを特定するステップと似ています。ベーステーブルに書き込まれた項目に GSI のパーティションキーとして使用される属性が含まれている場合、DynamoDB は項目を GSI にレプリケートするため、ベーステーブルが使用中であれば、未使用の GSI でも `ConsumedWriteCapacityUnits` が 0 を超える可能性があります。その結果、`ConsumedReadCapacityUnits` メトリクスのみを評価して、GSI が未使用かどうかを判断することになります。

AWS AWS CLI を使って GSI メトリクスを表示する場合は、次のコマンドを使用してテーブルの読み込みを評価できます。

```
aws cloudwatch get-metric-statistics --metric-name
```

```
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
Name=GlobalSecondaryIndexName,Value=<index-name>
```

テーブルが未使用と誤って識別されないようにするには、長期間にわたってメトリクスを評価する必要があります。30 日間などの適切な開始時刻と終了時刻の範囲、および 86400 などの適切な期間を選択します。

返されるデータで、[Sum] (合計) が 0 を超える場合は、評価対象のテーブルがその期間に受信した読み込みトラフィックを示します。

次の結果は、評価期間中に読み込みトラフィックを受信した GSI を示しています。

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

次の結果は、評価期間中に最小限の読み込みトラフィックを受信した GSI を示しています。

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-15T21:20:00Z",
          "Sum": 2.0,
          "Unit": "Count"
        },
```

次の結果は、評価期間中に読み込みトラフィックを受信しなかった GSI を示しています。

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
```

未使用の GSI リソースをクリーンアップする

未使用の GSI が見つかった場合は、それを削除することができます。GSI に存在するすべてのデータはベーステーブルにも存在するため、GSI を削除する前に追加のバックアップを行う必要はありません。今後、GSI が再び必要になった場合は、テーブルに追加し直すことができます。

使用頻度の低い GSI が見つかった場合は、それを削除したりコストを削減したりできるように、アプリケーションの設計変更を検討する必要があります。例えば、DynamoDB のスキャンは大量のシステムリソースを消費するため、控え目に使用する必要がありますが、サポートするアクセスパターンが極めて低頻度であれば、GSI よりもコスト効率が高くなる可能性があります。

さらに、低頻度のアクセスパターンを GSI でサポートする必要がある場合は、より限定された属性セットを射影することを検討してください。この場合、低頻度のアクセスパターンをサポートするために、ベーステーブルに対して後続のクエリが必要になる場合がありますが、ストレージと書き込みのコストを大幅に削減できる可能性があります。

未使用のグローバルテーブルをクリーンアップする

Amazon DynamoDB グローバルテーブルは、マルチリージョンにマルチアクティブデータベースをデプロイするための完全マネージド型のソリューションです。独自のレプリケーションソリューションを構築および管理する必要はありません。

グローバルテーブルは、ユーザーに近いデータに低レイテンシーでアクセスしたり、ディザスタリカバリのためのセカンダリリージョンにしたりするのに最適です。

データへの低レイテンシーアクセスを提供するためにリソースに対してグローバルテーブルオプションが有効になっていても、ディザスタリカバリ戦略の一部でない場合は、CloudWatch メトリクスを評価して、両方のレプリカが読み込みトラフィックをアクティブに処理していることを確認します。1 つのレプリカが読み込みトラフィックを処理しない場合、そのレプリカは未使用のリソースである可能性があります。

グローバルテーブルがディザスタリカバリ戦略の一部である場合、アクティブ/スタンバイパターンでは、1 つのレプリカが読み込みトラフィックを受信しないことが予想されます。

未使用のバックアップまたはポイントインタイムリカバリ (PITR) をクリーンアップする

DynamoDB には 2 種類のバックアップスタイルがあります。ポイントインタイムリカバリでは 35 日間の継続的バックアップを利用できるため、意図しない書き込みや削除を防ぐことができます。また、オンデマンドバックアップでは、長期的に保存できるスナップショットを作成できます。どちらのタイプのバックアップにも、関連するコストがあります。

「DynamoDB のオンデマンドバックアップおよび復元の使用」および「DynamoDB のポイントインタイムリカバリ」のマニュアルを参照して、不要になった可能性のあるバックアップがテーブルで有効になっているかどうかを確認してください。

テーブルの使用パターンを評価する

このセクションでは、DynamoDB テーブルを効率的に使用しているかどうかを評価する方法の概要を説明します。DynamoDB には最適ではない特定の使用パターンがあり、パフォーマンスとコストの両方の観点から最適化の余地があります。

トピック

強力な整合性のある読み込みオペレーションの数を減らす

DynamoDB では、リクエストごとに読み込み整合性を設定できます。デフォルトでは、読み込みリクエストは結果的に整合性が保たれます。結果整合性のある読み込みは、最大 4 KB のデータに対して 0.5 RCU で課金されます。

分散型ワークロードのほとんどの部分は柔軟性があり、最終的な一貫性を許容できます。ただし、強力な整合性のある読み込みを必要とするアクセスパターンもあり得ます。強力な整合性のある読み込みには、最大 4 KB のデータに対して 1 RCU の料金が課金されるため、読み込みコストは実質的に 2 倍になります。DynamoDB では、同じテーブルで両方の整合性モデルを柔軟に使用できます。

ワークロードとアプリケーションコードを評価して、強力な整合性のある読み込みが必要な場合にのみ使用されているかどうかを確認できます。

読み込み操作のトランザクション数を減らす

DynamoDB では、特定のアクションを「すべて」か「なし」でグループ化できます。つまり、DynamoDB で ACID トランザクションを実行することが可能です。ただし、リレーショナルデータベースの場合と同様に、すべてのアクションでこのアプローチに従う必要はありません。

最終的に一貫した方法で同じ量のデータを読み取るためにデフォルトでは 0.5 RCU が使用される一方、最大 4 KB のトランザクション読み込みオペレーションでは 2 RCU が使用されます。書き込み操作のコストは 2 倍になります。つまり、1 KB までのトランザクション書き込みは、2 WCU に相当します。

テーブルのすべてのオペレーションがトランザクションかどうかを判断するには、テーブルの CloudWatch メトリクスをトランザクション API に絞り込むことができます。テーブルの SuccessfulRequestLatency メトリクスで使用できるグラフがトランザクション API だけであれば、すべてのオペレーションがこのテーブルのトランザクションであることを確認できます。あるいは、全体的なキャパシティ使用率の傾向がトランザクション API の傾向と一致する場合は、トランザクション API が大部分を占めていると考えられるため、アプリケーション設計を見直すことを検討してください。

スキャンの回数を減らす

Scan オペレーションを多用するのは、一般的に DynamoDB データに対して分析クエリを実行する必要があるためです。大きなテーブルで頻繁に Scan オペレーションを実行するのは、非効率的でコストが高額になる可能性があります。

また、Export to S3 機能を使用し、特定の時点を選択してテーブルの状態を S3 にエクスポートする方法もあります。AWS には、テーブルの容量を消費することなく、データに対して分析クエリを実行できる Athena のようなサービスがあります。

Scan オペレーションの頻度は、Scan の SuccessfulRequestLatency メトリクスの SampleCount 統計を使用して決定できます。Scan オペレーションが非常に頻繁に行われる場合は、アクセスパターンとデータモデルの再評価が推奨されます。

属性名を短くする

DynamoDB の項目の合計サイズは、属性名と属性値の長さの合計です。属性名が長いと、ストレージコストだけでなく、RCU/WCU の消費量も増える可能性があります。属性名は長いものよりも短いものにすることをお勧めします。属性名を短くすると、項目サイズを次の 4KB/1KB の境界内に

制限でき、それを超えるとデータにアクセスするために追加の RCU/WCU を消費することになります。

有効期限 (TTL) の有効化

有効期限 (TTL) では、項目に設定した有効期限を過ぎた項目を特定してテーブルから削除できます。データが時間の経過とともに増加し、古いデータが重要でなくなった場合は、テーブルで TTL を有効にすると、データを削減してストレージコストを節約できます。

TTL のもう 1 つの便利な点は、期限切れの項目が DynamoDB ストリームで発生することです。そのため、データからデータを削除するだけでなく、ストリームからそれらの項目を消費して低コストのストレージ階層にアーカイブすることができます。さらに、TTL による項目の削除に追加コストはかかりません。容量を消費せず、クリーンアップアプリケーションを設計するオーバーヘッドもありません。

グローバルテーブルをクロスリージョンバックアップに置き換える

グローバルテーブルを使用すると、異なるリージョンに複数のアクティブなレプリカテーブルを管理できます。これらのテーブルはすべて書き込み操作を受け付け、相互にデータをレプリケートできます。レプリカの設定は簡単で、同期は自動的に管理されます。レプリカは、最新書き込み優先ストラテジーを使用して一貫した状態に収束します。

グローバルテーブルをフェイルオーバーまたはディザスタリカバリ (DR) 戦略の一環としてのみ使用している場合は、比較的緩やかな復旧ポイント目標と復旧時間目標の要件を満たすために、グローバルテーブルをクロスリージョンバックアップコピーに置き換えることができます。高速なローカルアクセスと 99.999% の高可用性が必要ない場合は、グローバルテーブルレプリカを維持することはフェイルオーバーに最適な方法ではないかもしれません。

別の方法として、AWS Backup を使用して DynamoDB バックアップを管理することを検討してください。定期的なバックアップをスケジュールしてリージョン間でコピーすることで、グローバルテーブルを使用するよりも費用対効果の高い方法で、DR 要件を満たすことができます。

Streams の使用量を評価する

このセクションでは、DynamoDB Streams の使用量を評価する方法についての概要を説明します。DynamoDB には最適ではない特定の使用パターンがあり、パフォーマンスとコストの両方の観点から最適化の余地があります。

ストリーミングとイベントドリブンのユースケースには、次の 2 つのネイティブストリーミング統合が用意されています。

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

このページでは、これらのオプションのコスト最適化戦略のみに焦点を当てます。2 つのオプションのどちらかを選択する方法については、「変更データキャプチャのストリーミングオプション」を参照してください。

トピック

DynamoDB Streams のコストの最適化

DynamoDB Streams の料金ページに記載されているように、テーブルのスループットキャパシティモードに関係なく、DynamoDB ではテーブルの DynamoDB Streams に対して行われた読み取りリクエストの数に応じて料金が発生します。DynamoDB Streams に対して行われる読み取りリクエストは、DynamoDB テーブルに対して行われる読み取りリクエストとは異なります。

Streams に関する各読み取りリクエストは `GetRecords` API コールの形式で行われ、最大 1,000 件のレコードまたは 1 MB 相当のレコードがレスポンスとして返されます。他のいずれの DynamoDB Streams API、またはアイドル状態の DynamoDB Streams では料金が発生しません。つまり、DynamoDB Streams に対して読み取りリクエストが行われない場合、テーブルで DynamoDB Streams を有効にしても料金は発生しません。

DynamoDB Streams コンシューマーアプリケーションをいくつか紹介します。

- AWS Lambda 関数
- Amazon Kinesis Data Streams ベースのアプリケーション
- AWS SDK を使用して構築されたお客様向けコンシューマーアプリケーション

DynamoDB Streams の AWS Lambda ベースのコンシューマーによる読み取りリクエストは無料ですが、他の種類のコンシューマーによる呼び出しには料金が発生します。毎月、Lambda 以外のユーザーによる読み取りリクエストのうち、最初の 2,500,000 件も無料です。これは、各 AWS リージョンの AWS アカウント内の DynamoDB Streams に対するすべての読み込みリクエストに適用されます。

DynamoDB Streams 使用量のモニタリング

請求コンソールの DynamoDB Streams の料金は、AWS アカウント内の AWS リージョンにあるすべての DynamoDB Streams でまとめられます。現在、DynamoDB Streams のタグ付けはサポートされていないため、コスト配分タグを使用して DynamoDB Streams の詳細なコストを特定することはできません。`GetRecords` の呼び出し量を DynamoDB Streams レベルで取得して、ストリームあたりの料金を計算することは可能です。呼び出し量は DynamoDB Streams の CloudWatch メトリクス `SuccessfulRequestLatency` とその `SampleCount` 統計によって表されます。このメトリクスには、継続的なレプリケーションを行うためのグローバルテーブルによる `GetRecords` コールや、AWS Lambdaコンシューマによるコールも含まれますが、いずれも料金は発生しません。DynamoDB Streams によって公開されるその他の CloudWatch メトリクスについては、「DynamoDB のメトリクスとディメンション」を参照してください。

AWS Lambda をコンシューマーとして使用する

AWS Lambda 関数を DynamoDB Streams のコンシューマーとして使用できるかどうかを評価してください。これにより、DynamoDB Streams からの読み取りに関連するコストを削減できるためです。一方、DynamoDB Streams Kinesis Adapter または SDK ベースのコンシューマーアプリケーションでは、DynamoDB Streams に対して行った `GetRecords` 呼び出しの数に応じて料金が発生します。

Lambda 関数の呼び出しは Lambda の標準料金に基づいて課金されますが、DynamoDB Streams では料金は発生しません。Lambda は、レコードの DynamoDB Streams にあるシャードを 1 秒あたり 4 回の基本レートでポーリングします。レコードが利用可能になると、Lambda は関数を呼び出し、結果を待機します。処理が成功すると、Lambda は、レコードをさらに受け取るまでポーリングを再開します。

DynamoDB Streams Kinesis Adapter ベースのコンシューマーアプリケーションのチューニング

Lambda ベースではないコンシューマーによる読み取りリクエストは DynamoDB Streams に対して課金されるため、ほぼリアルタイムの要件と、コンシューマーアプリケーションが DynamoDB Streams をポーリングしなければならない回数とのバランスをとることが重要です。

DynamoDB Streams Kinesis Adapter ベースのアプリケーションを使用して DynamoDB Streams をポーリングする頻度は、設定された `idleTimeBetweenReadsInMillis` 値によって決まります。このパラメータは、同じシャードへの前回の `GetRecords` 呼び出しでレコードが返されなかった場合に、コンシューマーがシャードを処理する前に待機する時間をミリ秒単位で決定します。デフォルトでは、このパラメータの値は 1,000 ミリ秒です。ほぼリアルタイムの処理が必要ない場合は、このパラメータを増やしてコンシューマーアプリケーションが行う `GetRecords` 呼び出しを減らし、DynamoDB Streams 呼び出しを最適化できます。

Kinesis Data Streams のコストの最適化

Kinesis Data Streams が DynamoDB テーブルの変更データキャプチャイベントの送信先として設定されている場合、Kinesis Data Streams は個別のサイズ管理を必要とする場合があり、これは全体のコストに影響を与えます。DynamoDB は、変更データキャプチャユニット (CDU) 単位で課金されます。この場合、各ユニットは、DynamoDB サービスが送信先の Kinesis Data Streams に対して試行する 1 KB の DynamoDB 項目サイズで構成されます。

DynamoDB サービスによる料金に加えて、Kinesis Data Streams の標準の料金が発生します。料金ページに記載されているように、サービスの料金は、DynamoDB テーブルのキャパシティモードとは異なり、ユーザー定義のキャパシティモード (プロビジョニングまたはオンデマンド) によって異なります。おおまかに言うと、Kinesis Data Streams は、キャパシティモードと DynamoDB サービスによってストリームに取り込まれたデータに基づいて、時間単位で課金されます。Kinesis Data Streams のユーザー設定によっては、データ検索 (オンデマンドモードの場合)、データ保持の延長 (デフォルトの 24 時間を超える場合)、拡張ファンアウトコンシューマーの取得などの追加料金が発生する場合があります。

Kinesis Data Streams の使用量のモニタリング

DynamoDB 用 Kinesis Data Streams は、標準の Kinesis Data Streams CloudWatch メトリクスに加えて、DynamoDB からのメトリクスを発行します。Kinesis Data Streams のキャパシティ不足の場合は Kinesis によって、または Kinesis Data Streams の保管中のデータを暗号化するように設定された AWS KMS サービスなどの依存コンポーネントによって、DynamoDBサービスによる `Put` の試みがスロットリングされる可能性があります。

DynamoDB サービスが Kinesis Data Streams 用に発行する CloudWatch メトリクスの詳細については、「Kinesis Data Streams を使用した変更データキャプチャのモニタリング」を参照してください。スロットリングが原因のサービスの再試行による追加コストを回避するために、プロビジョニングモードの場合は Kinesis Data Streams のサイズを適切に設定することが重要です。

Kinesis Data Streams に適したキャパシティモードの選択

Kinesis Data Streams は、プロビジョニングモードとオンデマンドモードの 2 つのキャパシティモードでサポートされています。

- Kinesis Data Streams を含むワークロードのアプリケーショントラフィックが予測可能な場合や、トラフィックが安定しているか徐々に増加している場合、またはトラフィックが正確に予測できる場合は、Kinesis Data Streams プロビジョニングモードが適しており、コスト効率も高くなります。

- ワークロードが新しい場合や、アプリケーショントラフィックが予測できない場合、またはキャパシティを管理したくない場合は、Kinesis Data Streams のオンデマンドモードが適しており、コスト効率も高くなります。

コストを最適化するためのベストプラクティスは、Kinesis Data Streams に関連する DynamoDB テーブルが、Kinesis Data Streams のプロビジョニングモードを活用できる予測可能なトラフィックパターンを持っているかどうかを評価することです。ワークロードが新しい場合は、最初の数週間は Kinesis Data Streams にオンデマンドモードを使用し、CloudWatch メトリクスを確認してトラフィックパターンを理解してから、同じストリームをワークロードの性質に基づいてプロビジョニングモードに切り替えることができます。プロビジョニングモードの場合、Kinesis Data Streams のシャード管理に関する考慮事項に従うことでシャード数を推定できます。

DynamoDB 用 Kinesis Data Streams を使用したコンシューマーアプリケーションの評価

Kinesis Data Streams は DynamoDB Streams のように `GetRecords` の呼び出し回数で課金されないため、`GetRecords` のスロットリング制限内であれば、コンシューマーアプリケーションで何度でも呼び出しを行えます。Kinesis Data Streams のオンデマンドモードでは、データ読み取りは GB 単位で課金されます。プロビジョニングモードの Kinesis Data Streams では、データが 7 日未満の場合、読み取りは課金されません。Kinesis Data Streams コンシューマーとしての Lambda 関数の場合、Lambda は 1 秒あたり 1 回の基本レートで、レコードの Kinesis Data Streams にある各シャードをポーリングします。

どちらのタイプの Streams 使用にも適したコスト最適化戦略

AWS Lambda コンシューマー向けイベントフィルタリング

Lambda イベントフィルタリングを使用すると、フィルター条件に基づくイベントを Lambda 関数呼び出しバッチで使用できないように破棄できます。これにより、コンシューマー関数ロジック内の不要なストリームレコードを処理または破棄するための Lambda コストが最適化されます。イベントフィルタリングの設定とフィルタリング条件の記述の詳細については、「Lambda のイベントフィルタリング」を参照してください。

AWS Lambda コンシューマーの調整

`BatchSize` を大きくして呼び出しごとに処理する量を増やす、`BisectBatchOnFunctionError` を有効にして重複処理 (追加コストが発生する) を防ぐ、再試行回数が多すぎることのないように `MaximumRetryAttempts` を設定するなど、Lambda の設定パラメータを調整することで、コストをさらに最適化できます。デフォルトでは、コンシューマーの Lambda 呼び出しが失敗すると、ストリームからレコードの有効期限が切れるまで無限に再試行されます。DynamoDB Streams 場

合は 24 時間前後、Kinesis Data Streams では 24 時間から最長 1 年まで設定できます。上記の DynamoDB Streams コンシューマー向けの Lambda 設定オプションなど、その他の Lambda 設定オプションについては、「AWS Lambda デベロッパーガイド」を参照してください。

適切なサイズのプロビジョニングを行うために、プロビジョンドキャパシティを評価する

このセクションでは、DynamoDB テーブルのプロビジョニングが適切なサイズであるかどうかを評価する方法の概要を説明します。ワークロードの変化に応じて、運用手順を適切に変更する必要があります。特に DynamoDB テーブルがプロビジョニングモードで設定されていて、テーブルを過剰にプロビジョニングしたり、プロビジョニング不足になったりするリスクがある場合はそうです。

以下に説明する手順では、本稼働アプリケーションをサポートしている DynamoDB テーブルから取得する必要がある統計情報が必要です。アプリケーションの動作を理解するには、アプリケーションからのデータの季節性を把握するのに十分な期間を定義する必要があります。たとえば、アプリケーションに週単位のパターンが見られる場合は、3 週間の期間を使用することで、アプリケーションのスループットニーズを分析するための十分な余裕が得られます。

何から始めればよいかわからない場合は、以下の計算に少なくとも 1 か月分のデータ使用量を使用してください。

容量を評価する際、DynamoDB テーブルでは、読み込みキャパシティユニット (RCU) と書き込みキャパシティユニット (WCU) を個別に設定できます。テーブルにグローバルセカンダリインデックス (GSI) が設定されている場合は、そのテーブルが消費するスループットを指定する必要があります。これもベーステーブルの RCU や WCU から独立しています。

> Note
>
> ローカルセカンダリインデックス (LSI) はベーステーブルの容量を消費します。

トピック

- DynamoDB テーブルの消費メトリクスを取得する方法
- プロビジョニング不足の DynamoDB テーブルを識別する方法
- 過剰にプロビジョニングされた DynamoDB テーブルを識別する方法

DynamoDB テーブルの消費メトリクスを取得する方法

テーブルと GSI の容量を評価するには、次の CloudWatch メトリクスをモニタリングし、適切なディメンションを選択してテーブルまたは GSI 情報を取得します。

読み込みキャパシティユニット	書き込みキャパシティユニット
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

この操作は、AWS CLI または AWS Management Console で実行できます。

AWS CLI

テーブル消費メトリクスを取得する前に、まず CloudWatch API を使用して履歴データポイントを取得する必要があります。

まず、`write-calc.json` と `read-calc.json` の 2 つのファイルを作成します。これらのファイルは、テーブルまたは GSI の計算を表します。下の表に示すように、一部のフィールドを環境に合わせて更新する必要があります。

フィールド名	定義	例
`<table-name>`	分析するテーブルの名前	サンプルテーブル
`<period>`	ターゲット使用率の評価に使用する期間 (秒単位)	1 時間の場合は 3600 と指定します
`<start-time>`	ISO8601 形式で指定された評価間隔の開始	2022-02-21T23:00:00
`<end-time>`	ISO8601 形式で指定された評価間隔の終了	2022-02-22T06:00:00

書き込み計算ファイルに、指定された日付範囲の期間にプロビジョニングおよび消費された WCU の数が取得されます。また、分析に使用される使用率も生成されます。write-calc.json ファイルのすべての内容は以下のようになります。

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<ent-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

読み込み計算ファイルにも同様のファイルを使用します。このファイルには、指定された日付範囲にプロビジョニングおよび消費された RCU の数が取得されます。`read-calc.json` ファイルの内容は以下のようになります。

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedRCU/PERIOD(consumedRCU)",
      "Label": "Consumed RCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedRCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

ファイルを作成したら、使用率データの取得を開始できます。

1. 書き込み使用率データを取得するには、次のコマンドを実行します。

```
aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
```

2. 読み込み使用率データを取得するには、次のコマンドを実行します。

```
aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
```

両方のクエリの結果は、分析に使用される JSON 形式の一連のデータポイントになります。結果は、指定したデータポイントの数、期間、および特定のワークロードデータによって異なります。次のように表示されます。

```
{
    "MetricDataResults": [
        {
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
                "2022-02-22T05:00:00+00:00",
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
                24.9496875,
                40.94725694444445,
                25.61819444444444,
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

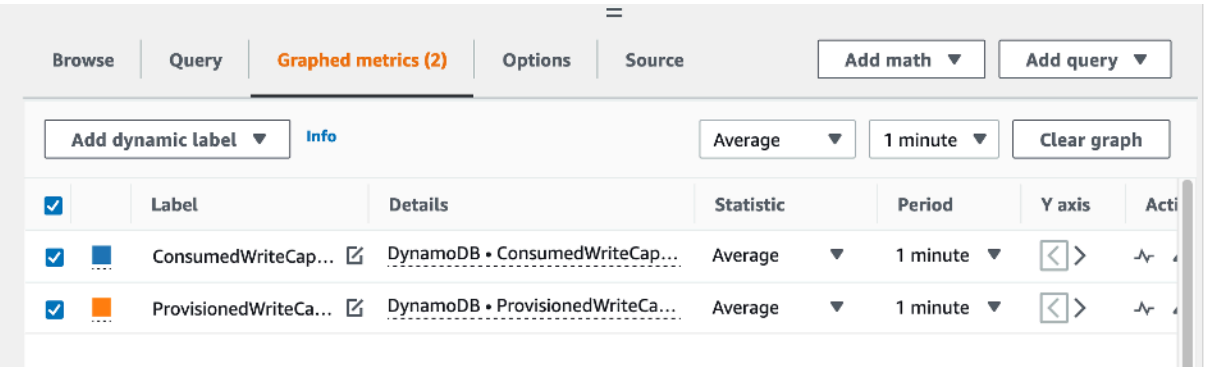
> Note
>
> 短い期間および長い時間範囲を指定する場合、スクリプトのデフォルトで 24 に設定されている `MaxDatapoints` を変更する必要があるかもしれません。これは、1 時間あたり 1 データポイント、1 日あたり 24 データポイントに相当します。

AWS Management Console

1. AWS Management Console にログインし、「AWS Management Console の開始方法」の CloudWatch サービスページに移動します。必要に応じて、適切な AWS リージョンを選択します。
2. 左のナビゲーションバーで、[Metrics] (メトリクス) セクションを見つけ、[All metrics] (すべてのメトリクス) を選択します。
3. これにより、2 つのパネルで構成されるダッシュボードが開きます。上のパネルにはグラフィックが表示され、下のパネルにはグラフ化するメトリクスが表示されます。DynamoDB パネルを選択します。
4. サブパネルから [Table Metrics] (テーブルメトリクス) カテゴリを選択します。これにより、現在のリージョンのテーブルが表示されます。
5. メニューを下にスクロールしてテーブル名を確認し、`ConsumedWriteCapacityUnits` および `ProvisionedWriteCapacityUnits` の書き込みオペレーションメトリクスを選択します。

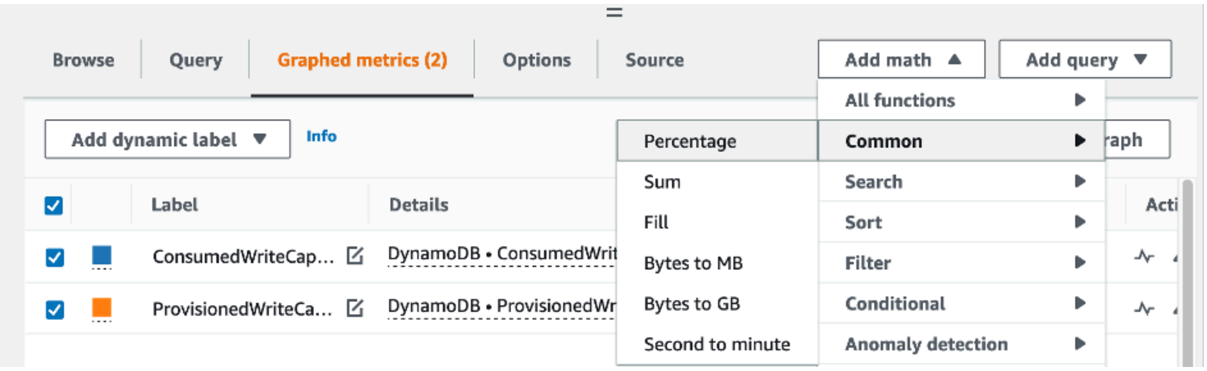
   > Note
   >
   > この例では書き込みオペレーションメトリクスについて説明していますが、これらの手順を使用して読み込みオペレーションメトリクスをグラフ化することもできます。

6. 数式を変更するには、[Graphed metrics (2)] (グラフ化したメトリクス (2)) タブを選択します。デフォルトでは、CloudWatch はグラフの統計関数 [Average] (平均) を選択します。

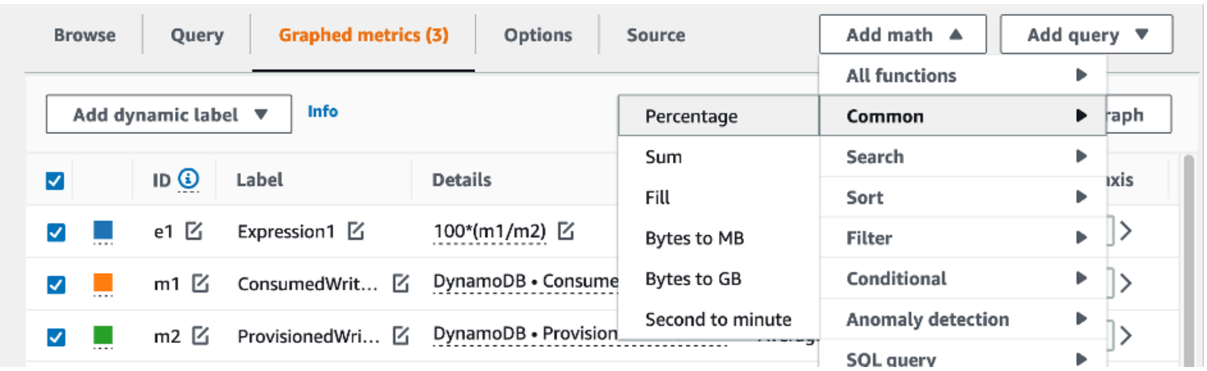

Browse
Query
Graphed metrics (2)
Options
Source
Add math ▼
Add query ▼
Add dynamic label ▼
Info
Average ▼
1 minute ▼
Clear graph
☑ Label Details Statistic Period Y axis Acti
☑ 🟦 ConsumedWriteCap... DynamoDB • ConsumedWriteCap... Average ▼ 1 minute ▼
☑ 🟧 ProvisionedWriteCa... DynamoDB • ProvisionedWriteCa... Average ▼ 1 minute ▼

7. グラフ化された両方のメトリクスを選択した状態 (左側のチェックボックス) で、[Add math] (算術の追加) メニュー、[Common] (共通) の順に選択し、次に [Percentage] (パーセンテージ) 関数を選択します。この手順を 2 回繰り返します。

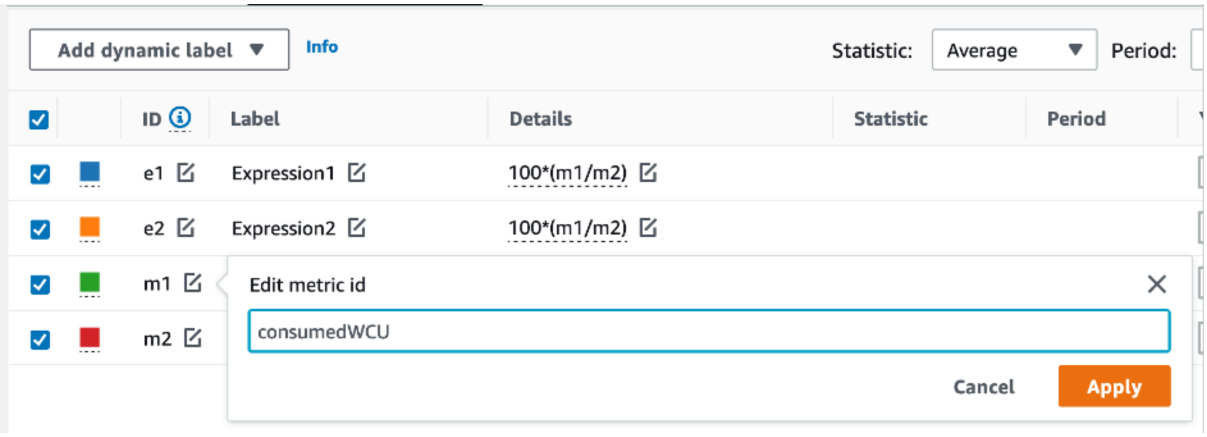
   [Percentage] (パーセンテージ) 関数を初めて選択する場合:



   [Percentage] (パーセンテージ) 関数を 2 回目に選択する場合:



8. この時点で、下部のメニューに 4 つのメトリクスが表示されているはずです。`ConsumedWriteCapacityUnits` の計算に取り掛かりましょう。一貫性を保つために、AWS CLI セクションで使用した名前と一致させる必要があります。[m1 ID] をクリックし、この値を [ConsumedWCU] に変更します。

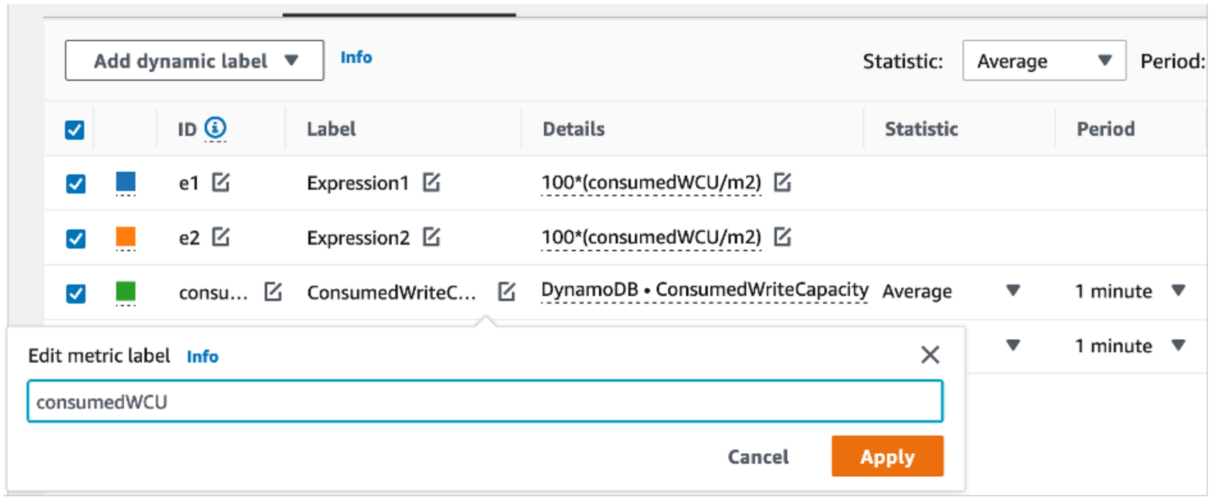

Add dynamic label ▼ Info Statistic: Average ▼ Period:

☑		ID	Label	Details	Statistic	Period
☑	🟦	e1	Expression1	100*(m1/m2)		
☑	🟧	e2	Expression2	100*(m1/m2)		
☑	🟩	m1				
☑	🟥	m2				

Edit metric id

consumedWCU

Cancel Apply


Add dynamic label ▼ Info Statistic: Average ▼ Period:

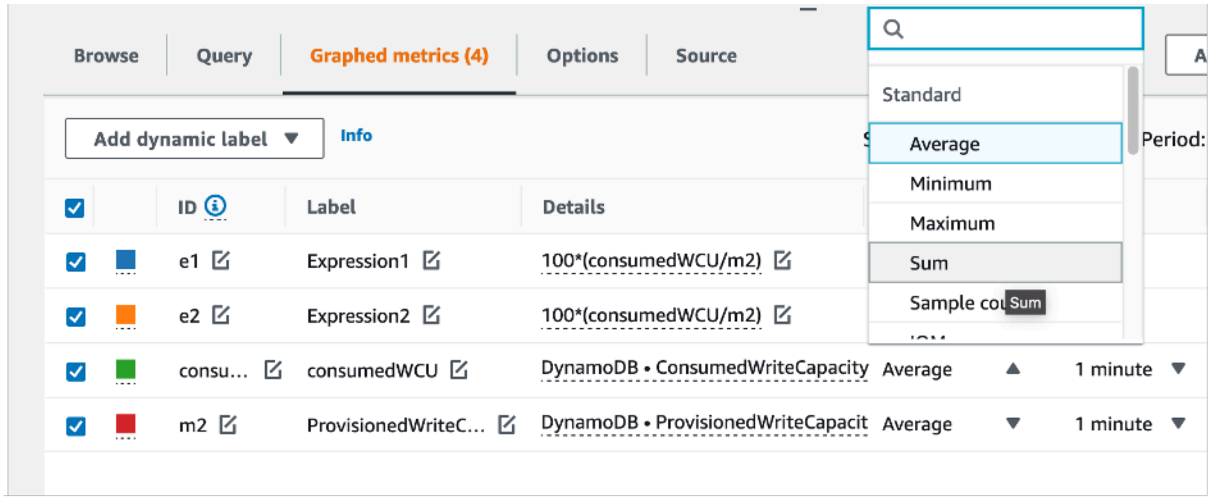
☑		ID	Label	Details	Statistic	Period
☑	🟦	e1	Expression1	100*(consumedWCU/m2)		
☑	🟧	e2	Expression2	100*(consumedWCU/m2)		
☑	🟩	consu…	ConsumedWriteC…	DynamoDB • ConsumedWriteCapacity	Average ▼	1 minute ▼
					▼	1 minute ▼

Edit metric label Info

consumedWCU

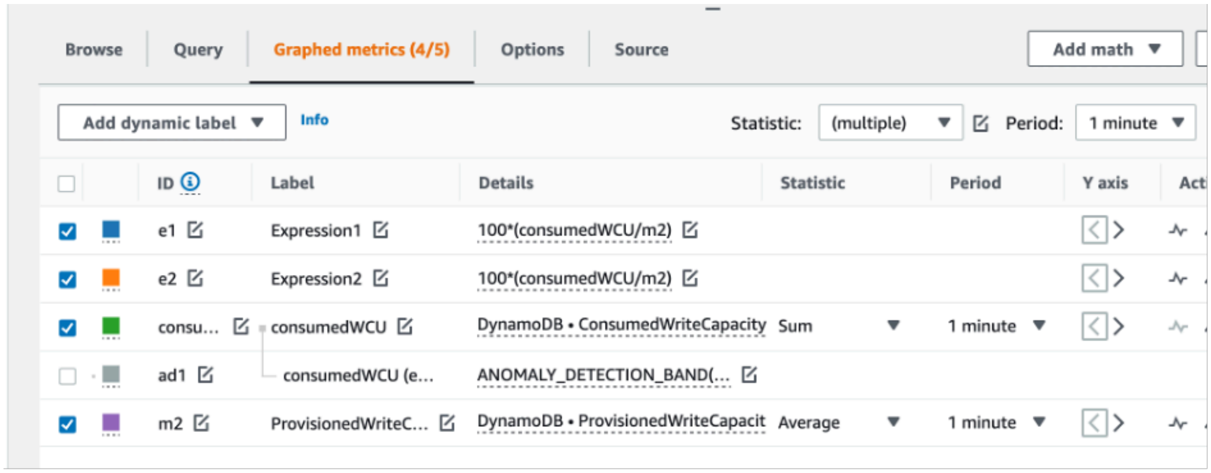
Cancel Apply

9. 統計を Average から Sum に変更します。このアクションにより、ANOMALY_DETECTION_BAND という名前の別のメトリクスが自動的に作成されます。この手順の範囲は、新しく生成された [ad1 metric] (ad1 メトリクス) のチェックボックスをオフにすれば無視できます。



Add dynamic label ▼ Info Period:

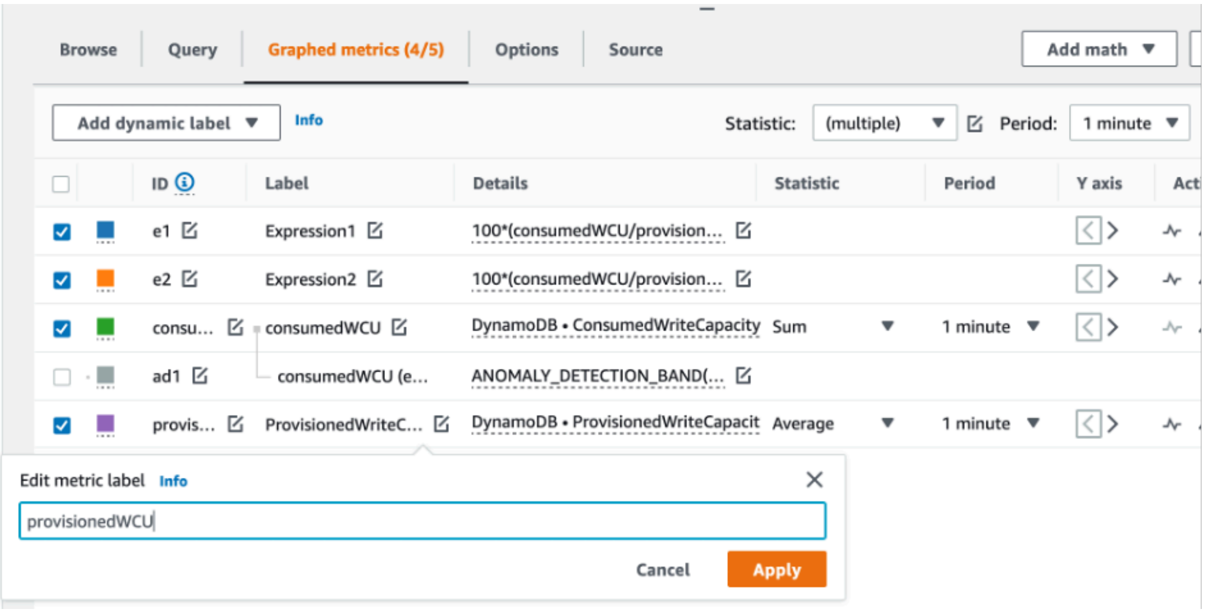
☑		ID	Label	Details	Statistic	Period
☑	🟦	e1	Expression1	100*(consumedWCU/m2)		
☑	🟧	e2	Expression2	100*(consumedWCU/m2)		
☑	🟩	consu…	consumedWCU	DynamoDB • ConsumedWriteCapacity	Average ▲	1 minute ▼
☑	🟥	m2	ProvisionedWriteC…	DynamoDB • ProvisionedWriteCapacit	Average ▼	1 minute ▼



Statistic: (multiple) Period: 1 minute

☐		ID	Label	Details	Statistic	Period	Y axis	Act
☑	🟦	e1	Expression1	100*(consumedWCU/m2)				
☑	🟧	e2	Expression2	100*(consumedWCU/m2)				
☑	🟩	consu...	consumedWCU	DynamoDB • ConsumedWriteCapacity	Sum	1 minute		
☐	⬜	ad1	consumedWCU (e...	ANOMALY_DETECTION_BAND(...				
☑	🟪	m2	ProvisionedWriteC...	DynamoDB • ProvisionedWriteCapacit	Average	1 minute		

10. 手順 8 を繰り返して m2 ID の名前を ProvisionedWCU に変更します。統計は [Average] (平均) に設定したままにします。



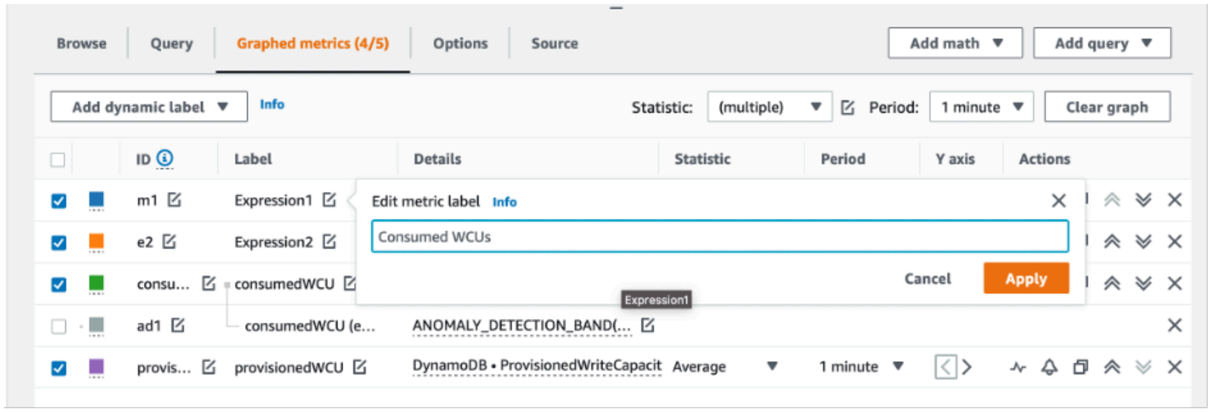
Statistic: (multiple) Period: 1 minute

☐		ID	Label	Details	Statistic	Period	Y axis	Act
☑	🟦	e1	Expression1	100*(consumedWCU/provision...				
☑	🟧	e2	Expression2	100*(consumedWCU/provision...				
☑	🟩	consu...	consumedWCU	DynamoDB • ConsumedWriteCapacity	Sum	1 minute		
☐	⬜	ad1	consumedWCU (e...	ANOMALY_DETECTION_BAND(...				
☑	🟪	provis...	ProvisionedWriteC...	DynamoDB • ProvisionedWriteCapacit	Average	1 minute		

Edit metric label Info

provisionedWCU

11. [Expression1] ラベルを選択し、値を [m1] に更新し、ラベルを [Consumed WCUs] に更新します。



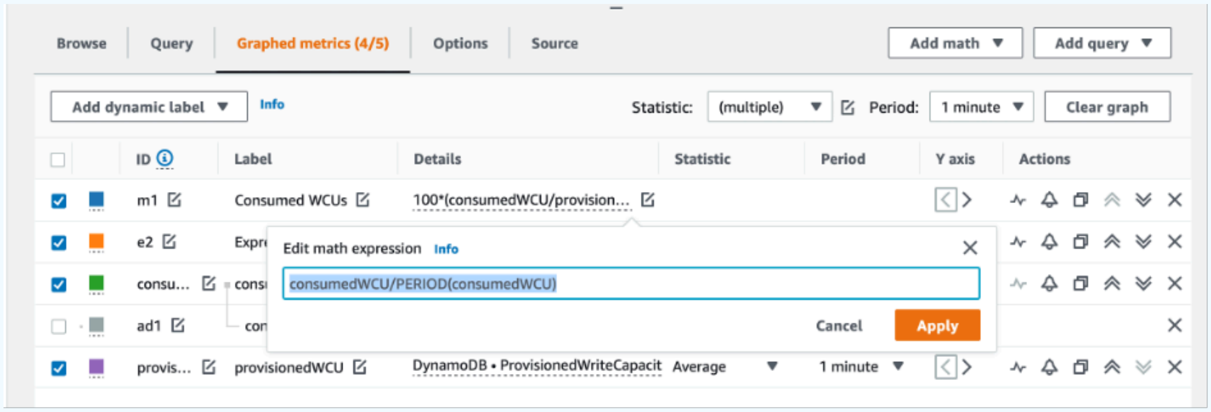
Statistic: (multiple) Period: 1 minute

☐		ID	Label	Details	Statistic	Period	Y axis	Actions
☑	🟦	m1	Expression1					
☑	🟧	e2	Expression2					
☑	🟩	consu...	consumedWCU					
☐	⬜	ad1	consumedWCU (e...	ANOMALY_DETECTION_BAND(...				
☑	🟪	provis...	provisionedWCU	DynamoDB • ProvisionedWriteCapacit	Average	1 minute		

Edit metric label Info

Consumed WCUs

> **Note**
>
> データを正しく視覚化するには、必ず [m1] (左側のチェックボックス) と [ProvisionedWCU] のみを選択してください。[Details] (詳細) をクリックし、数式を [consumedWCU/PERIOD(consumedWCU)]」に変更して、数式を更新します。このステップで別の [ANOMALY_DETECTION_BAND] メトリクスを生成することもありますが、この手順の範囲では無視できます。



Statistic: (multiple) ▼ Period: 1 minute ▼

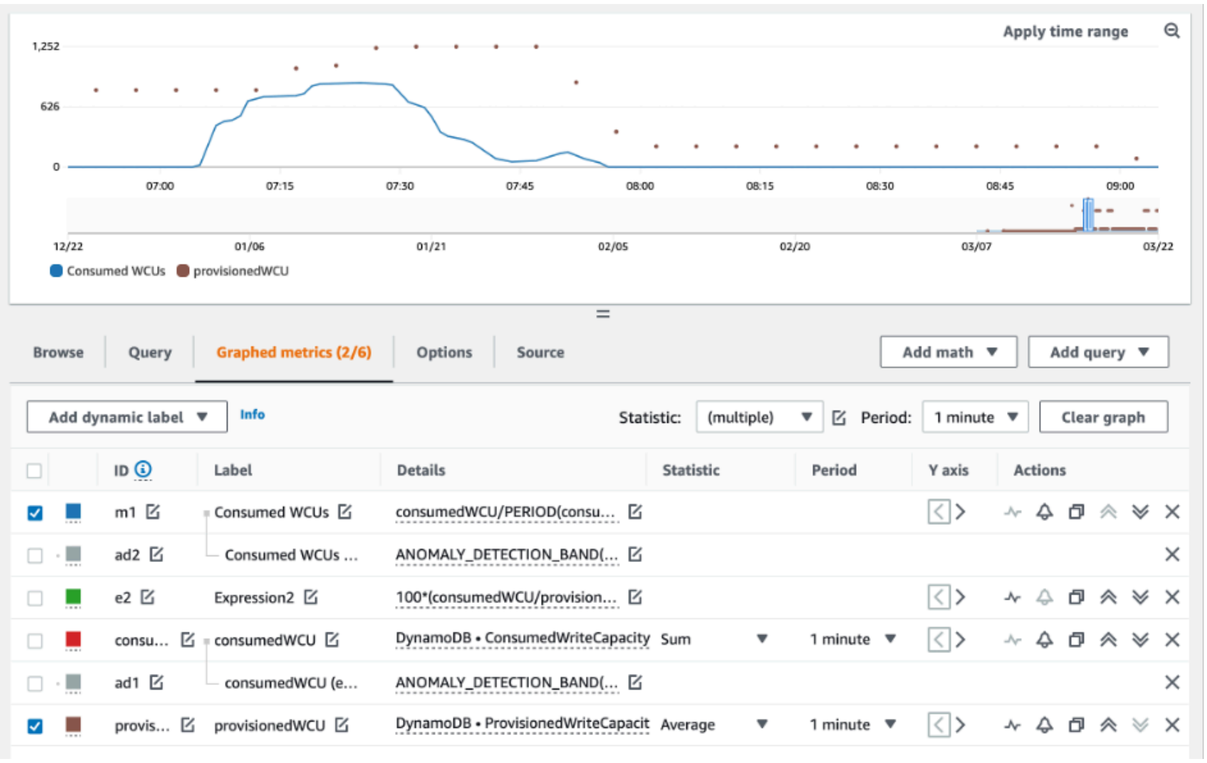
		ID	Label	Details	Statistic	Period	Y axis	Actions
☑	🟦	m1	Consumed WCUs	100*(consumedWCU/provision...				
☑	🟧	e2	Expr					
☑	🟩	consu...	cons					
□	▨	ad1	con					
☑	🟪	provis...	provisionedWCU	DynamoDB • ProvisionedWriteCapacit	Average	1 minute		

Edit math expression Info

consumedWCU/PERIOD(consumedWCU)

Cancel Apply

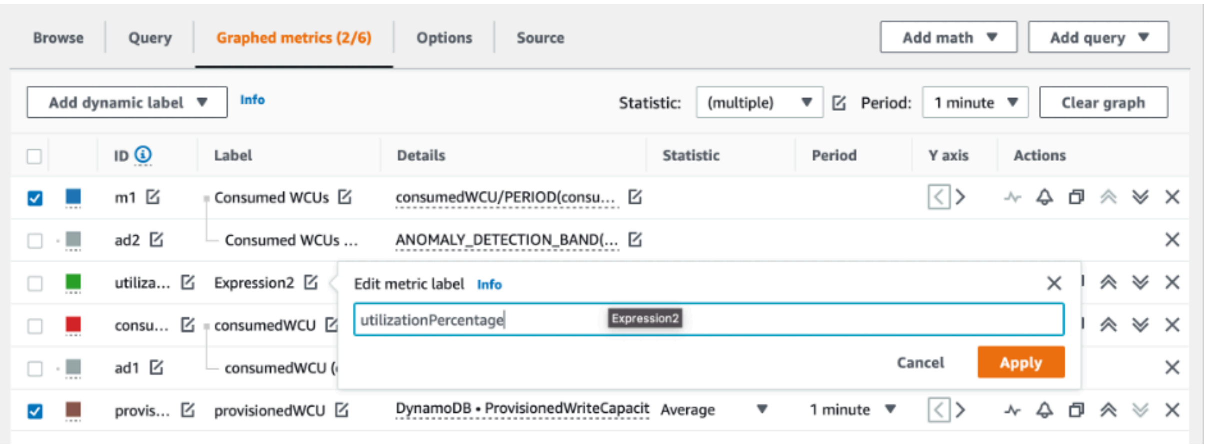
12. これで 2 つのグラフィックができたはずです。1 つはテーブル上にプロビジョニングされた WCU を示し、もう 1 つは消費された WCU を示しています。グラフィックの形状は下の図と異なる場合がありますが、参考にしてください。



Statistic: (multiple) ▼ Period: 1 minute ▼

		ID	Label	Details	Statistic	Period	Y axis	Actions
☑	🟦	m1	Consumed WCUs	consumedWCU/PERIOD(consu...				
□	▨	ad2	Consumed WCUs ...	ANOMALY_DETECTION_BAND(...				
□	🟩	e2	Expression2	100*(consumedWCU/provision...				
□	🟥	consu...	consumedWCU	DynamoDB • ConsumedWriteCapacity	Sum	1 minute		
□	▨	ad1	consumedWCU (e...	ANOMALY_DETECTION_BAND(...				
☑	🟫	provis...	provisionedWCU	DynamoDB • ProvisionedWriteCapacit	Average	1 minute		

13. Expression2 グラフィック (e2) を選択して、パーセンテージ数式を更新します。ラベルと ID の名前を utilizationPercentage に変更します。100*(m1/provisionedWCU) と一致するように数式の名前を変更します。

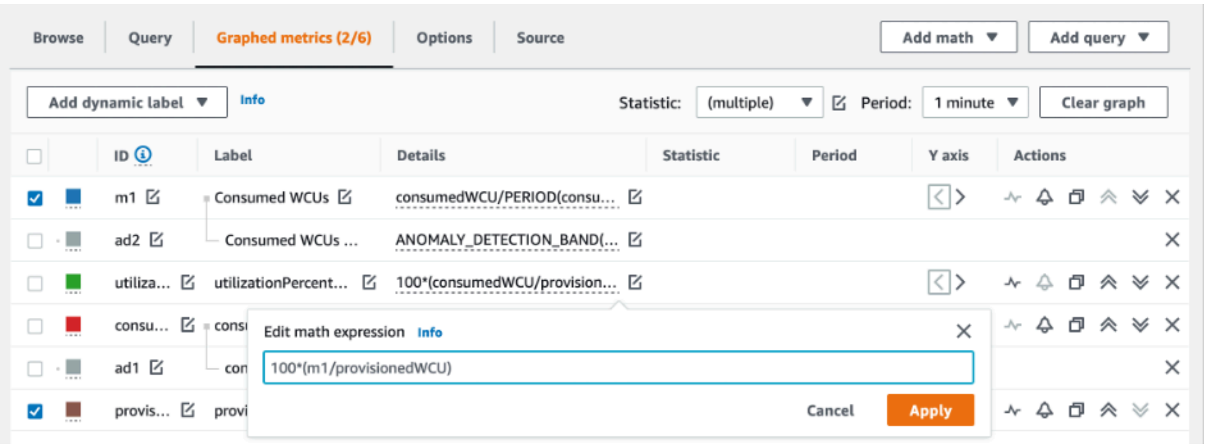


Statistic: (multiple) ▼ Period: 1 minute ▼

☐		ID	Label	Details	Statistic	Period	Y axis	Actions
☑	🟦	m1	Consumed WCUs	consumedWCU/PERIOD(consu...				
☐		ad2	Consumed WCUs ...	ANOMALY_DETECTION_BAND(...				
☐	🟩	utiliza...	Expression2					
☐	🟥	consu...	consumedWCU					
☐		ad1	consumedWCU (					
☑	🟫	provis...	provisionedWCU	DynamoDB • ProvisionedWriteCapacit	Average ▼	1 minute ▼		

Edit metric label Info

utilizationPercentage



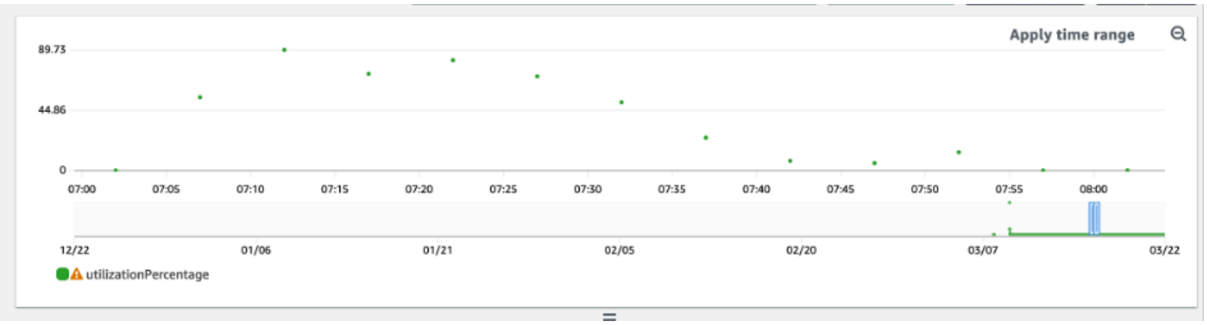
Statistic: (multiple) ▼ Period: 1 minute ▼

☐		ID	Label	Details	Statistic	Period	Y axis	Actions
☑	🟦	m1	Consumed WCUs	consumedWCU/PERIOD(consu...				
☐		ad2	Consumed WCUs ...	ANOMALY_DETECTION_BAND(...				
☐	🟩	utiliza...	utilizationPercent...	100*(consumedWCU/provision...				
☐	🟥	consu...	consı					
☐		ad1	con					
☑	🟫	provis...	provi					

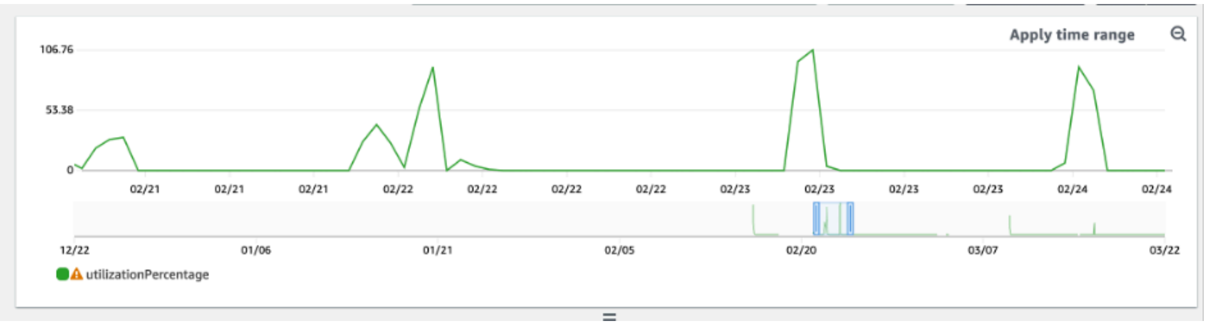
Edit math expression Info

100*(m1/provisionedWCU)

14. 使用率パターンを視覚化するには、utilizationPercentage 以外のすべてのメトリクスのチェックボックスをオフにします。デフォルトの間隔は 1 分に設定されていますが、必要に応じて自由に変更できます。



こちらは、1 時間の拡大表示と長時間の表示です。使用率が 100% を超える間隔がいくつかあることがわかりますが、この特定のワークロードでは、使用率がゼロの間隔が長くなっています。



この時点では、この例の写真とは異なる結果が得られるかもしれません。すべてはワークロードのデータ次第です。使用率が 100% を超える間隔では、スロットリングイベントが発生しやすくなります。DynamoDB にはバーストキャパシティがありますが、バーストキャパシティが完了すると、100% を超えるものはすべてスロットリングされます。

プロビジョニング不足の DynamoDB テーブルを識別する方法

ほとんどのワークロードでは、テーブルがプロビジョンドキャパシティの 80% 以上を絶えず消費している場合、そのテーブルはプロビジョニング不足と見なされます。

バーストキャパシティは DynamoDB の機能の 1 つで、お客様が当初のプロビジョニング量よりも多くの (表で定義されている 1 秒あたりのプロビジョニングスループットを超える) RCU/WCU を一時的に消費できるようにするものです。バーストキャパシティは、特別なイベントや使用量の急増によるトラフィックの急激な増加を吸収するために作成されました。このバーストキャパシティはいつまでも続くわけではありません。未使用の RCU と WCU がなくなると同時に、プロビジョニングされた容量よりも多くの容量を消費しようとすると、スロットリング状態になります。アプリケーショントラフィックの使用率が 80% に近づくと、スロットリングのリスクが大幅に高まります。

80% 使用率ルールは、データの季節性やトラフィックの増加によって異なります。次のシナリオを考えてみます。

- 過去 12 か月間、トラフィックの使用率が約 90% で安定していれば、テーブルのキャパシティは適切であると言えます
- アプリケーショントラフィックが 3 か月以内に毎月 8% の割合で増加した場合、100% に到達します
- アプリケーショントラフィックが 4 か月余りで 5% の割合で増加している場合でも、100% に到達します

上記のクエリの結果から、使用率の全体像がわかります。これらを参考にして、必要に応じてテーブルのキャパシティを増やす方法を選択するのに役立つ他のメトリクス (月間または毎週の増加率など) をさらに評価してください。運用チームと協力して、ワークロードとテーブルに適したパーセンテージを定義してください。

データを毎日または毎週分析すると、データが歪んでしまう特別なシナリオがあります。例えば、季節性のアプリケーションで、勤務時間中に使用量が急増する (ただし、勤務時間外はほぼゼロになる) 場合は、自動スケーリングをスケジュールして時間帯 (および曜日) を指定してプロビジョンドキャパシティを増やすと効果的です。繁忙期に対応できるように容量を増やすかわりに、季節性がそれほどない場合には、DynamoDB テーブルの自動スケーリング設定を利用することもできます。

> Note
>
> ベーステーブルに DynamoDB Auto Scaling 設定を作成するときは、そのテーブルに関連付けられている GSI にも別の設定を含めることを忘れないでください。

過剰にプロビジョニングされた DynamoDB テーブルを識別する方法

上記のスクリプトから取得したクエリ結果から、初期分析を実行するために必要なデータポイントが得られます。データセットの使用率が複数の間隔で 20% 未満の値を示す場合は、テーブルが過剰にプロビジョニングされている可能性があります。WCU と RCU の数を減らす必要があるかどうかをさらに明確にするには、その間隔で他の測定値を見直す必要があります。

テーブルに使用頻度の低い間隔が複数ある場合は、自動スケーリングをスケジュールするか、使用率に基づくテーブルのデフォルトの自動スケーリングポリシーを設定することで、自動スケーリングポリシーの使用から大きなメリットが得られます。

使用率が低いのにスロットリング率 (間隔内の Max(ThrottleEvents)/Min(ThrottleEvents)) が高いワークロードがある場合、この現象は、ある日 (または時間) にトラフィックが大幅に増加するが一般的にトラフィックは常に少ない、非常にスパイクの多いワークロードがある場合に発生する可能性があります。このようなシナリオでは、スケジュールされた自動スケーリングを使用すると有益な場合があります。

AWS Well-Architected フレームワークは、クラウドアーキテクトがさまざまなアプリケーションやワークロード向けに、安全で高性能、かつ回復力のある効率的なインフラストラクチャを構築するのに役立ちます。AWS Well-Architected は、運用上の優秀性、セキュリティ、信頼性、パフォーマンス効率、コスト最適化、持続可能性の 6 つの柱に基づいて構築されており、顧客とパートナーがアーキテクチャを評価し、スケーラブルな設計を実装するための一貫したアプローチを提供します。

AWS Well-Architected レンズによって、AWS Well-Architected が提供するガイダンスを特定の業界や技術分野に拡張できます。Amazon DynamoDB Well-Architected レンズでは、DynamoDB ワークロードに焦点を当てています。このレンズは、DynamoDB ワークロードを評価およびレビューするためのベストプラクティス、設計原則、質問を提供します。Amazon DynamoDB Well-Architected レンズによるレビューを完了すると、AWS Well-Architected の各柱に関連する推奨設計原則に関する教育とガイダンスを利用できるようになります。このガイダンスは、さまざまな業界、セグメント、規模、地域のお客様との作業経験に基づいています。

Well-Architected レンズによるレビューの直接的な結果として、DynamoDB ワークロードを最適化および改善するための実用的な推奨事項の概要が得られます。

Amazon DynamoDB Well-Architected レンズによるレビューの実施

DynamoDB Well-Architected レンズによるレビューは通常、AWSソリューションアーキテクトがお客様と一緒に実施しますが、お客様がセルフサービスとして実施することもできます。Amazon DynamoDB Well-Architected レンズの一環として Well-Architected の 6 本の柱をすべて確認することをお勧めしますが、まずは 1 本または複数の柱に優先的に焦点を当てでもかまいません。

Amazon DynamoDB の Well-Architected レンズによるレビューを実施するための追加情報と手順については、このビデオと DynamoDB Well-Architected Lens GitHub ページをご覧ください。

Amazon DynamoDB の Well-Architected レンズの柱

Amazon DynamoDB の Well-Architected レンズには 6 本の柱で構成されています。

パフォーマンス効率の柱

パフォーマンス効率の柱には、コンピューティングリソースを効率的に使用してシステム要件を満たし、需要の変化や技術の進歩に合わせてこの効率性を維持する能力が含まれます。

この柱の主な DynamoDB の設計原則は、データのモデル化、パーティションキーとソートキーの選択、およびアプリケーションアクセスパターンに基づくセカンダリインデックスの定義を中心に展開されています。その他の考慮事項には、ワークロードに最適なスループットモードの選択、AWS SDK によるチューニング、および必要な場合は、最適なキャッシュ戦略の使用などがあります。これらの設計原則の詳細については、DynamoDB の Well-Architected レンズのパフォーマンス効率の柱に関するこのディープダイブビデオをご覧ください。

コスト最適化の柱

コスト最適化の柱は、不必要なコストを回避することに焦点を当てています。

主なトピックは、資金の用途の把握と管理、最適なリソースタイプとその適切な数の選択、経時的な支出の分析、アプリケーション固有のアクセスパターンに応じてコストを最適化するデータモデルの設計、浪費することなくビジネスニーズを満たすためのスケーリングです。

DynamoDB の主なコスト最適化設計原則は、テーブルに最適なキャパシティモードとテーブルクラスの選択、オンデマンドキャパシティモードまたは自動スケーリングを使用するプロビジョンドキャパシティモードのいずれかを使用したキャパシティの過剰プロビジョニングの回避を中心に展開されています。その他の考慮事項には、効率的なデータモデリングとクエリによる消費される容量の削減、消費容量の一部の割引価格での予約、項目サイズの最小化、未使用のリソースの特定と削除、TTL を使用した古くなったデータの無料の自動削除などがあります。これらの設計原則の詳細については、DynamoDB の Well-Architected レンズのコスト最適化の柱に関するこのディープダイブビデオをご覧ください。

DynamoDB のコスト最適化のベストプラクティスの詳細については、「DynamoDB テーブルのコストの最適化」を参照してください。

運用上の優秀性の柱

運用上の優秀性の柱では、ビジネス価値をもたらし、プロセスと手順の継続的な向上を実現するために、システムを実行およびモニタリングすることに焦点を当てています。主なトピックは、変更の自動化、イベントへの対応、日常業務を管理するための標準の定義です。

DynamoDB の運用上の優秀性の主な設計原則は、Amazon CloudWatch および AWS Config を通じて DynamoDB メトリックスを監視すること、事前に定義されたしきい値に違反した場合や準拠していないルールが検出された場合に自動的に警告して修正することです。その他の考慮事項としては、インフラストラクチャ経由の DynamoDB リソースをコードとして定義し、タグを活用して DynamoDB リソースの整理、識別、およびコスト計算を改善することがあります。これらの設計原則の詳細については、DynamoDB の Well-Architected レンズの運用上の優秀性の柱に関するこのディープダイブビデオをご覧ください。

信頼性の柱

信頼性の柱では、期待されるときにワークロードが意図された機能を正しく一貫して実行することを重視しています。回復力のあるワークロードは、障害から迅速に復旧し、ビジネスや顧客の需要を満たします。主なトピックは、分散システムの設計、復旧計画、および変更への対応方法です。

DynamoDB の信頼性の基本的な設計原則は、RPO と RTO の要件に基づいたバックアップ戦略と保持を選択すること、DynamoDB グローバルテーブルをマルチリージョンのワークロードや低 RTO のクロスリージョンのディザスタリカバリシナリオに使用すること、AWS SDK でこれらの機能を設

定して使用することにより、アプリケーションにエクスポネンシャルバックオフを伴う再試行ロジックを実装すること、Amazon CloudWatch を通じて DynamoDB メトリックスを監視すること、および事前に定義されたしきい値が違反した場合は自動的にアラートと修正を行うことです。これらの設計原則の詳細については、DynamoDB の Well-Architected レンズの信頼性の柱に関するこのディープダイブビデオをご覧ください。

セキュリティの柱

セキュリティの柱は、情報とシステムの保護に焦点を当てています。主なトピックは、データの機密性と完全性、権限管理による誰が何を実行できるのかの特定と管理、システムの保護、セキュリティイベントを検出するための制御の確立です。

DynamoDB のセキュリティの主な設計原則は、転送中のデータを HTTPS で暗号化すること、保管中のデータを暗号化するためのキーの種類を選択すること、DynamoDB リソースに対して認証および承認を行い、きめ細かなアクセスを提供するための IAM ロールとポリシーを定義することです。その他の考慮事項は、AWS CloudTrail を通じた DynamoDB のコントロールプレーンオペレーションとデータプレーンオペレーションです。これらの設計原則の詳細については、DynamoDB の Well-Architected レンズのセキュリティの柱に関するこのディープダイブビデオをご覧ください。

DynamoDBのセキュリティの詳細については、「Amazon DynamoDB のセキュリティとコンプライアンス」を参照してください。

持続可能性の柱

持続可能性の柱は、クラウドワークロードの実行による環境に対する影響を最小限に抑えることに重点を置いています。主なトピックは、持続可能性に関する責任共有モデル、影響の把握、必要なリソースを最小限に抑えてダウンストリームへの影響を軽減するための使用率の最大化です。

DynamoDB の持続可能性の主な設計原則は、未使用の DynamoDB リソースを特定して削除すること、自動スケーリングによってオンデマンドキャパシティモードまたはプロビジョニンドキャパシティモードを使用することでオーバープロビジョニングを回避すること、消費されるキャパシティの量を減らすための効率的なクエリ、TTL を使用してデータを圧縮し、古くなったデータを削除することでストレージのフットプリントを減らすことです。これらの設計原則の詳細については、DynamoDB の Well-Architected レンズの持続可能性の柱に関するこのディープダイブビデオをご覧ください。

パーティションキーを効率的に設計し、使用するためのベストプラクティス

Amazon DynamoDB テーブルの各項目を一意に識別する主キーは、シンプル (パーティションキーのみ)、または複合 (パーティションキーとソートキーの組み合わせ) で構成できます。

一般的に言えば、テーブルとそのセカンダリインデックスの論理的なすべてのパーティションキー全体でアクティビティが均一になるようにアプリケーションを設計する必要があります。アプリケーションが必要とするアクセスパターンと、各テーブルとセカンダリインデックスが必要とする読み込みユニットと書き込みユニットを決定することができます。

デフォルトでは、テーブル内の各パーティションは、3,000 RCU と 1,000 WCU のフル容量を提供するよう努めます。テーブル内のすべてのパーティションの合計スループットは、プロビジョニングモードでプロビジョニングされたスループット、またはオンデマンドモードのテーブルレベルのスループット制限によって制約される場合があります。詳細については、「Service Quotas」を参照してください。

トピック

パーティションキーを設計してワークロードを分散する

テーブルの主キーのパーティションキー部分は、テーブルのデータが格納されている論理パーティションを決定します。これにより、基本的な物理パーティションに影響を及ぼします。I/O リクエストを効率的に分散しないパーティションキー設計では、「ホット」パーティションが作成される場合があります。これにより、スロットリングが発生することや、プロビジョニングされた I/O 容量が効率的に使用されないことがあります。

テーブルのプロビジョニングされたスループットの最適な使用は、個々の項目のワークロードパターンだけでなく、パーティションキー設計にも左右されます。この操作によって、スループットレベルを達成するためにすべてのパーティションキーバリューにアクセスしたり、アクセスされるパーティションキーバリューの割合を高くしたりする必要があるわけではありません。これは、ワークロードがアクセスするパーティションキーバリューが大きくなるほど、リクエストが分割された空間に分散されることを意味します。一般的に、パーティションキーバリューの合計数に対する、アクセスされ

たパーティションキーバリューの割合が大きくなるにつれて、プロビジョニングされたスループットをより効率的に使用することができます。

一般的なパーティションキースキーマのプロビジョニングされたスループット効率の比較を次に示します。

パーティションキーバリュー	均一性
ユーザー ID (アプリケーションに多くのユーザーが存在する場合)	良好
ステータスコード (可能性のあるステータスコードが少しだけある場合)	不良
項目の作成日。直近の期間 (日、時、分など) に切り上げられます。	不良
デバイス ID (各デバイスが比較的類似した間隔でデータにアクセスする場合)。	良好
デバイス ID (追跡中のデバイスが多数あり、そのうちの 1 つのデバイスが他のすべてのデバイスよりも頻繁に使用される場合)	不良

単一のテーブル内にあるパーティションキーバリューの数が少ない場合は、より明確に区別できるパーティションキーバリュー全体を対象として、書き込みオペレーションを分散することを検討してください。つまり、「ホット」パーティションキーバリュー (何度もリクエストされるパーティションキーバリュー) が原因で全体のパフォーマンス速度が低下する場合があるので、このようなパーティションキーバリューを回避するようにプライマリキー要素を構築してください。

例えば、複合プライマリキーを持つテーブルがあるとします。パーティションキーは項目の作成日を表します (直近の日付に切り上げられます)。ソートキーは項目の識別子を表します。特定の日付 (`2014-07-09` など) で、すべての新しい項目が 1 つのパーティションキーバリュー (およびその物理パーティション) として書き込まれます。

テーブル全体が単一のパーティションに収まり (時間の経過に伴うデータの増加を考慮)、アプリケーションで必要となる読み込みスループットと書き込みスループットが単一のパーティションにおける読み込みと書き込みの容量を超過しない場合は、パーティション分割をしても、アプリケーションが予想外のスロットリングを受けることはありません。

NoSQL Workbench for DynamoDB を使用してパーティションキー設計を視覚化する方法については、「NoSQL Workbench を使用したデータモデルの構築」を参照してください。

書き込みシャーディングを使用してワークロードを均等に分散させる

Amazon DynamoDB のパーティションキースペースで効率的な書き込みの分散を行う方法として、スペースの拡張があります。これにはさまざまな方法があります。パーティションキーバリューに乱数を追加して、パーティション間で項目を分散させることができます。または、クエリ対象に基づいて計算された数値を使用することができます。

ランダムなサフィックスを使用したシャーディング

パーティションキースペース全体に均等にロードを分散するための方法として、パーティションキーバリューの最後に乱数を追加する方法があります。次に、より大きなスペース全体の書き込みをランダム化します。

例えば、今日の日付を表すパーティションキーの場合は、1 と 200 の間の乱数を選択し、それをサフィックスとして日付に連結することができます。これにより、2014-07-09.1 や 2014-07-09.2 などのパーティションキーバリューが生成されます。この場合、最大のパーティションキーバリューは 2014-07-09.200 になります。パーティションキーをランダム化しているため、毎日のテーブルへの書き込みは複数のパーティション間で均等に分散されます。これにより、並列性が強化され、全体的なスループットが向上します。

ただし、特定の日のすべての項目を読み込むには、すべてのサフィックスの項目をクエリしてから結果をマージする必要があります。例えば、最初にパーティションキーバリュー Query に対して 2014-07-09.1 リクエストを発行します。次に 2014-07-09.2 に対して別の Query を発行します。この処理を 2014-07-09.200 まで繰り返します。最終的には、アプリケーションですべての Query リクエストの結果をマージする必要があります。

計算されたサフィックスを使用したシャーディング

ランダム化の方法は、書き込みスループットを大幅に向上させることができます。ただし、項目を書き込むときに使用されたサフィックスの値がわからないため、特定の項目を読み込むのは困難です。別の方法を使用して個々の項目の読み込みを容易にすることができます。パーティション間で項目を分散させるには、乱数を使用せずに、クエリする項目に基づいて計算できる数値を使用します。

前の例では、パーティションキーで今日の日付が使用されています。各項目にはアクセス可能な OrderId 属性があり、日付だけでなく、順序 ID を使用して、項目を最も頻繁に見つける必要があるとします。アプリケーションから項目をテーブルに書き込む前に、この順序 ID に基づいてハッ

シュサフィックスを計算し、その値をパーティションキーの日付に追加します。この計算では、1 から 200 までの数値が生成され、ランダムな方法と同様に、完全に均一に分散されます。

この計算はシンプルな計算で十分です (順序 ID の文字の UTF-8 コードポイント値を乗算して、それを 200 で割った余りに + 1 するなど)。パーティションキーバリューは、日付に計算結果を連結した値になります。

この方法を使用すると、書き込みがパーティションキーバリュー全体に均一に分散されます。したがって、物理パーティション全体にも均一に分散されます。これで、特定の GetItem 値のパーティションキーバリューを計算できるため、OrderId オペレーションを実行して、特定の項目や日付を簡単に読み込むことができます。

特定の日のすべての項目を読み込むには、2014-07-09.N キー (ここで N は 1 〜 200) をQueryする必要があり、アプリケーションはすべての結果をマージする必要があります。この方法には、すべてのワークロードを利用する単一の「ホット」パーティションキーバリューを使用しないというメリットがあります。

ⓘ Note

大容量の時系列データを処理するために特別に設計されている効率的な方法については、「時系列データ」を参照してください。

データアップロード中の書き込みアクティビティの効率的な分散

通常、他のデータソースからデータをロードする場合、Amazon DynamoDB がテーブルデータを複数のサーバーでパーティション化します。割り当てられたすべてのサーバーに同時にデータをアップロードするとパフォーマンスが向上します。

例えば、パーティションキーとして UserID とソートキーとして MessageID を含む複合プライマリキーを使用する DynamoDB テーブルにユーザーメッセージをアップロードするとします。

データをアップロードする場合、1 つのアプローチとして、ユーザーごとにすべてのメッセージ項目を別々にアップロードできます。

UserId	MessageID
U1	1

UserId	MessageID
U1	2
U1	...
U1	... 最大 100
U2	1
U2	2
U2	...
U2	... 最大 200

この場合の問題は、パーティションキーバリュー全体で DynamoDB への書き込みリクエストを分散していないことです。一度に 1 つのパーティションキーバリューを取得し、そのすべての項目をアップロードした後に、次のパーティションキーバリューに移動して同じ処理が行われます。

その裏では、DynamoDB がテーブル内のデータを複数のサーバーでパーティション化しています。テーブルにプロビジョニングされたすべてのスループットキャパシティを完全に使用するには、ワークロードをパーティションキーバリューに分散する必要があります。同じパーティションキーバリューを持つ項目に対して不均等な量のアップロード作業を指示すると、DynamoDB がテーブル用にプロビジョニングしたすべてのリソースを完全に使用することはできません。

アップロード作業を分散するには、ソートキーを使用して各パーティションキーバリューから 1 つの項目をロードし、次に各パーティションキーバリューから別の項目をロードします。

UserId	MessageID
U1	1
U2	1
U3	1
...	...
U1	2

UserId	MessageID
U2	2
U3	2
...	...

このシーケンスの各アップロードでは、それぞれ異なるパーティションキーバリューが使用されるので、多くの DynamoDB サーバーが同時にビジー状態になり、スループットパフォーマンスが向上します。

ソートキーを使用してデータを整理するためのベストプラクティス

Amazon DynamoDB テーブルでは、テーブルの各アイテムを一意に識別するプライマリキーは、パーティションキーとソートキーで構成することができます。

設計が優れたソートキーには、2 つの主な利点があります。

- 関連情報を 1 つの場所にまとめて、効率的にクエリを実行することができます。ソートキーを慎重に設計することで、begins_with、between、>、< などの演算子による範囲のクエリを使用して、一般的に必要な関連項目のグループを検索することができます。
- 複合ソートキーを作成すれば、データの階層的 (1 対多) な関係を定義して、任意の階層レベルでクエリを実行することができます。

  例えば、地理的場所を示すテーブルでは、次のようにソートキーを構成できます。

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

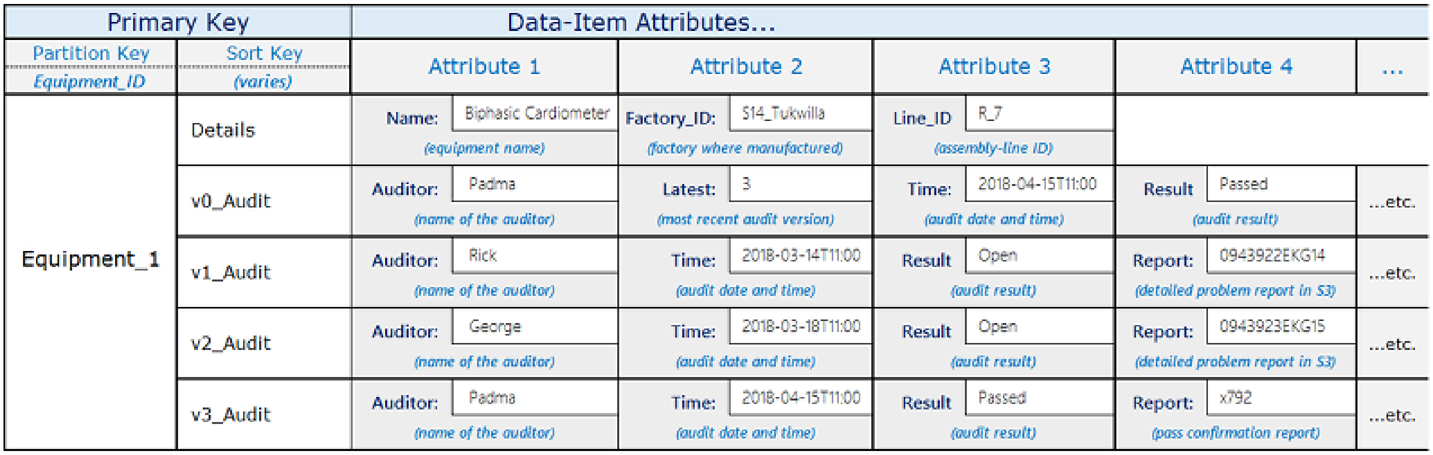
  これにより、これらの集計レベルのいずれかの場所のリスト (country から neighborhood、またその間にあるすべてのもの) に対して、範囲のクエリを効率的に行うことができます。

バージョンコントロールにソートキーを使用する

多くのアプリケーションでは、監査またはコンプライアンス目的で項目レベルのリビジョン履歴を保持し、最新バージョンを簡単に取得できる必要があります。ソートキープレフィックスを使用してこれを実現する効果的な設計パターンがあります。

- 新しい項目ごとに、コピーを 2 つ作成します。一方は、ソートキーの冒頭にバージョン番号 0 のプレフィックス (v0_ など) を、もう一方には、バージョン番号 1 のプレフィックス (v1_ など) を付加します。
- 項目が更新されるたびに、更新されたバージョンのソートキーの次の上位バージョンプレフィックスを使用して、更新後の内容をバージョンのプレフィックス 0 の項目にコピーします。つまり、任意の項目の最新バージョンは、0 のプレフィックスを使用して簡単に見つけることができます。

例えば、部品メーカーは、以下に示すようなスキーマを使用する場合があります。



Primary Key		Data-Item Attributes...				
Partition Key *Equipment_ID*	Sort Key *(varies)*	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...
Equipment_1	Details	Name: Biphasic Cardiometer *(equipment name)*	Factory_ID: S14_Tukwilla *(factory where manufactured)*	Line_ID R_7 *(assembly-line ID)*		
	v0_Audit	Auditor: Padma *(name of the auditor)*	Latest: 3 *(most recent audit version)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	...etc.
	v1_Audit	Auditor: Rick *(name of the auditor)*	Time: 2018-03-14T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943922EKG14 *(detailed problem report in S3)*	...etc.
	v2_Audit	Auditor: George *(name of the auditor)*	Time: 2018-03-18T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943923EKG15 *(detailed problem report in S3)*	...etc.
	v3_Audit	Auditor: Padma *(name of the auditor)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	Report: x792 *(pass confirmation report)*	...etc.

Equipment_1 項目は、さまざまな監査人によって、一連の監査が行われます。新しい各監査の結果は、テーブルの新しい項目に取り込まれます。バージョン番号 1 から始まり、その後のリビジョンごとに番号が増えます。

新しいリビジョンが追加されるたびに、アプリケーション層でゼロバージョンの項目 (v0_Audit のソートキー) の内容が、新しいリビジョンの内容に置き換えられます。

アプリケーションで最新の監査ステータスを取得する必要がある場合は、ソートキープレフィックス v0_ をクエリできます。

アプリケーションでリビジョン履歴全体を取得する必要がある場合は、その項目のパーティションキーの下にあるすべての項目をクエリして、v0_ 項目をフィルターで除外することができます。

この設計では、ソートキープレフィックスの後にソートキーの各部品 ID が含まれている場合、機器の複数の部分を監査することもできます。

DynamoDB でセカンダリインデックスを使用するためのベストプラクティス。

多くの場合、セカンダリインデックスは、アプリケーションに必要なクエリパターンをサポートするために必要です。同時に、セカンダリインデックスを過度に使用するか、非効率的に使用すると、コストがかかり、不要にパフォーマンスが低下する可能性があります。

目次

DynamoDB のセカンダリインデックスの一般的なガイドライン

Amazon DynamoDB では、次の 2 種類のセカンダリインデックスをサポートしています。

- グローバルセカンダリインデックス (GSI) — パーティションキーおよびソートキーを持つインデックス。ベーステーブルのものとは異なる場合があります。このインデックスに関するクエリがすべてのパーティションにまたがり、ベーステーブル内のすべてのデータを対象とする可能性があるため、グローバルセカンダリインデックスは「グローバル」と見なされます。グローバルセカンダリインデックスにはサイズの制限はなく、読み込みアクティビティと書き込みアクティビティ用にプロビジョニングされた独自のスループット設定がテーブルのものとは異なります。
- ローカルセカンダリインデックス (LSI) - パーティションキーはベーステーブルと同じですが、ソートキーが異なるインデックスです。ローカルセカンダリインデックスは、ローカルセカンダリインデックスのすべてのパーティションの範囲が同じパーティションキーバリューを持つベーステーブルのパーティションに限定されるという意味で「ローカル」です。その結果、任意の 1 つ

のパーティションキーバリューに対してインデックスが作成された項目の合計サイズが、10 GB を超えることはありません。また、ローカルセカンダリインデックスでは、読み込みアクティビティおよび書き込みアクティビティのプロビジョニングされたスループット設定が、インデックス作成中のテーブルと共有されます。

DynamoDB の各テーブルには、最大で 20 のグローバルセカンダリインデックス (デフォルトのクォータ) と 5 つのローカルセカンダリインデックスを持つことができます。

多くの場合、グローバルセカンダリインデックスはローカルセカンダリインデックスよりも便利です。どのタイプのインデックスを使用するかの決定は、アプリケーションの要件によっても異なります。グローバルセカンダリインデックスとローカルセカンダリインデックスの比較および、どちらを選ぶかの詳細については、「the section called “インデックスの使用”」を参照してください。

DynamoDB でインデックスを作成する際に留意すべき一般的な原則と設計パターンは次のとおりです。

トピック

インデックスを効率的に使用する

インデックス数は最小限に抑えます。頻繁にクエリを行わない属性では、セカンダリインデックスを作成しないようにします。ほとんど使用されていないインデックスは、ストレージおよび I/O のコスト増大の一因になり、アプリケーションのパフォーマンスには効果がありません。

計画を慎重に選択する

セカンダリインデックスはストレージとプロビジョニング済みのスループットを消費するため、インデックスのサイズは可能な限り小さくすべきです。また、インデックスが小さいほど、テーブル全体に対してクエリを行うのに比べてパフォーマンスが向上します。使用するクエリが属性の一部しか返さないことが多く、それらの属性のサイズを合計しても項目全体より大幅に小さい場合には、頻繁にリクエストを行う属性だけを対象とするようにします。

読み込みに比べて、テーブルでの書き込みアクティビティが多くなることが予想される場合には、次のベストプラクティスに従います。

- インデックスに書き込まれる項目のサイズが最小になるように、計画される属性が少なくなるようにします。ただし、計画される属性のサイズが単一の書き込み容量ユニット (1 KB) より大きい場合にのみ適用されます。例えば、インデックスエントリのサイズが 200 バイトである場合、DynamoDB ではこれが 1 KB に切り上げられます。言い換えれば、インデックス項目のサイズが小さい間は、追加コストが発生することなく、より多くの属性を計画することができます。
- クエリではほとんど必要にならないことが分かっている属性は計画しないでください。インデックスで計画されている属性を更新する度に、インデックス更新による追加コストが発生します。プロビジョニングされたスループットの高いコストでは、`Query` では計画されない属性を引き続き検索できますが、クエリのコストは頻繁にインデックスを更新するコストよりも大幅に低くなる可能性があります。
- ALL は、異なるソートキーによってソートされるテーブル項目全体を返るようにする場合にのみ指定します。すべての属性を計画することでテーブルをフェッチする必要がなくなりますが、ほとんどの場合、ストレージおよび書き込みアクティビティに要するコストが倍加します。

次のセクションで説明するように、フェッチを最小限に抑える必要性に応じて、インデックスをできるだけ小さくします。

フェッチを回避するための頻繁なクエリの最適化

レイテンシーを可能な限り小さくしてクエリを最速にするには、クエリによって返ることが予想されるすべての属性を計画します。特に、計画されていない属性のローカルセカンダリインデックスにクエリを実行すると、DynamoDB は自動的にテーブルからこれらの属性を取得します。そのため、項目全体をテーブルから読み取る必要があります。これにより、レイテンシーと不要な I/O オペレーションを減らすことができます。

「不定期な」クエリは、「必須」のクエリに変化することがあります。不定期にのみ、クエリを実行することが予想されるために計画しない属性がある場合は、状況が変わる可能性があるかどうかを検討します。

テーブルのフェッチの詳細については、「ローカルセカンダリインデックスに対するプロビジョニングされたスループットに関する考慮事項」を参照してください。

ローカルセカンダリインデックス作成時に項目コレクションのサイズ制限に注意する

項目コレクションとは、テーブルとそのローカルセカンダリインデックス内で、同じパーティションキーを持つすべての項目を意味します。10 GB を超えることができる項目コレクションはないため、パーティションキーバリューによっては容量が不足する可能性があります。

テーブル項目を追加または更新すると、DynamoDB は影響を受けるローカルセカンダリインデックスをすべて更新します。インデックスが付けられた属性がテーブル内で定義されている場合は、ローカルセカンダリインデックスも増加します。

ローカルセカンダリインデックスを作成する場合は、インデックスに書き込まれるデータの量と、および同じパーティションキーバリューを含むデータ項目数を考慮します。特定のパーティションキーバリューに対するテーブルおよびインデックス項目の合計が 10 GB を超えると予想される場合は、そのインデックスの作成を回避できないかどうかを検討してください。

ローカルセカンダリインデックスの作成を回避できない場合は、項目コレクションのサイズ制限を超える前に対処する必要があります。ベストプラクティスとして、10 GB のサイズ制限に近づいたアイテムコレクションのサイズを監視して警告するアイテムを書き込む際は、`ReturnItemCollectionMetrics` パラメータを使用します。アイテムコレクションの最大サイズを超えると、書き込みが失敗します。アプリケーションに影響が出る前にアイテムコレクションのサイズを監視してアラートを出すことで、アイテムコレクションのサイズに関する問題を軽減できます。

> **Note**
>
> ローカルセカンダリインデックスは、一度作成すると削除できません。

制限の範囲内での作業と是正措置のための方法については、「項目コレクションのサイズ制限」を参照してください。

スパースインデックスの利用

テーブル内の項目について、DynamoDB は項目内にインデックスソートのキーバリューがある場合のみ、対応するインデックスエントリを書き込みます。ソートキーがすべてのテーブル項目に表示されない場合や、インデックスパーティションキーが項目に存在しない場合、インデックスはスパースであると言います。

スパースインデックスは、テーブルの小さなサブセクションに対して行うクエリに役立ちます。例えば、次のキー属性を使用して顧客の注文をすべて格納するテーブルがあるとします。

- パーティションキー: `CustomerId`
- ソートキー: `OrderId`

クローズされていない注文を追跡するために、未出荷の注文項目に isOpen というブール属性を挿入することができます。その後、注文が出荷されたら、その属性は削除できます。CustomerId (パーティションキー) と isOpen (ソートキー) でインデックスを作成した場合は、isOpen が定義されている注文のみ、その中に表示されます。多くの注文のうち、少数の注文がクローズされていない場合は、テーブル全体をスキャンするよりも、クローズされていない注文のインデックスをクエリする方が時間もコストもかかりません。

isOpen のようなタイプの属性を使用する代わりに、インデックスに有益なソート順になる値を持つ属性を使用できます。例えば、各注文時の日付に設定された OrderOpenDate 属性を使用して、注文が確定した後でそれを削除することができます。このように、スパースなインデックスをクエリすると、各注文の確定日でソートされた項目が返されます。

DynamoDB のスパースインデックスの例

グローバルセカンダリインデックスは、デフォルトでスパースです。グローバルセカンダリインデックスを作成する際、パーティションキーおよびソートキー (オプション) を指定します。インデックスには、これらの属性を含むベーステーブル内の項目のみ、表示されます。

グローバルセカンダリインデックスをスパースに設計することにより、優れたパフォーマンスを達成しながら、ベーステーブルの書き込みスループットよりも低くプロビジョニングできます。

例えば、ゲームアプリケーションは、各ユーザーのすべてのスコアを追跡することができますが、一般的に、いくつかの高いスコアをクエリするだけで済みます。このシナリオは、次のように効率的に処理されます。

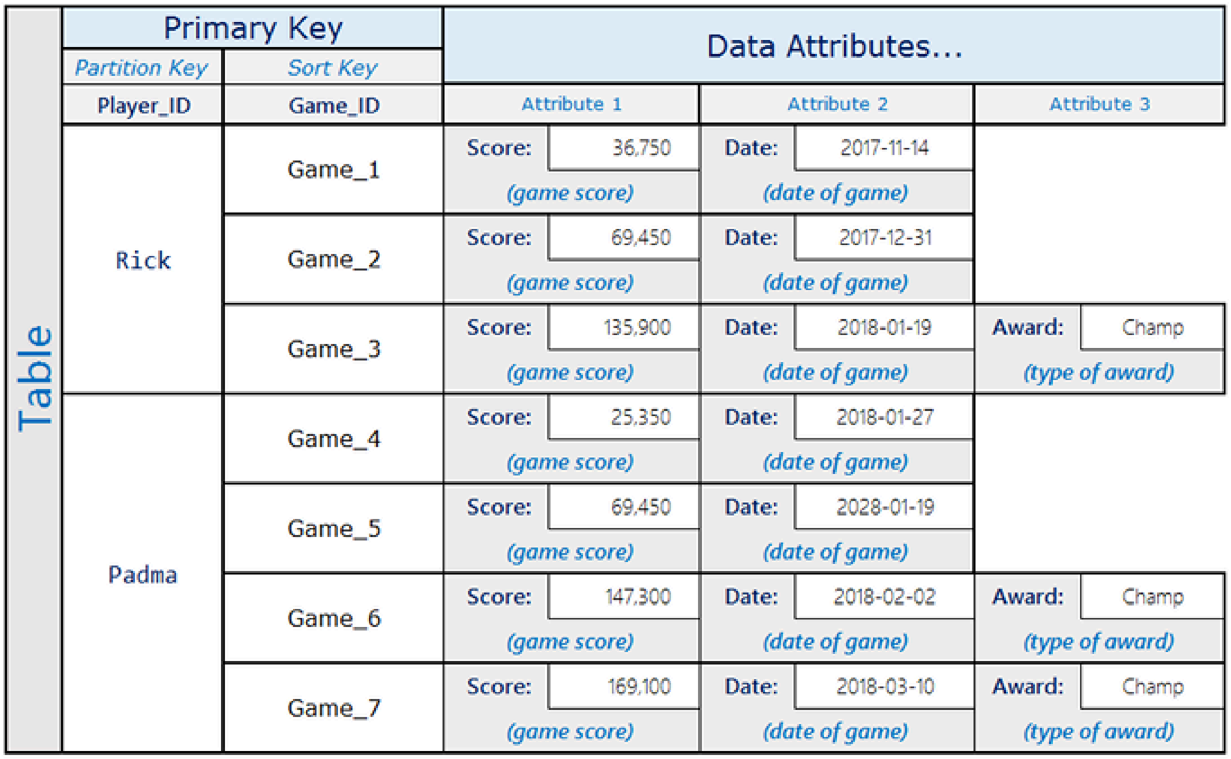
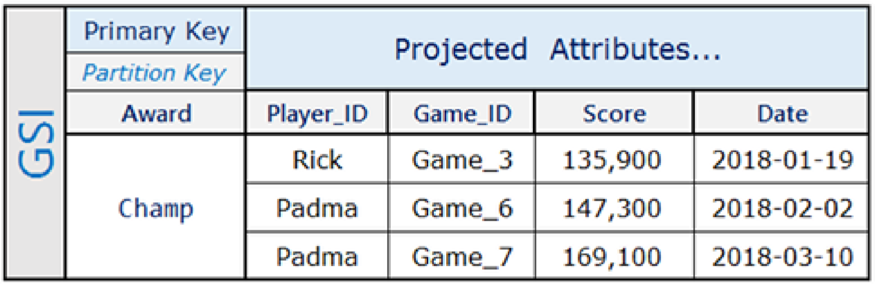


Table	Primary Key		Data Attributes...		
	Partition Key	Sort Key			
	Player_ID	Game_ID	Attribute 1	Attribute 2	Attribute 3
	Rick	Game_1	Score: 36,750 (game score)	Date: 2017-11-14 (date of game)	
		Game_2	Score: 69,450 (game score)	Date: 2017-12-31 (date of game)	
		Game_3	Score: 135,900 (game score)	Date: 2018-01-19 (date of game)	Award: Champ (type of award)
	Padma	Game_4	Score: 25,350 (game score)	Date: 2018-01-27 (date of game)	
		Game_5	Score: 69,450 (game score)	Date: 2028-01-19 (date of game)	
		Game_6	Score: 147,300 (game score)	Date: 2018-02-02 (date of game)	Award: Champ (type of award)
		Game_7	Score: 169,100 (game score)	Date: 2018-03-10 (date of game)	Award: Champ (type of award)

ここで、Rick は 3 つのゲームをプレイし、そのうちの 1 つで Champ ステータスを達成しました。Padma は 4 つのゲームをプレイし、そのうちの 2 つで Champ ステータスを達成しました。Award 属性は、ユーザーが賞を獲得した項目にのみ存在することに注意してください。関連するグローバルセカンダリインデックスは次のようになります。



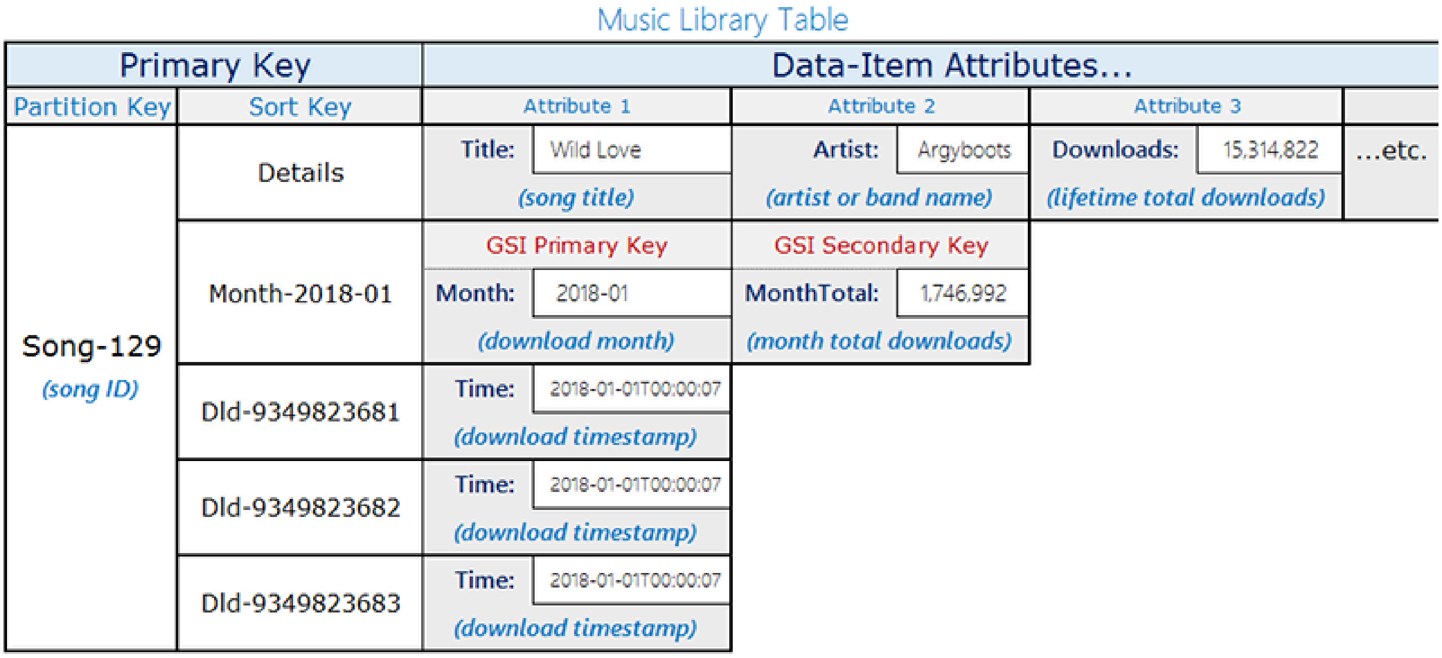
GSI	Primary Key	Projected Attributes...			
	Partition Key				
	Award	Player_ID	Game_ID	Score	Date
	Champ	Rick	Game_3	135,900	2018-01-19
		Padma	Game_6	147,300	2018-02-02
		Padma	Game_7	169,100	2018-03-10

グローバルセカンダリインデックスには、頻繁にクエリされる上位スコアのみ含まれます。これらのスコアは、ベーステーブルの項目の小さなサブセットです。

マテリアライズされた集計クエリでグローバルセカンダリインデックスを使用する

急速に変化するデータ上で、ほぼリアルタイムの集約や主要メトリクスを維持することは、企業が意思決定を迅速に行う上で、ますます価値が高まりつつあります。例えば、音楽ライブラリでは、最も多くダウンロードされた曲をほぼリアルタイムに紹介する場合があります。

次の音楽ライブラリのテーブルレイアウトを考えてみましょう。



Music Library Table

Primary Key		Data-Item Attributes...			
Partition Key	Sort Key	Attribute 1	Attribute 2	Attribute 3	
Song-129 (song ID)	Details	Title: Wild Love (song title)	Artist: Argyboots (artist or band name)	Downloads: 15,314,822 (lifetime total downloads)	...etc.
	Month-2018-01	GSI Primary Key Month: 2018-01 (download month)	GSI Secondary Key MonthTotal: 1,746,992 (month total downloads)		
	DId-9349823681	Time: 2018-01-01T00:00:07 (download timestamp)			
	DId-9349823682	Time: 2018-01-01T00:00:07 (download timestamp)			
	DId-9349823683	Time: 2018-01-01T00:00:07 (download timestamp)			

このテーブルの例では、songID をパーティションキーとして使用して曲を保存しています。このテーブルで Amazon DynamoDB Streams を有効にして、Lambda 関数をこのストリームにアタッチすることで、各曲がダウンロードされたときに、Partition-Key=SongID と Sort-Key=DownloadID を含むテーブルにエントリが追加されます。これらの更新が行われると、DynamoDB Streams で Lambda 関数がトリガーされます。この Lambda 関数を使用して、songID ごとにダウンロードを集計してグループ化し、上位レベルの項目 Partition-Key=songID と Sort-Key=Month を更新します。新しい集計値を書き込んだ直後に Lambda 実行が失敗した場合は、再試行して値を複数回集計し、おおよその値を残す可能性があることに注意してください。

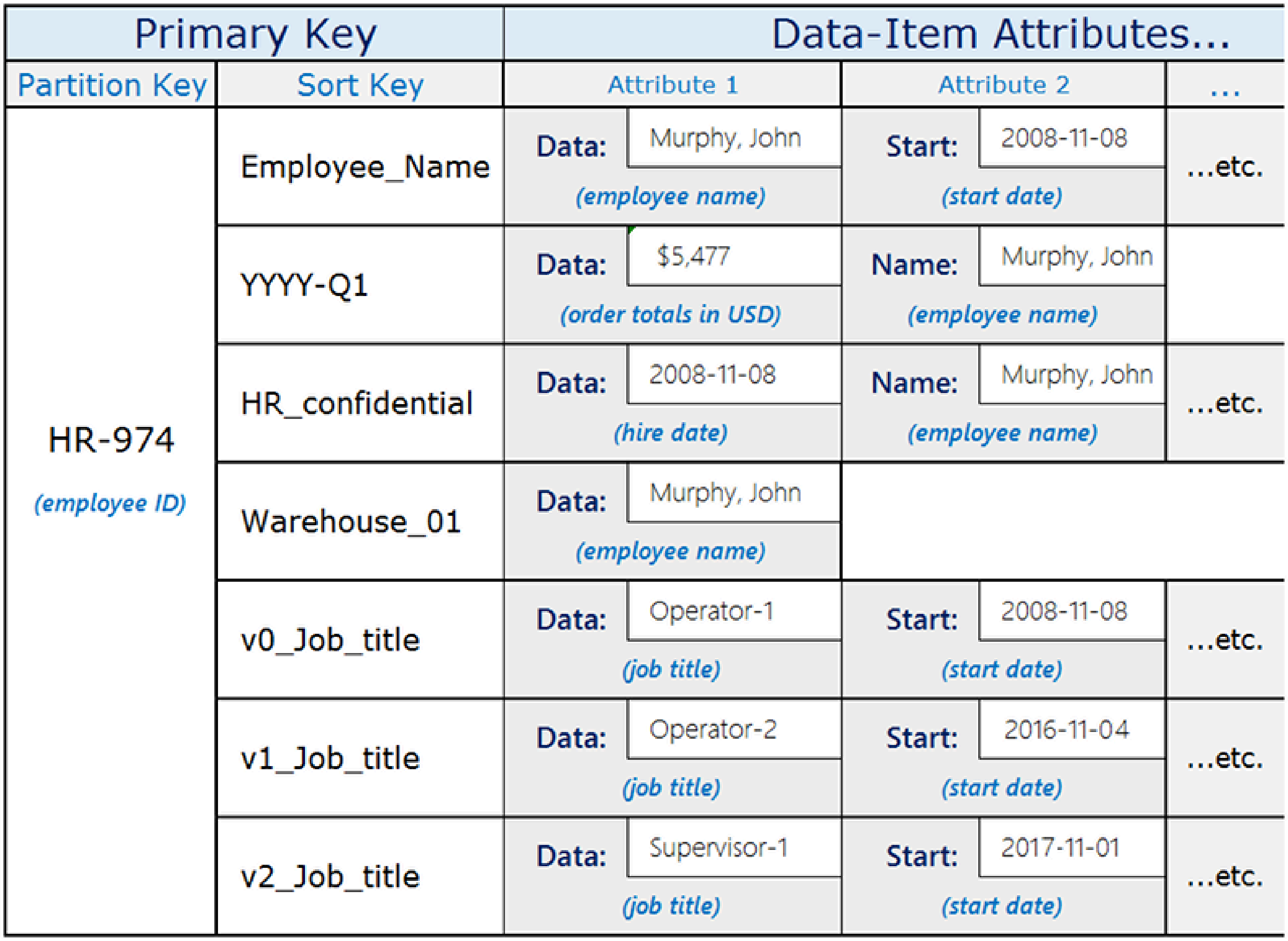
1 桁台のミリ秒のレイテンシーでほぼリアルタイムに更新を読み取るには、クエリ条件 Month=2018-01、ScanIndexForward=False、Limit=1 のグローバルセカンダリインデックスを使用します。

ここでは、グローバルセカンダリインデックスがスパースインデックスであり、リアルタイムにデータを取得するために照会する必要のある項目に対してのみ使用できる、もう 1 つのキー最適化を使用します。グローバルセカンダリインデックスは、人気のあった上位 10 曲またはその月にダウンロードされた曲に関する情報を必要とする追加のワークフローに対応できます。

グローバルセカンダリインデックスの多重定義

Amazon DynamoDB には、テーブルあたりの 20 グローバルセカンダリインデックスのデフォルトのクォータがありますが、実際には 20 以上のデータフィールドでインデックスを作成できます。スキーマが均一であるリレーショナルデータベース管理システム (RDBMS) のテーブルとは対照的に、DynamoDB のテーブルでは、一度に多数の異なるタイプのデータ項目を保持することができます。さらに、異なる項目に含まれている同じ属性に、まったく異なる種類の情報を含めることができます。

さまざまな種類のデータを保存する DynamoDB テーブルレイアウトの次の例を考えてみましょう。



Primary Key		Data-Item Attributes...		
Partition Key	Sort Key	Attribute 1	Attribute 2	...
HR-974 (employee ID)	Employee_Name	Data: Murphy, John (employee name)	Start: 2008-11-08 (start date)	...etc.
	YYYY-Q1	Data: $5,477 (order totals in USD)	Name: Murphy, John (employee name)	
	HR_confidential	Data: 2008-11-08 (hire date)	Name: Murphy, John (employee name)	...etc.
	Warehouse_01	Data: Murphy, John (employee name)		
	v0_Job_title	Data: Operator-1 (job title)	Start: 2008-11-08 (start date)	...etc.
	v1_Job_title	Data: Operator-2 (job title)	Start: 2016-11-04 (start date)	...etc.
	v2_Job_title	Data: Supervisor-1 (job title)	Start: 2017-11-01 (start date)	...etc.

Data 属性。すべての項目に共通ですが、親項目に応じて内容が異なります。テーブルのソートキーをパーティションキーとして使用し、Data 属性をソートキーとして使用するテーブルのグローバルセカンダリインデックスを作成した場合、その 1 つのグローバルセカンダリインデックスを使用してさまざまな異なるクエリを作成できます。例えば、次のようなクエリがあります。

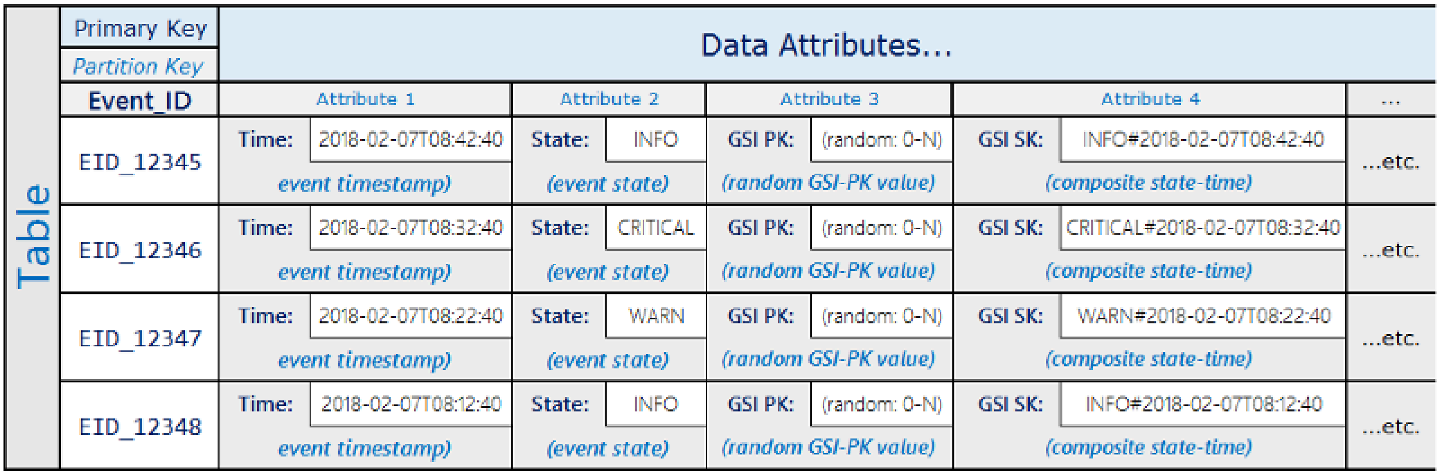
- Employee_Name をパーティションキーバリューとして使用し、従業員の名前 (Murphy, John など) をソートキーバリューとして使用し、グローバルセカンダリインデックスの名前で従業員を検索します。
- グローバルセカンダリインデックスを使用して、倉庫 ID (Warehouse_01 など) を検索して特定の倉庫で作業しているすべての従業員を検索します。
- 最新の採用者のリストを取得して、HR_confidential のグローバルセカンダリインデックスをパーティションキーバリューとして使用し、日付範囲をソートキーバリューとして使用してクエリします。

選択テーブルクエリでグローバルセカンダリインデックス書き込みシャーディングを使用する

アプリケーションは、Amazon DynamoDB テーブル内の特定の条件を満たす項目の小さなサブセットを識別する必要があることがあります。これらの項目がテーブルのパーティションキー全体に無作為に分散されている場合は、テーブルスキャンを使用してそれらを取得できます。このオプションは高価になることもありますが、テーブル上の多数の項目が検索条件を満たす場合にはよく機能します。しかし、キースペースが大きく、検索条件が非常に選択的である場合、この戦略は、多くの不要な処理を引き起こす可能性があります。

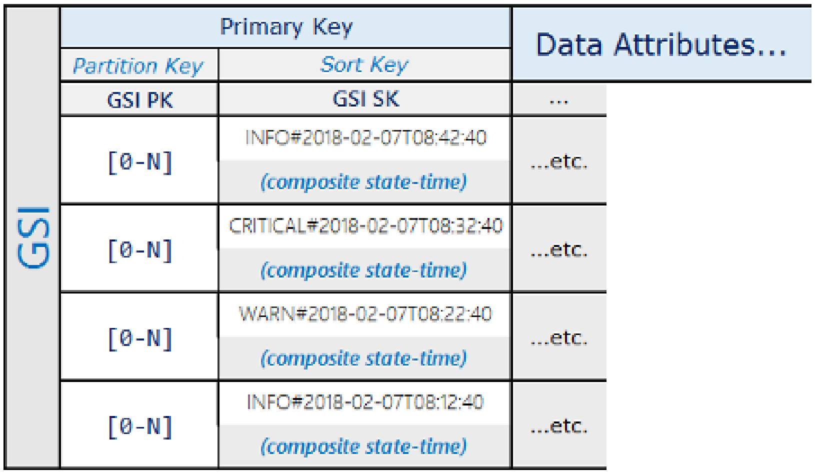
より良い解決策は、データをクエリすることです。キースペース全体で選択クエリを有効にするには、グローバルセカンダリインデックスのパーティションキーに使用する各項目に (0-N) 値を含む属性を追加して、書き込みシャーディングを使用できます。

以下に示しているのは、これを Critical-Event ワークフローで使用するスキーマの例です。



Table

Primary Key *Partition Key*	Data Attributes...				
Event_ID	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...
EID_12345	Time: 2018-02-07T08:42:40 *event timestamp)*	State: INFO *(event state)*	GSI PK: (random: 0-N) *(random GSI-PK value)*	GSI SK: INFO#2018-02-07T08:42:40 *(composite state-time)*	...etc.
EID_12346	Time: 2018-02-07T08:32:40 *event timestamp)*	State: CRITICAL *(event state)*	GSI PK: (random: 0-N) *(random GSI-PK value)*	GSI SK: CRITICAL#2018-02-07T08:32:40 *(composite state-time)*	...etc.
EID_12347	Time: 2018-02-07T08:22:40 *event timestamp)*	State: WARN *(event state)*	GSI PK: (random: 0-N) *(random GSI-PK value)*	GSI SK: WARN#2018-02-07T08:22:40 *(composite state-time)*	...etc.
EID_12348	Time: 2018-02-07T08:12:40 *event timestamp)*	State: INFO *(event state)*	GSI PK: (random: 0-N) *(random GSI-PK value)*	GSI SK: INFO#2018-02-07T08:12:40 *(composite state-time)*	...etc.



GSI

Primary Key		Data Attributes...
Partition Key	*Sort Key*	
GSI PK	GSI SK	...
[0-N]	INFO#2018-02-07T08:42:40 *(composite state-time)*	...etc.
[0-N]	CRITICAL#2018-02-07T08:32:40 *(composite state-time)*	...etc.
[0-N]	WARN#2018-02-07T08:22:40 *(composite state-time)*	...etc.
[0-N]	INFO#2018-02-07T08:12:40 *(composite state-time)*	...etc.

このスキーマ設計を使用すると、イベント項目は GSI 上の複数の 0-N パーティションに分散されるので、複合キーのソート条件を使用して分散読み込みを行い、指定された期間に特定の状態を持つすべての項目を取得できます。

このスキーマパターンは、テーブルスキャンを必要とせずに、最小限のコストで高度に選択的な結果セットを提供します。

グローバルセカンダリインデックスを使用した結果整合性レプリカの作成

グローバルセカンダリインデックスを使用して、テーブルの結果整合性レプリカを作成できます。レプリカを作成すると、次のことが可能になります。

- リーダーごとに異なるプロビジョニングされた読み込み容量を設定する。例えば、優先順位の高いクエリを処理し、最高レベルの読み込みパフォーマンスを必要とするアプリケーションと読み込みアクティビティのスロットリングを許容できる優先順位の低いクエリを処理するアプリケーションの 2 つのアプリケーションがあるとします。

これらのアプリケーションが同じテーブルから読み込むと、優先順位の低いアプリケーションからの読み込み負荷が高くなると、テーブルで使用可能な読み込み容量がすべて消費される可能性があります。これにより、優先度の高いアプリケーションの読み込みアクティビティが抑制されます。

代わりに、テーブル自体の読み込み容量とは別に設定できるグローバルセカンダリインデックスを通じてレプリカを作成できます。その後、テーブルの代わりに、優先度の低いアプリでレプリカにクエリすることができます。

- テーブルからの読み込みを完全に排除する。例えば、Web サイトから大量のクリックストリームアクティビティをキャプチャするアプリケーションがある場合、読み込みが妨げられる危険性はありません。このテーブルを分離し、グローバルセカンダリインデックスを使用して作成されたレプリカを他のアプリケーションに読み取らせながら、他のアプリケーションによる読み込みを防止することができます (「詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用」を参照)。

レプリカを作成するには、親テーブルと同じキースキーマを持つグローバルセカンダリインデックスを設定します。このインデックスには、キー以外の属性の一部またはすべてが入力されます。アプリケーションでは、一部またはすべての読み込みアクティビティを親テーブルではなく、このグローバルセカンダリインデックスに送信できます。その後、親テーブルのプロビジョニングされた読み込み容量を変更せずに、グローバルセカンダリインデックスのプロビジョニングされた読み込み容量を調整して、これらの読み込みを処理できます。

親テーブルへの書き込みから、書き込みデータがインデックスに表示されるまでの間には、常に短い伝播遅延があります。つまり、アプリケーションでは、グローバルセカンダリインデックスレプリカの結果整合性は親テーブルのものであるということを考慮に入れる必要があります。

複数のグローバルセカンダリインデックスレプリカを作成して、異なる読み込みパターンをサポートできます。レプリカを作成するときは、各読み込みパターンが実際に必要とする属性のみを計画します。これにより、アプリケーションは、親テーブルから項目を読み取る必要はなく、必要なデータだけを取得するために、プロビジョニングされた読み込み容量を少なくすることができます。この最適化により、長期的には大幅なコスト削減を実現できます。

大きな項目と属性を格納するベストプラクティス

Amazon DynamoDB は、テーブルに格納する各アイテムのサイズを 400 KB に制限します (「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照)。アプリケーションで DynamoDB のサイズ制限よりも多くのデータを格納する必要がある場合は、1 つ以上の大

きな属性を圧縮するか、項目を複数の項目に分割することができます (ソートキーによって効率的にインデックス付けされます)。Amazon Simple Storage Service (Amazon S3) にオブジェクトとして項目を保存して、Amazon S3 オブジェクト識別子を DynamoDB 項目に格納することもできます。

ベストプラクティスとして、400 KB の最大アイテムサイズに近づいたアイテムサイズを監視して警告するアイテムを書き込む際は、`ReturnConsumedCapacity` パラメータを使用します。アイテムの最大サイズを超えると、書き込みが失敗します。アプリケーションに影響が出る前にアイテムサイズを監視してアラートを出すことで、アイテムサイズに関する問題を軽減できます。

大量の属性値を圧縮する

大きな属性値を圧縮すると、DynamoDB の項目制限に収まり、ストレージコストが削減されます。GZIP や LZO などの圧縮アルゴリズムはバイナリ出力を生成し、これをアイテム内の `Binary` 属性タイプに格納することができます。

例えば、フォーラムユーザーの書いたメッセージを格納するテーブルを考えてみましょう。このようなメッセージは、圧縮に適した長い文字列を含むことがよくあります。圧縮するとアイテムサイズを小さくすることができますが、圧縮された属性値はフィルタリングで使用できないという問題があります。

DynamoDB で長いメッセージを圧縮する方法を示すサンプルコードについては、次のトピックを参照してください。

- 例: AWS SDK for Java ドキュメント API を使用したバイナリタイプ属性の処理
- 例: AWS SDK for .NET 低レベル API を使用したバイナリタイプ属性の処理

垂直パーティショニング

大きなアイテムを処理する際の代替解決策は、アイテムを小さなデータに分割し、関連するすべてのアイテムをパーティションキー値で関連付けることです。その後、ソートキー文字列を使用して、一緒に格納されている関連情報を特定します。さらに、複数のアイテムを同じパーティションキー値でグループ化することで、アイテムコレクションを作成できます。

このアプローチの詳細については、以下を参照してください。

- Use vertical partitioning to scale data efficiently in Amazon DynamoDB
- Implement vertical partitioning in Amazon DynamoDB using AWS Glue

大きな属性値を Amazon S3 に格納する

前述のように、Amazon S3 を使用して、DynamoDB 項目に収まらない大きな属性値を格納することもできます。Amazon S3 のオブジェクトとして保存してから、DynamoDB 項目にオブジェクト識別子を格納することができます。

また、Amazon S3 のオブジェクトメタデータサポートを使用して、DynamoDB の親項目にリンクを戻すこともできます。項目のプライマリキーバリューを Amazon S3 のオブジェクトの Amazon S3 メタデータとして格納します。この操作は、Amazon S3 オブジェクトのメンテナンスに役立ちます。

例えば、「DynamoDB でのコード例用のテーブルの作成とデータのロード」セクションの `ProductCatalog` テーブルを考えてみます。このテーブル内の項目は、価格、説明、書籍の著者、その他の製品の寸法に関する情報を格納します。大きすぎて項目に収まらない各製品のイメージを保存する場合は、DynamoDB ではなく、Amazon S3 にイメージを保存できます。

この方法を実装するときは、次の制限事項に注意してください。

- DynamoDB は、Amazon S3 と DynamoDB 間のトランザクションをサポートしていません。このため、アプリケーションでは失敗が発生する状況に対処する必要があります。例えば、孤立した Amazon S3 オブジェクトのクリーンアップなどがあります。
- Amazon S3 では、オブジェクト ID の長さが制限されます。そのため、過度に長いオブジェクト識別子を生成したり、他の Amazon S3 制約に違反しないようにデータを整理する必要があります。

Amazon S3 の使用方法の詳細については、Amazon Simple Storage Service ユーザーガイドを参照してください。

DynamoDB で時系列データを処理するベストプラクティス。

Amazon DynamoDB の一般的な設計原則では、使用するテーブルの数を最小限に抑えることをお勧めします。ほとんどのアプリケーションで、必要なテーブルは 1 つだけです。ただし、時系列データでは、期間あたりアプリケーションごとにテーブルを 1 個使用すると、多くの場合最適に処理できます。

時系列データの設計パターン

大容量のイベントを追跡する一般的な時系列シナリオを考えてみます。書き込みアクセスパターンは、記録されているすべてのイベントが今日の日付であることです。お客様の読み込みアクセスパ

ターンでは、今日のイベントは最も頻繁に、昨日のイベントはそれよりはるかに低い頻度で読み込まれ、それ以前の古いイベントはほとんど読み込まれません。これを処理する 1 つの方法として、現在の日時をプライマリキーに構築できます。

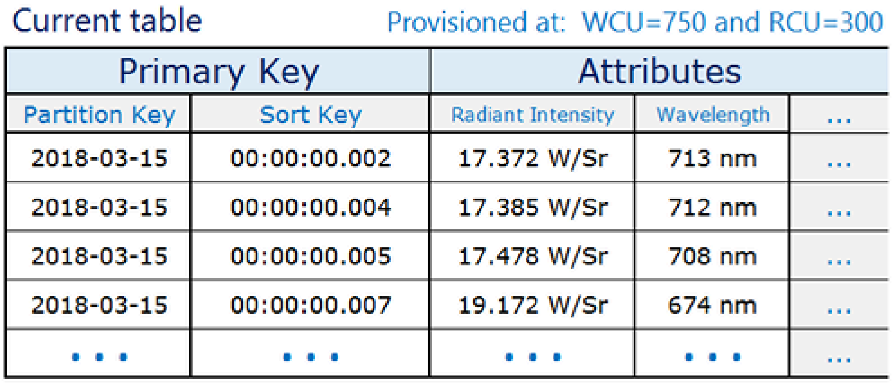
次の設計パターンでは、このようなシナリオを効率的に処理します。

- 必要な読み込みおよび書き込み容量と必要なインデックスでプロビジョニングされたテーブルを 1 つの期間に 1 つ作成します。
- 各期間が終了する前に、次の期間でテーブルを事前に作成します。現在の期間が終了するように、イベントトラフィックを新しいテーブルにリダイレクトします。これらのテーブルには、記録した期間を指定する名前を割り当てることができます。
- テーブルに書き込まれなくなったら、プロビジョニングされた書き込み容量を小さい値 (1 WCU など) に減らし、適切な読み込み容量をプロビジョニングします。時間の経過と共に、以前のテーブルのプロビジョニングされた読み込み容量を減らします。コンテンツがほとんど必要ないテーブルやコンテンツがまったく必要ないテーブルをアーカイブまたは削除することもできます。

トラフィックボリュームが最も多くなる現在の期間に必要なリソースを割り当て、アクティブに使用されなくなった以前のテーブルのプロビジョニングをスケールダウンすることでコストを削減できます。ビジネスニーズによっては、トラフィックを論理パーティションキーに均等に分散するために書き込みシャーディングを検討します。詳細については、「書き込みシャーディングを使用してワークロードを均等に分散させる」を参照してください。

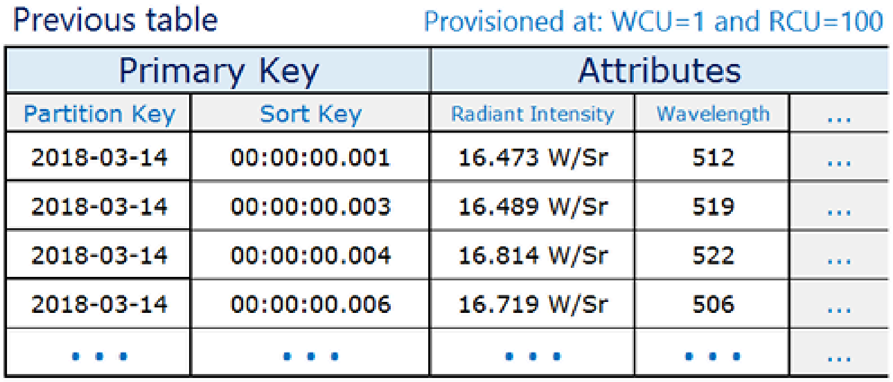
時系列テーブルの例

読み込み/書き込み容量が高いときに現在のテーブルがプロビジョニングされ、頻繁にアクセスされないために以前のテーブルが縮小される時系列データの例を以下に示します。



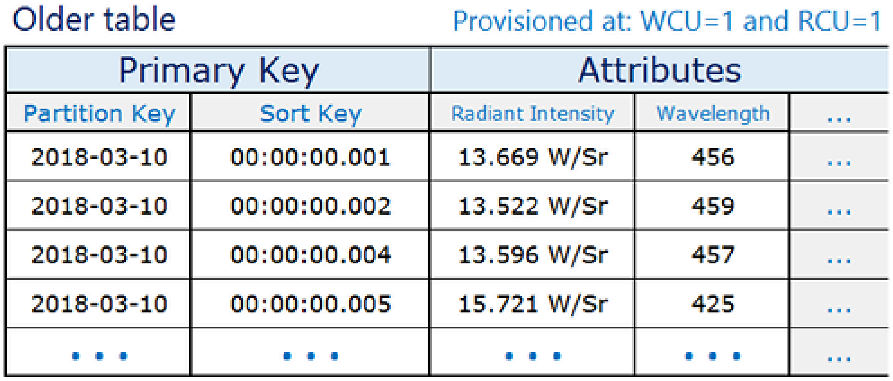
Current table Provisioned at: WCU=750 and RCU=300

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-15	00:00:00.002	17.372 W/Sr	713 nm	...
2018-03-15	00:00:00.004	17.385 W/Sr	712 nm	...
2018-03-15	00:00:00.005	17.478 W/Sr	708 nm	...
2018-03-15	00:00:00.007	19.172 W/Sr	674 nm	...
• • •	• • •	• • •	• • •	...



Previous table Provisioned at: WCU=1 and RCU=100

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-14	00:00:00.001	16.473 W/Sr	512	...
2018-03-14	00:00:00.003	16.489 W/Sr	519	...
2018-03-14	00:00:00.004	16.814 W/Sr	522	...
2018-03-14	00:00:00.006	16.719 W/Sr	506	...
• • •	• • •	• • •	• • •	...



Older table Provisioned at: WCU=1 and RCU=1

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-10	00:00:00.001	13.669 W/Sr	456	...
2018-03-10	00:00:00.002	13.522 W/Sr	459	...
2018-03-10	00:00:00.004	13.596 W/Sr	457	...
2018-03-10	00:00:00.005	15.721 W/Sr	425	...
• • •	• • •	• • •	• • •	...

多対多の関係を管理するためのベストプラクティス

隣接関係のリストは、Amazon DynamoDB の多対多の関係のモデリングに役立つ設計パターンです。より一般的には、DynamoDB のグラフデータ (ノードとエッジ) を表現する方法です。

隣接関係のリスト設計パターン

異なるエンティティのアプリケーションの間に、多対多の関係がある場合、その関係は隣接関係のリストとしてモデル化することができます。このパターンでは、最上位エンティティ (グラフモデル内

のノードと同義) はすべて、パーティションキーを使用して表現されます。他のエンティティとの関係 (グラフのエッジ) は、ソートキーの値をターゲットエンティティ ID (ターゲットノード) に設定することによって、パーティション内の項目として表現されます。

このパターンの利点として、ターゲットエンティティ (ターゲットノードへのエッジを含む) に関連するすべてのエンティティ (ノード) を見つけるために重複データを最小限に抑えられることや、クエリパターンを簡素化できる点などがあります。

このパターンが実際に役に立った例として、請求書に複数の請求書を含めることができる請求システムがあります。1 つの請求書を複数の請求書に含めることができます。この例のパーティションキーは、`InvoiceID` または `BillID` です。`BillID` パーティションには、請求書固有の属性がすべて含まれます。`InvoiceID` パーティションには、請求書固有の属性を格納する項目と、各 `BillID` の項目が含まれており、この項目は、1 つの請求書にまとめられます。

スキーマは次のようになります。

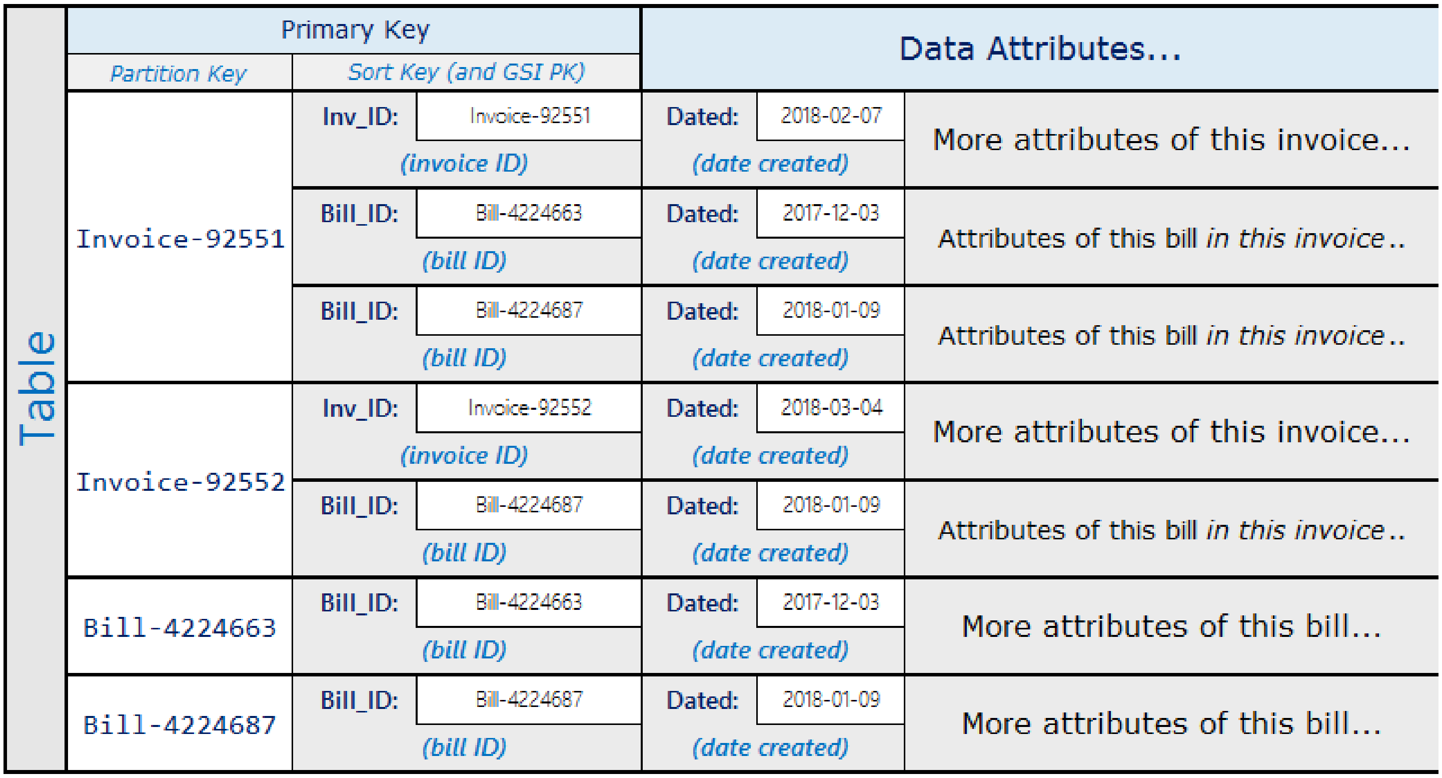
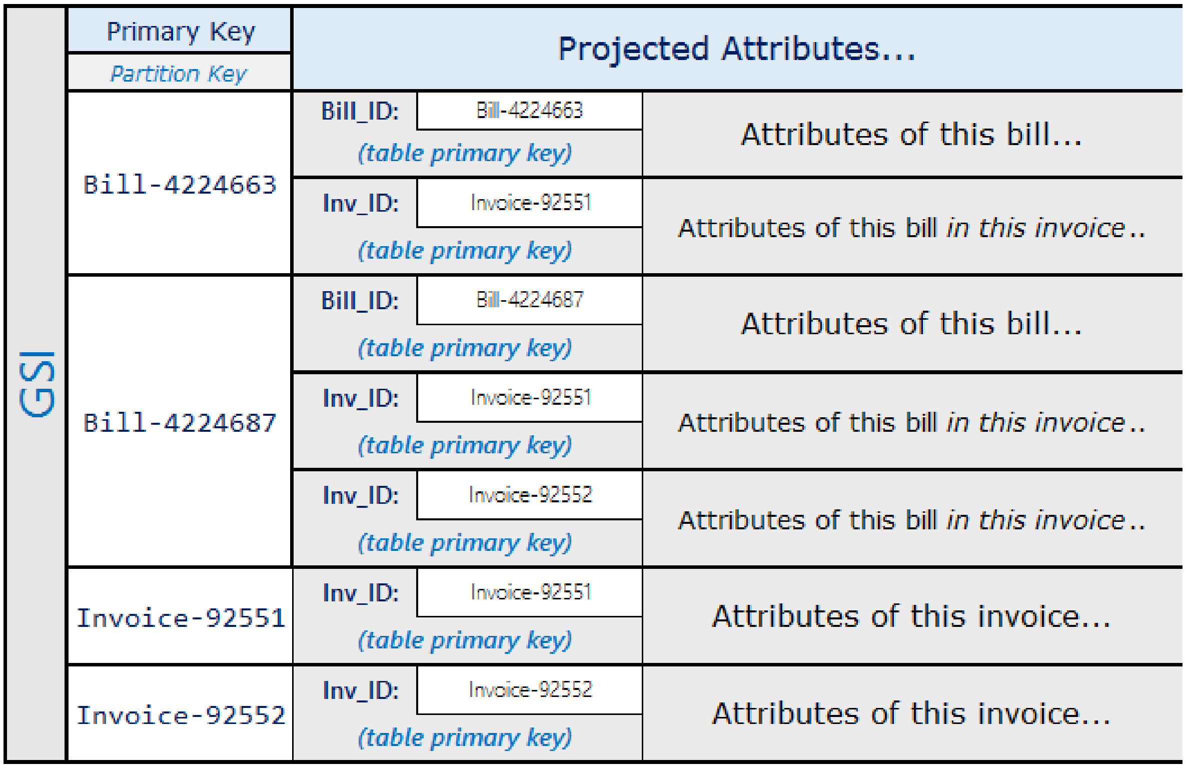


Table	Primary Key		Data Attributes...	
	Partition Key	*Sort Key (and GSI PK)*		
	`Invoice-92551`	Inv_ID: Invoice-92551 *(invoice ID)*	Dated: 2018-02-07 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224663 *(bill ID)*	Dated: 2017-12-03 *(date created)*	Attributes of this bill *in this invoice*..
		Bill_ID: Bill-4224687 *(bill ID)*	Dated: 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Invoice-92552`	Inv_ID: Invoice-92552 *(invoice ID)*	Dated: 2018-03-04 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224687 *(bill ID)*	Dated: 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Bill-4224663`	Bill_ID: Bill-4224663 *(bill ID)*	Dated: 2017-12-03 *(date created)*	More attributes of this bill...
	`Bill-4224687`	Bill_ID: Bill-4224687 *(bill ID)*	Dated: 2018-01-09 *(date created)*	More attributes of this bill...

上記のスキーマを使用すると、テーブルのプライマリキーを使用して、請求書の請求内容をすべて照会することができます。請求書の一部を含むすべての請求書を検索するには、テーブルのソートキーにグローバルセカンダリインデックスを作成します。

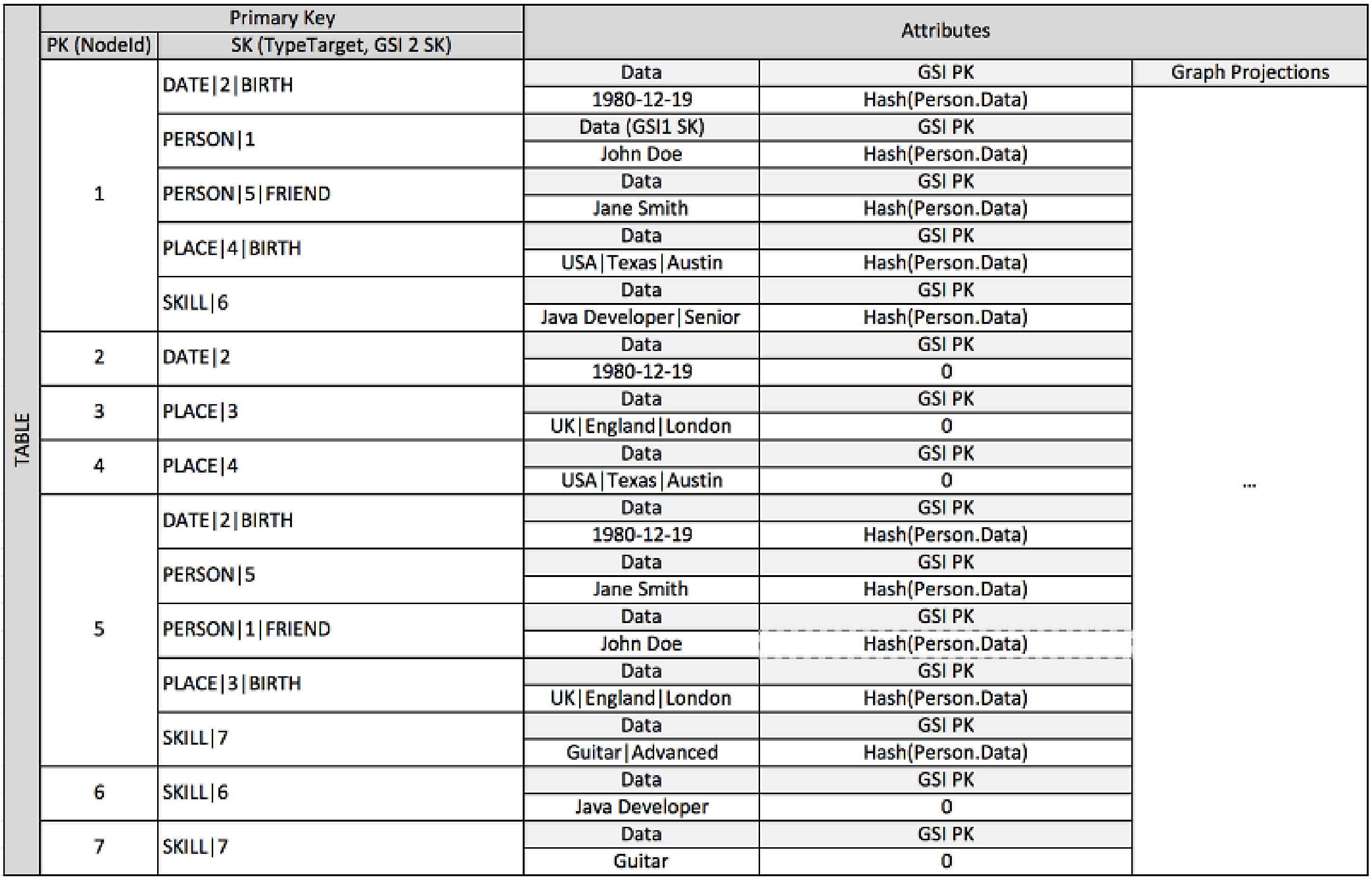
グローバルセカンダリインデックスの射影は、次のようになります。



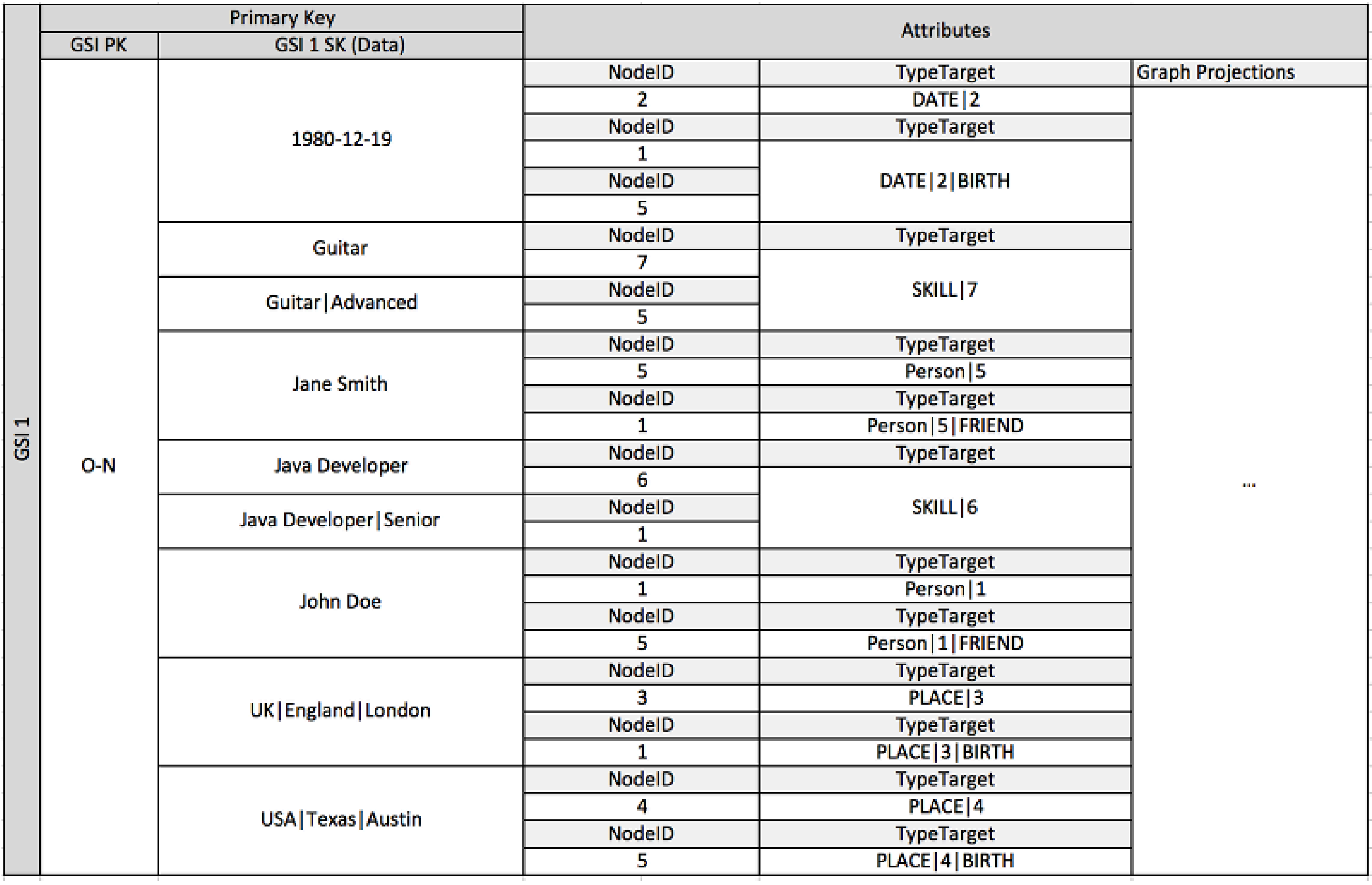
GSI	Primary Key	Projected Attributes...		
	Partition Key			
	Bill-4224663	Bill_ID: *(table primary key)*	Bill-4224663	Attributes of this bill...
		Inv_ID: *(table primary key)*	Invoice-92551	Attributes of this bill *in this invoice*..
	Bill-4224687	Bill_ID: *(table primary key)*	Bill-4224687	Attributes of this bill...
		Inv_ID: *(table primary key)*	Invoice-92551	Attributes of this bill *in this invoice*..
		Inv_ID: *(table primary key)*	Invoice-92552	Attributes of this bill *in this invoice*..
	Invoice-92551	Inv_ID: *(table primary key)*	Invoice-92551	Attributes of this invoice...
	Invoice-92552	Inv_ID: *(table primary key)*	Invoice-92552	Attributes of this invoice...

マテリアライズされたグラフパターン

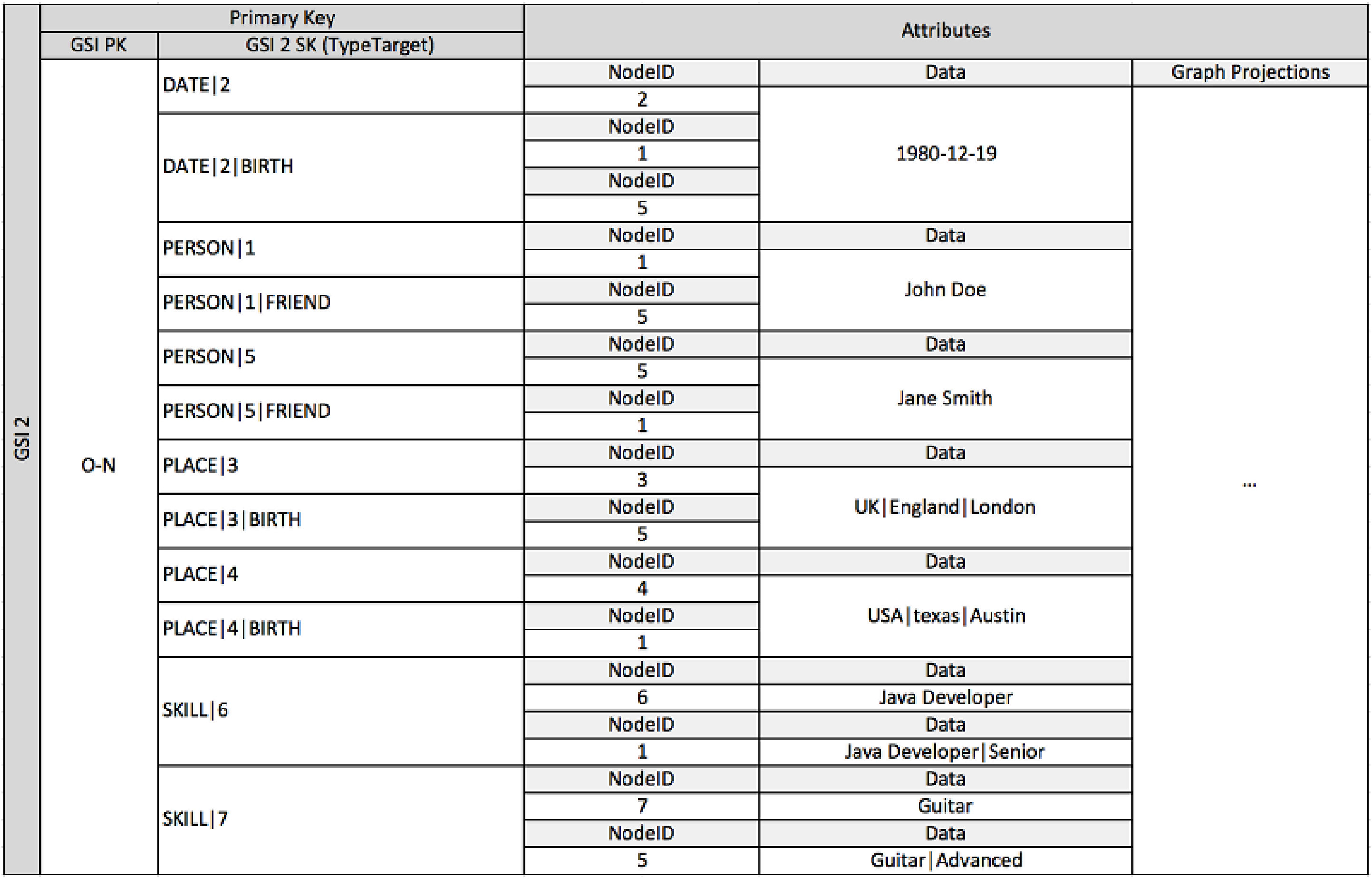
多くのアプリケーションは、ピア間のランキング、エンティティ間の共通関係、ネイバーエンティティの状態に加え、他のタイプのグラフスタイルのワークフローを理解することを中心として作成されています。これらのアプリケーションタイプの場合は、次のスキーマ設計パターンを検討します。



	Primary Key		Attributes		
	PK (NodeId)	SK (TypeTarget, GSI 2 SK)			
TABLE	1	DATE|2|BIRTH	Data	GSI PK	Graph Projections
			1980-12-19	Hash(Person.Data)	...
		PERSON|1	Data (GSI1 SK)	GSI PK	
			John Doe	Hash(Person.Data)	
		PERSON|5|FRIEND	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PLACE|4|BIRTH	Data	GSI PK	
			USA|Texas|Austin	Hash(Person.Data)	
		SKILL|6	Data	GSI PK	
			Java Developer|Senior	Hash(Person.Data)	
	2	DATE|2	Data	GSI PK	
			1980-12-19	0	
	3	PLACE|3	Data	GSI PK	
			UK|England|London	0	
	4	PLACE|4	Data	GSI PK	
			USA|Texas|Austin	0	
	5	DATE|2|BIRTH	Data	GSI PK	
			1980-12-19	Hash(Person.Data)	
		PERSON|5	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PERSON|1|FRIEND	Data	GSI PK	
			John Doe	Hash(Person.Data)	
		PLACE|3|BIRTH	Data	GSI PK	
			UK|England|London	Hash(Person.Data)	
		SKILL|7	Data	GSI PK	
			Guitar|Advanced	Hash(Person.Data)	
	6	SKILL|6	Data	GSI PK	
			Java Developer	0	
	7	SKILL|7	Data	GSI PK	
			Guitar	0	



	Primary Key		Attributes		
	GSI PK	GSI 1 SK (Data)			
GSI 1	O-N	1980-12-19	NodeID	TypeTarget	Graph Projections
			2	DATE|2	...
			NodeID	TypeTarget	
			1	DATE|2|BIRTH	
			NodeID		
			5		
		Guitar	NodeID	TypeTarget	
			7	SKILL|7	
		Guitar|Advanced	NodeID		
			5		
		Jane Smith	NodeID	TypeTarget	
			5	Person|5	
			NodeID	TypeTarget	
			1	Person|5|FRIEND	
		Java Developer	NodeID	TypeTarget	
			6	SKILL|6	
		Java Developer|Senior	NodeID		
			1		
		John Doe	NodeID	TypeTarget	
			1	Person|1	
			NodeID	TypeTarget	
			5	Person|1|FRIEND	
		UK|England|London	NodeID	TypeTarget	
			3	PLACE|3	
			NodeID	TypeTarget	
			1	PLACE|3|BIRTH	
		USA|Texas|Austin	NodeID	TypeTarget	
			4	PLACE|4	
			NodeID	TypeTarget	
			5	PLACE|4|BIRTH	



GSI 2	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)			
	O-N	DATE|2	NodeID	Data	Graph Projections
			2	1980-12-19	...
		DATE|2|BIRTH	NodeID		
			1		
			NodeID		
			5		
		PERSON|1	NodeID	Data	
			1	John Doe	
		PERSON|1|FRIEND	NodeID		
			5		
		PERSON|5	NodeID	Data	
			5	Jane Smith	
		PERSON|5|FRIEND	NodeID		
			1		
		PLACE|3	NodeID	Data	
			3	UK|England|London	
		PLACE|3|BIRTH	NodeID		
			5		
		PLACE|4	NodeID	Data	
			4	USA|texas|Austin	
		PLACE|4|BIRTH	NodeID		
			1		
		SKILL|6	NodeID	Data	
			6	Java Developer	
			NodeID	Data	
			1	Java Developer|Senior	
		SKILL|7	NodeID	Data	
			7	Guitar	
			NodeID	Data	
			5	Guitar|Advanced	

前述のスキーマは、グラフのエッジおよびノードを定義する項目を含むデータパーティションのセットによって定義されるグラフデータ構造を示します。エッジ項目には、Target 属性および Type 属性が含まれます。これらの属性は、プライマリテーブルのパーティションまたはグローバルセカンダリインデックスの項目を識別する複合キー名 "TypeTarget" の一部として使用されます。

最初のグローバルセカンダリインデックスは、Data 属性で構築されます。この属性は、以前説明したようなグローバルセカンダリインデックスの多重定義を使用して、複数の異なる属性タイプ (Dates、Names、Places、Skills のインデックス) を作成します。ここで、1 つのグローバルセカンダリインデックスは、4 つの異なる属性のインデックスを効果的に作成します。

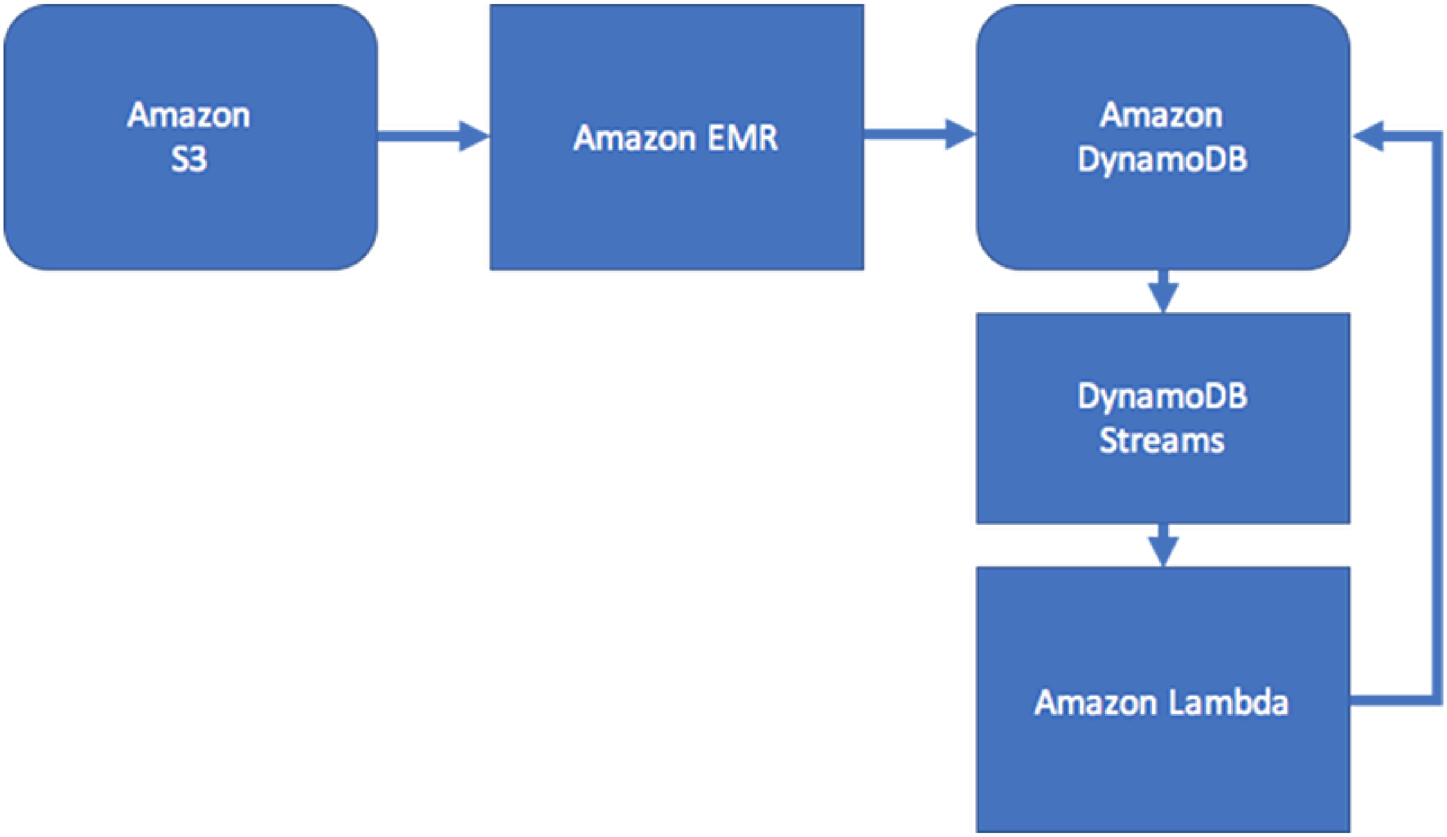
項目をテーブルに挿入する際に、インテリジェントなシャーディング戦略を使用して、大量の集計 (生年月日、スキル) を含む項目セットを、ホット読み書き問題を回避するために必要なグローバルなセカンダリインデックスの論理パーティションに分散させることができます。

このような設計パターンの組み合わせによって、効率の高いリアルタイムグラフワークフローのソリッドなデータストアが実現しました。これらのワークフローでは、推奨のエンジン、ソーシャルネットワーキングアプリケーション、ノードのランキング、サブツリーの集約、およびその他の一般

的なグラフのユースケースに対して、高パフォーマンスなネイバーエンティティの状態とエッジの集約クエリを使用できます。

ユースケースがリアルタイムデータの一貫性に優れていない場合は、スケジュールされた Amazon EMR プロセスを使用して、エッジをワークフローに関連するグラフサマリー集約に追加することができます。グラフにエッジが追加されたときにアプリケーションですぐに認識する必要がない場合は、スケジュールされたプロセスを使用して結果を集約できます。

一定レベルの一貫性を維持するために、この設計には、エッジ更新を処理する Amazon DynamoDB Streams や AWS Lambda が含まれることがあります。また、Amazon EMR ジョブを使用して、定期的に結果を検証することもできます。このアプローチを次の図に示します。このアプローチは、リアルタイムクエリのコストが高く、各ユーザーの更新をすぐに知る必要性が低いソーシャルネットワーキングアプリケーションで一般的に使用されます。



IT サービス管理 (ITSM) およびセキュリティアプリケーションでは通常、複雑なエッジ集約で構成されるエンティティ状態の変化にリアルタイムで応答する必要があります。このようなアプリケーションでは、第 2 レベルと第 3 レベルの関係や複雑なエッジトラバーサルの複数のリアルタイムノード集約をサポートできるシステムが必要です。ユースケースに、これらのタイプのリアルタイムグラフクエリワークフローが必要な場合は、これらのワークフローの管理に Amazon Neptune の使用を検討することをお勧めします。

> Note
>
> 高度に接続されたデータセットをクエリする必要がある場合、またはミリ秒のレイテンシーで複数のノードを通過する必要があるクエリ (マルチホップクエリとも呼ばれます) を実行する必要がある場合は、Amazon Neptune の使用を検討してください。Amazon Neptune は専用の高性能グラフデータベースエンジンです。このエンジンは、数十億の関係を保存し、ミリ秒単位のレイテンシーでグラフをクエリできるよう最適化されています。

ハイブリッドなデータベースシステムを実装するためのベストプラクティス

状況によっては、1 つ以上のリレーショナルデータベース管理システム (RDBMS) から Amazon DynamoDB へ移行することは有効ではない場合があります。このような場合は、ハイブリッドシステムを作成することをお勧めします。

すべてを DynamoDB に移行したくない場合

例えば、組織によっては、会計や運用に必要な多数のレポートを生成するコードに多額の資金を投資している場合があります。レポートの生成に要する時間は重要ではありません。リレーショナルシステムの柔軟性はこのような類のタスクに適しており、NoSQL コンテキストでこれらのレポートをすべて作成し直すことは非常に困難かもしれません。

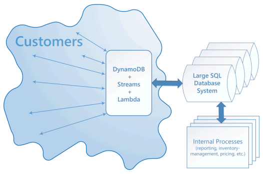
また、一部の組織では、何十年にも渡って取得または継承してきた従来のさまざまなリレーショナルシステムが使用され続けています。これらのシステムからデータを移行することは、あまりにも危険なだけでなく、高価なため、その労力を正当に評価できない場合があります。

ただし、このような組織の業務は、高トラフィックのお客様向けウェブサイトに依存していること分かります。ここでは、ミリ秒単位の対応が不可欠です。リレーショナルシステムでは、巨大な (多くの場合は承認できない) 費用がある場合を除き、この要件を満たすようにスケーリングすることはできません。

このような状況では、DynamoDB で 1 つ以上のリレーショナルシステムに格納されたデータのマテリアライズドビューを作成し、このビューに対して高トラフィックのリクエストを処理するハイブリッドシステムを作成することができます。このタイプのシステムでは、お客様向けのトラフィックを処理するために以前必要だったサーバーハードウェアやメンテナンスおよび RDBMS ライセンスを排除することで、潜在的なコストを削減できます。

ハイブリッドシステムの実装方法

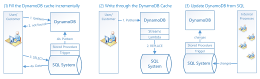
DynamoDB では、DynamoDB Streams と AWS Lambda を利用して、1 つ以上の既存のリレーショナルデータベースシステムとシームレスに統合できます。


Customers
DynamoDB
+
Streams
+
Lambda
Large SQL Database System
Internal Processes (reporting, inventory-management, pricing, etc.)

DynamoDB Streams と AWS Lambda を統合するシステムには、いくつかの利点があります。

- マテリアライズドビューの永続キャッシュとして機能する。
- データがクエリされるときや、SQL システムでデータが変更されるときに、そのデータを徐々に満たすように設定することができる。つまり、ビュー全体を事前に入力する必要はありません。したがって、プロビジョンドスループット性能が効率的に使用される可能性が高くなります。
- 管理コストが低く、高い可用性と信頼性を備えています。

このような統合を実装するには、本質的に 3 種類の相互運用を実現する必要があります。



1. DynamoDB キャッシュを段階的に埋めます。項目をクエリする場合は、最初に DynamoDB で検索します。存在しない場合は、SQL システムで検索し、DynamoDB にロードします。
2. DynamoDB キャッシュを通じて書き込みます。顧客が DynamoDB で値を変更すると、Lambda 関数がトリガーされ、新しいデータが SQL システムに書き戻されます。
3. SQL システムから DynamoDB を更新します。インベントリ管理や価格設定などの内部プロセスで SQL システムの値が変更されると、ストアドプロシージャがトリガーされ、DynamoDB マテリアライズドビューに変更が反映されます。

これらのオペレーションは簡単です。また、すべてのシナリオでこれらのオペレーションがすべて必要とは限りません。

ハイブリッドソリューションは、主に DynamoDB を使用する場合にも便利ですが、クエリを一度のみ実行する場合、または特別なセキュリティが必要なオペレーションや、時間が重要でないオペレーションに対しては、小さなリレーショナルシステムを維持する場合もあります。

DynamoDB でリレーショナルデータをモデル化するためのベストプラクティス

このセクションでは、Amazon DynamoDB でリレーショナルデータをモデル化するためのベストプラクティスを説明します。まず、従来のデータモデリングのコンセプトを紹介します。次に、従来のリレーショナルデータベース管理システムではなく DynamoDB を使用する利点について説明します。つまり JOIN オペレーションが不要になり、オーバーヘッドが削減されるということです。

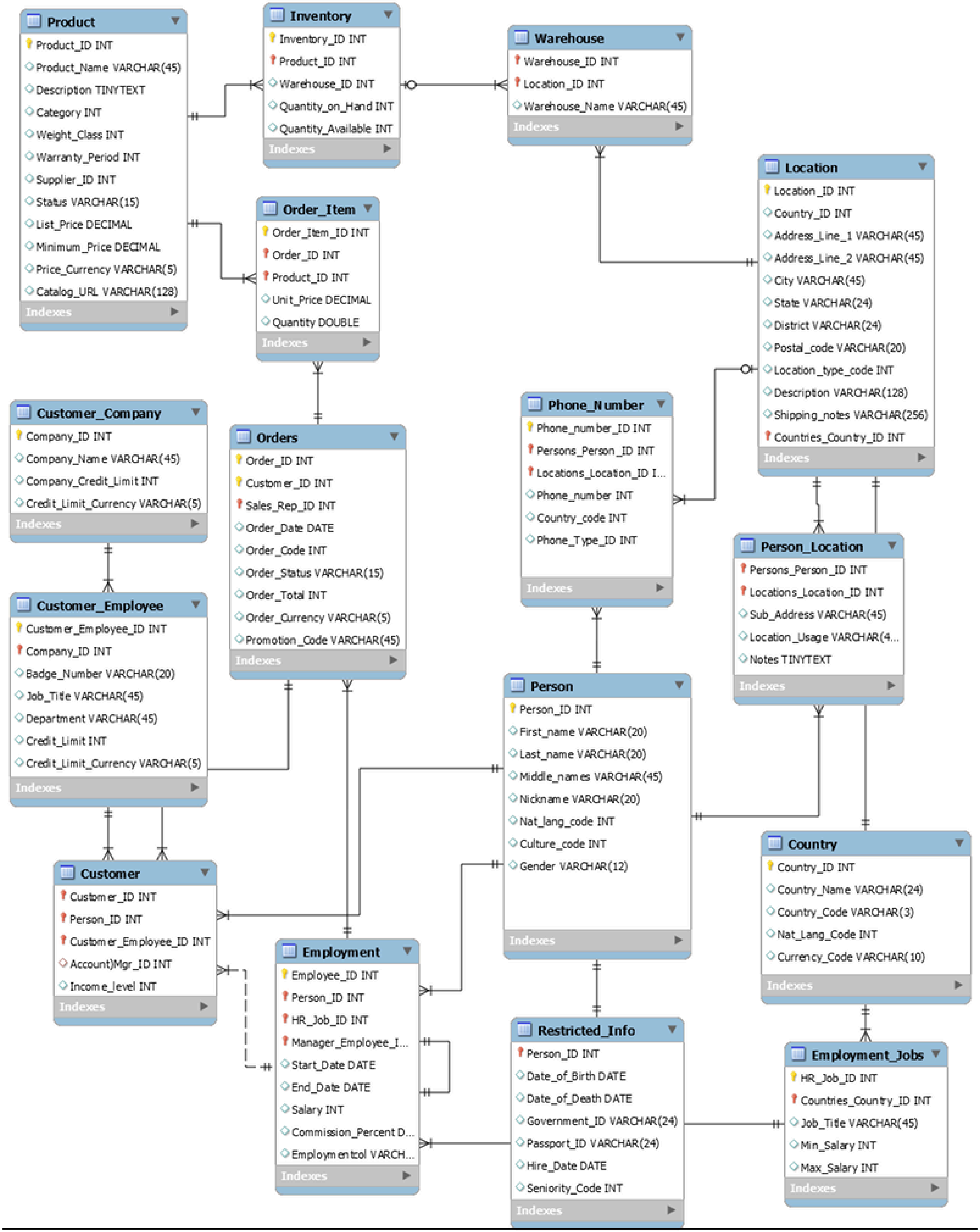
次に、効率的にスケーリングする DynamoDB テーブルを設計する方法を説明します。最後に、DynamoDB でリレーショナルデータをモデル化する方法の例を示します。

トピック

従来のリレーショナルデータベースモデル

従来のリレーショナルデータベース管理システム (RDBMS) は、正規化されたリレーショナル構造でデータを保存します。リレーショナルデータモデルの目的は、(正規化によって) データの重複を減らして参照整合性を維持し、データの異常を削減することです。

次のスキーマは、一般的な注文入力アプリケーションのリレーショナルデータ型の一例です。このアプリケーションは、理論上の製造元の運用およびビジネスサポートシステムを支える人事スキーマをサポートします。


Product
Product_ID INT
Product_Name VARCHAR(45)
Description TINYTEXT
Category INT
Weight_Class INT
Warranty_Period INT
Supplier_ID INT
Status VARCHAR(15)
List_Price DECIMAL
Minimum_Price DECIMAL
Price_Currency VARCHAR(5)
Catalog_URL VARCHAR(128)
Indexes
Inventory
Inventory_ID INT
Product_ID INT
Warehouse_ID INT
Quantity_on_Hand INT
Quantity_Available INT
Indexes
Warehouse
Warehouse_ID INT
Location_ID INT
Warehouse_Name VARCHAR(45)
Indexes
Order_Item
Order_Item_ID INT
Order_ID INT
Product_ID INT
Unit_Price DECIMAL
Quantity DOUBLE
Indexes
Location
Location_ID INT
Country_ID INT
Address_Line_1 VARCHAR(45)
Address_Line_2 VARCHAR(45)
City VARCHAR(45)
State VARCHAR(24)
District VARCHAR(24)
Postal_code VARCHAR(20)
Location_type_code INT
Description VARCHAR(128)
Shipping_notes VARCHAR(256)
Countries_Country_ID INT
Indexes
Customer_Company
Company_ID INT
Company_Name VARCHAR(45)
Company_Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Orders
Order_ID INT
Customer_ID INT
Sales_Rep_ID INT
Order_Date DATE
Order_Code INT
Order_Status VARCHAR(15)
Order_Total INT
Order_Currency VARCHAR(5)
Promotion_Code VARCHAR(45)
Indexes
Phone_Number
Phone_number_ID INT
Persons_Person_ID INT
Locations_Location_ID I...
Phone_number INT
Country_code INT
Phone_Type_ID INT
Indexes
Person_Location
Persons_Person_ID INT
Locations_Location_ID INT
Sub_Address VARCHAR(45)
Location_Usage VARCHAR(4...
Notes TINYTEXT
Indexes
Customer_Employee
Customer_Employee_ID INT
Company_ID INT
Badge_Number VARCHAR(20)
Job_Title VARCHAR(45)
Department VARCHAR(45)
Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Person
Person_ID INT
First_name VARCHAR(20)
Last_name VARCHAR(20)
Middle_names VARCHAR(45)
Nickname VARCHAR(20)
Nat_lang_code INT
Culture_code INT
Gender VARCHAR(12)
Indexes
Country
Country_ID INT
Country_Name VARCHAR(24)
Country_Code VARCHAR(3)
Nat_Lang_Code INT
Currency_Code VARCHAR(10)
Indexes
Customer
Customer_ID INT
Person_ID INT
Customer_Employee_ID INT
Account)Mgr_ID INT
Income_level INT
Indexes
Employment
Employee_ID INT
Person_ID INT
HR_Job_ID INT
Manager_Employee_I...
Start_Date DATE
End_Date DATE
Salary INT
Commission_Percent D...
Employmentcol VARCH...
Indexes
Restricted_Info
Person_ID INT
Date_of_Birth DATE
Date_of_Death DATE
Government_ID VARCHAR(24)
Passport_ID VARCHAR(24)
Hire_Date DATE
Seniority_Code INT
Indexes
Employment_Jobs
HR_Job_ID INT
Countries_Country_ID INT
Job_Title VARCHAR(45)
Min_Salary INT
Max_Salary INT
Indexes

非リレーショナルデータベースサービスである DynamoDB には、従来のリレーショナルデータベース管理システムと比較して多くの利点があります。

DynamoDB によって JOIN オペレーションが不要になる理由

RDBMS は、構造クエリ言語 (SQL) を使用してデータをアプリケーションに返します。データ型の正規化により、このようなクエリでは通常、JOIN 演算子を使用して 1 つ以上のテーブルのデータを結合する必要があります。

例えば、各項目を出荷するすべてのウェアハウスで在庫数量でソートされた発注書項目のリストを生成するには、前のスキーマに対して次の SQL クエリを発行します。

```
SELECT * FROM Orders
  INNER JOIN Order_Items ON Orders.Order_ID = Order_Items.Order_ID
  INNER JOIN Products ON Products.Product_ID = Order_Items.Product_ID
  INNER JOIN Inventories ON Products.Product_ID = Inventories.Product_ID
  ORDER BY Quantity_on_Hand DESC
```

この種の SQL クエリは、データにアクセスするための柔軟な API を提供できますが、大量の処理が必要です。クエリを結合するたびに、クエリのランタイムの複雑さが増します。これは、各テーブルのデータをステージングしてからアセンブルして結果セットを返す必要があるためです。

クエリの実行時間に影響するその他の要因は、テーブルのサイズおよび結合される列にインデックスがあるかどうかです。前述のクエリは、複数のテーブルにわたって複雑なクエリを実行し、結果として生成されるセットをソートします。

JOINs の必要性をなくすことが NoSQL データモデリングの中核となります。これが、Amazon.com をサポートするために DynamoDB を構築した理由であり、DynamoDB があらゆる規模で一貫したパフォーマンスを提供できる理由です。SQL クエリと JOINs のランタイムの複雑さを考慮すると、RBDMS のパフォーマンスは規模に応じて一定ではありません。そのため、お客様のアプリケーションが大きくなるにつれてパフォーマンスの問題が発生します。

データを正規化することでディスクに保存されるデータの量は削減されますが、多くの場合、パフォーマンスに影響する最も制約の厳しいリソースは CPU 時間とネットワーク遅延です。

DynamoDB は、JOINs を削除し(かつデータの非正規化を促進し)、データベースアーキテクチャを最適化して、項目に 1 回のリクエストでアプリケーションクエリに完全に応答します。これによって、両方の制約を最小限に抑えます。これらの特性により、DynamoDB は規模にかかわらず 1 桁のミリ秒単位のパフォーマンスを実現できます。これは、一般的なアクセスパターンでは、DynamoDB オペレーションの実行時の複雑さが一定であるためです。

DynamoDB トランザクションが書き込みプロセスのオーバーヘッドを排除する方法

RDBMS の速度を低下させるもう 1 つの要因は、正規化されたスキーマに書き込むためにトランザクションを使用することです。例に示す通り、ほとんどのオンライントランザクション処理 (OLTP) アプリケーションで使用されるリレーショナルデータ構造は、RDBMS に格納されているときに複数の論理テーブルに分割して分散させる必要があります。

そのため、アプリケーションで書き込み中のオブジェクトを読み込もうとした場合に発生する可能性のある競合状態およびデータの整合性の問題を回避するためには、ACID 準拠のトランザクションフレームワークが必要です。こうしたトランザクションフレームワークをリレーショナルスキーマと組み合わせると、書き込みプロセスにかなりのオーバーヘッドが発生する可能性があります。

DynamoDB にトランザクションを実装すると、RDBMS で一般的に見られるスケーリングの問題が防止されます。DynamoDB では、トランザクションを 1 回の API 呼び出しとして発行し、その 1 回のトランザクションでアクセスできる項目の数を制限することで、これを行っています。トランザクションの実行時間が長いと、トランザクションがクローズされないため、データのロックが長期間、または永続的に保持され、運用上の問題が発生する可能性があります。

DynamoDB でのこのような問題を防ぐため、トランザクションは `TransactWriteItems` と `TransactGetItems` の 2 つの異なる API オペレーションを使用して実装されました。これらの API オペレーションには、RDBMS で一般的な開始および終了セマンティクスがありません。さらに、DynamoDB では、同様に長時間実行されるトランザクションを防ぐため、1 回のトランザクション内で 100 項目のアクセス制限を設けています。DynamoDB トランザクションの詳細については、「トランザクションでの使用」を参照してください。

こうした理由から、高トラフィックのクエリに対して低レイテンシーの応答が必要な場合は、NoSQL システムを利用すると、一般的に技術的および経済的な効果がもたらされます。Amazon DynamoDB では、リレーショナルシステムのスケーラビリティを制限する問題を回避できます。

RDBMS のパフォーマンスは、通常、以下の理由から適切にスケーリングできません。

- 高価な結合を使用して、クエリ結果の必要なビューを再構成します。
- データを正規化し、複数のクエリを必要とする複数のテーブルに格納して、ディスクに書き込みます。
- 一般的に、ACID 準拠のトランザクションシステムのパフォーマンスコストが発生します。

DynamoDB は、次の理由により適切に拡張されます。

- スキーマの柔軟性により、DynamoDB は複雑な階層データを 1 つの項目内に格納できます。
- 複合キー設計では、関連する項目を同じテーブルの範囲内に格納できます。
- トランザクションは 1 回のオペレーションで実行されます。オペレーションの実行が長時間に及ぶのを避けるため、アクセスできる項目の上限数は 100 となっています。

データストアに対するクエリは、多くの場合次のような形式で、非常に簡単になります。

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB は、以前の例の RDBMS と比較して、リクエストされたデータを返す作業がはるかに少なくなっています。

DynamoDB でリレーショナルデータをモデル化するための最初のステップ

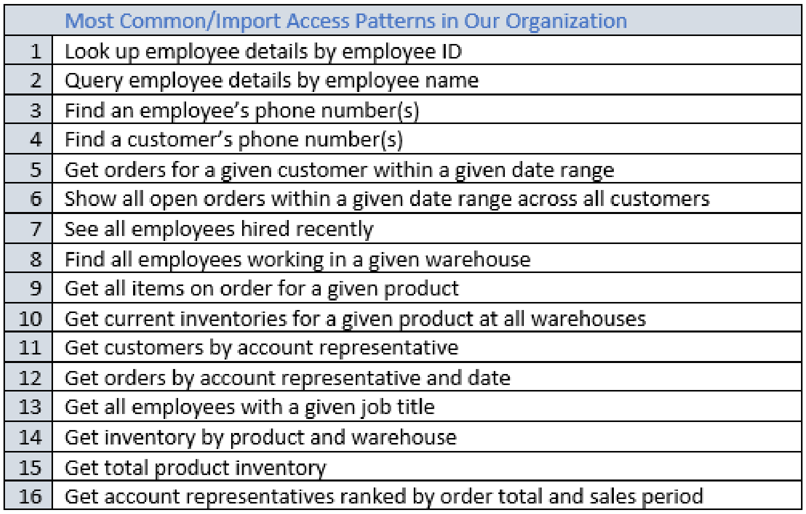
⚠ Important

NoSQL 設計では、RDBMS 設計とは異なる考え方が必要です。RDBMS の場合は、アクセスパターンを考慮せずに正規化されたデータモデルを作成できます。その後、新しい課題とクエリの要件が発生したら、そのデータモデルを拡張することができます。対照的に、Amazon DynamoDB の場合は答えが必要な質問が分かるまで、スキーマの設計を開始しないでください。ビジネス上の問題とアプリケーションのユースケースを理解することが極めて重要です。

効率的に拡張する DynamoDB テーブルの設計を開始するには、サポートする必要のあるオペレーションおよびビジネスサポートシステム (OSS/BSS) で必要とされるアクセスパターンを特定するために、まずいくつかの手順を実行する必要があります。

- 新しいアプリケーションの場合は、アクティビティや目的に関するユーザーストーリーを確認します。特定するさまざまなユースケースを文書化し、必要なアクセスパターンを分析します。
- 既存のアプリケーションでは、クエリログを分析して、ユーザーが現在どのようにシステムを使用しているか、主要なアクセスパターンを調べます。

このプロセスが完了したら、次のようなリストが表示されます。



	Most Common/Import Access Patterns in Our Organization
1	Look up employee details by employee ID
2	Query employee details by employee name
3	Find an employee’s phone number(s)
4	Find a customer’s phone number(s)
5	Get orders for a given customer within a given date range
6	Show all open orders within a given date range across all customers
7	See all employees hired recently
8	Find all employees working in a given warehouse
9	Get all items on order for a given product
10	Get current inventories for a given product at all warehouses
11	Get customers by account representative
12	Get orders by account representative and date
13	Get all employees with a given job title
14	Get inventory by product and warehouse
15	Get total product inventory
16	Get account representatives ranked by order total and sales period

実際のアプリケーションでは、リストはさらに長くなる場合があります。しかし、このコレクションは、本番環境で見つかる可能性のあるクエリパターンの複雑さの範囲を表します。

DynamoDB スキーマ設計の一般的なアプローチとして、アプリケーションレイヤーエンティティを識別し、非正規化と複合キー集約を使用してクエリの複雑さを軽減します。

DynamoDB では、複合ソートキー、多重定義のグローバルセカンダリインデックス、パーティションテーブル/インデックス、その他のデザインパターンを使用することを意味します。これらの要素を使用してデータを構造化することで、アプリケーションは、テーブルまたはインデックスの単一のクエリを使用して、特定のアクセスパターンに必要なものを取得できます。リレーショナルモデル化に示されている正規化されたスキーマをモデル化するために使用できる主なパターンは、隣接関係リストのパターンです。この設計で使用されるその他のパターンには、グローバルセカンダリインデックスの書き込みシャーディング、グローバルセカンダリインデックスの多重定義、複合キー、マテリアライズされた集計があります。

Important

一般的に、DynamoDB アプリケーションではできるだけ少ないテーブルを維持する必要があります。例外として、大容量の時系列データが必要な場合や、データセットのアクセスパターンが非常に異なる場合などがあります。反転されたインデックスを含む単一のテーブルでは通常、シンプルなクエリを使用してアプリケーションで必要とされる複雑な階層データ構造を構築および取得できます。

NoSQL Workbench for DynamoDB を使用してパーティションキー設計を視覚化する方法については、「NoSQL Workbench を使用したデータモデルの構築」を参照してください。

DynamoDB でリレーショナルデータをモデル化する例

この例では、Amazon DynamoDB でリレーショナルデータをモデル化する方法について説明します。DynamoDB テーブルの設計は、「リレーショナルモデル化」に示されているリレーショナルオーダーのエントリスキーマに対応しています。これは、DynamoDB のリレーショナルデータ構造を表す一般的な方法である「隣接関係のリスト設計パターン」に基づいています。

この設計パターンでは、通常、リレーショナルスキーマのさまざまなテーブルに相関するエンティティタイプのセットを定義する必要があります。その後、エンティティ項目は、複合 (パーティションおよびソート) のプライマリキーを使用してテーブルに追加されます。これらのエンティティ項目のパーティションキーは、項目を一意に識別する属性です。また、このパーティションキーは一般的に、すべての項目で PK と呼ばれます。ソートキー属性には、反転されたインデックスまたはグローバルセカンダリインデックスに使用できる属性値が含まれています。これは、一般的に SK と呼ばれます。

以下のエンティティを定義します。このエンティティは、リレーショナルオーダーのエントリスキーマをサポートしています。

1. HR-Employee - PK: EmployeeID, SK: Employee Name
2. HR-Region - PK: RegionID, SK: Region Name
3. HR-Country - PK: CountryId, SK: Country Name
4. HR-Location - PK: LocationID, SK: Country Name
5. HR-Job - PK: JobID, SK: Job Title
6. HR-Department - PK: DepartmentID, SK: DepartmentName
7. OE-Customer - PK: CustomerID, SK: AccountRepID
8. OE-Order - PK OrderID, SK: CustomerID
9. OE-Product - PK: ProductID, SK: Product Name
10. OE-Warehouse - PK: WarehouseID, SK: Region Name

これらのエンティティ項目をテーブルに追加したら、エンティティ項目のパーティションにエッジ項目を追加することで、それらのエンティティ項目間の関係を定義できます。このステップを以下のテーブルに示します。

この例では、`Employee`、`Order`、`Product Entity` のパーティションには、テーブル上の他のエンティティ項目へのポインタを含む追加のエッジ項目があります。次に、以前に定義されたすべての

アクセスパターンをサポートするために、グローバルセカンダリインデックス (GSI) をいくつか定義します。すべてのエンティティ項目で、プライマリキーまたはソートキー属性に同じタイプの値を使用するわけではありません。必要なことは、プライマリキー属性とソートキー属性をテーブルに挿入することだけです。

これらのエンティティの一部に適切な名前が使用されており、他のエンティティではソートキーバリューとして他のエンティティ ID が使用されているため、同じグローバルセカンダリインデックスで複数のタイプのクエリをサポートすることができます。この技術は、GSI 多重定義と呼ばれます。これにより、複数の項目タイプを含むテーブルのデフォルトの 20 のグローバルセカンダリインデックスの制限は効果的に排除されます。これは、GSI 1 として次の図に示されています。

GSI 2 は、一般的なアプリケーションアクセスパターンをサポートするように設計されています。これは、特定の状態のテーブル上のすべての項目を取得することを目的としています。使用可能な状態間で項目の分散が不均一な大きなテーブルの場合は、この項目が、並列でクエリ可能な複数の論理パーティションに分散されていない限り、このアクセスパターンはホットキーになります。この設計パターンは、`write sharding` と呼ばれます。

GSI 2 で実現するには、アプリケーションで、すべての Order 項目に GSI 2 プライマリキー属性を追加します。これは、0〜N の範囲内の乱数にそれを移入します。N は、特別な理由がない限り、一般的に次の式を使用して計算できます。

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

例えば、次のことを前提とします。

- 最大 200 万件の注文がシステム内にあり、5 年間で 300 万件に増加します。
- これらの注文のうち、最大 20% は任意の時点で OPEN 状態になります。
- 平均注文レコードは約 100 バイトで、注文パーティション内の 3 つの OrderItem レコードはそれぞれ約 50 バイト、注文エンティティの平均サイズは 250 バイトです。

そのテーブルでは、N 係数の計算方法は次のようになります。

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K
```

```
N = (0.2 * 3M) / 48K = 13
```

この場合、Order 状態のすべての OPEN 項目を読み込んでも、物理ストレージレイヤーにホットパーティションが発生しないように、GSI 2 の少なくとも 13 の論理パーティションにわたってすべての注文を分散させる必要があります。この番号は、データセットの異常を考慮した上で、埋め込むことをお勧めします。そのため、N = 15 を使用したモデルは問題ありません。前述したように、これを行うには、テーブルに挿入された Order レコードと OrderItem のレコードの GSI 2 PK 属性にランダムな 0～N 値を追加します。

この概要では、すべての OPEN の請求書を収集する必要があるアクセスパターンは比較的発生頻度が低いことを前提としているため、バースト容量を使用してリクエストを満たすことができます。State と Date Range のソートキー条件を使用して、次のグローバルセカンダリインデックスを照会し、必要に応じて特定の状態のサブセットまたはすべての Orders を生成することができます。

この例では、項目は 15 の論理パーティションにわたってランダムに分散されています。この構造は、アクセスパターンで多数の項目を取得する必要があるために機能します。したがって、15 スレッドのいずれかが空の結果セットを返して、無駄な容量を表すことはほとんどありません。何も返されなかった場合やデータが書き込まれていない場合でも、クエリは常に 1 つの読み込み容量ユニット (RCU) または 1 つの書き込み容量ユニット (WCU) を使用します。

アクセスパターンが、スパースの結果セットを返すこのグローバルセカンダリインデックスで、高速なクエリを必要とする場合は、項目を分散させるため、ランダムパターンではなく、ハッシュアルゴリズムを使用することをお勧めします。この場合、実行時にクエリが実行されたときに認識される属性を選択し、項目が挿入されたときにその属性を 0～14 のキースペースにハッシュします。その後、それらをグローバルなセカンダリインデックスから効率的に読み込むことができます。

最後に、前に定義したアクセスパターンを再度使用することができます。次に、新しい DynamoDB バージョンのアプリケーションで使用するアクセスパターンとクエリ条件の一覧を示します。

	アクセスパターン	クエリの状態
1	従業員 ID で従業員の詳細を検索する	テーブルのプライマリキー、ID="HR-EMPLOYEE"
2	従業員名で従業員の詳細をクエリする	GSI-1、PK="Employee Name" を使用する

	アクセスパターン	クエリの状態
3	従業員の現在のジョブの詳細のみを取得する	テーブルのプライマリキー、PK=HR-EMPLOYEE-1、SK は「JH」で始まる
4	日付範囲の顧客の注文を取得する	GSI-1、PK=CUSTOMER1、SK="STATUS-DATE" を StatusCode ごとに使用する
5	すべての顧客にわたって日付範囲内でステータスが OPEN のすべての注文を表示する	GSI-2、範囲 [0..N] での並列の PK=query、SK を OPEN-Date1 と OPEN-Date2 の間で使用する
6	最近雇用したすべての従業員	GSI-1、PK= "HR-CONFIDENTIAL'、SK > date1 を使用する
7	特定の倉庫のすべての従業員を検索する	GSI-1、PK=WAREHOUSE1 を使用する
8	倉庫の場所の在庫も含めて、商品のすべての注文項目を取得する	GSI-1、PK=PRODUCT1 を使用する
9	アカウント担当者別に顧客を取得する	GSI-1、PK=ACCOUNT-REP を使用する
10	アカウント担当者および日付別に注文を取得する	GSI-1、PK=ACCOUNT-REP、SK="STATUS-DATE" を StatusCode ごとに使用する
11	特定の役職を持つすべての従業員を取得する	GSI-1、PK=JOBTITLE を使用する
12	商品および倉庫別に在庫を取得する	テーブルのプライマリキー、PK=OE-PRODUCT1、SK=PRODUCT1

	アクセスパターン	クエリの状態
13	商品在庫総数を取得する	テーブルのプライマリキー、PK=OE-PRODUCT1、SK=PRODUCT1
14	注文合計および販売期間別にアカウント担当者をランク付けする	GSI-1、PK=YYYY-Q1、scanIndexForward=False を使用する

データのクエリとスキャンのベストプラクティス

このセクションでは、Amazon DynamoDB で Query オペレーションと Scan オペレーションを使用するためのベストプラクティスについて説明します。

スキャンのパフォーマンスに関する考慮事項

一般的に、Scan オペレーションは、DynamoDB の他のオペレーションよりも効率が低くなります。Scan オペレーションは常にテーブルまたはセカンダリインデックス全体をスキャンします。次に、値をフィルターして必要な結果を提供し、本質的に結果セットからデータを削除する余分なステップを追加します。

可能な場合、多くの結果を削除するフィルターを使用して大きなテーブルやインデックスで Scan オペレーションを実行することは避けるべきです。また、テーブルまたはインデックスが大きくなるにつれて、Scan オペレーションの処理速度は遅くなります。Scan オペレーションは、すべての項目でリクエストされた値を調べるので、大きなテーブルまたはインデックスに対してプロビジョニングされたスループットを 1 回のオペレーションで使い果たすことがあります。応答時間を短縮するには、アプリケーションが Scan ではなく Query を使用できるようにテーブルおよびインデックスを設計します。(テーブルの場合、GetItem API と BatchGetItem API を使用することを検討できます。)

リクエスト率への影響を最小限に抑える方法で Scan オペレーションを使用できるようアプリケーションを設計することもできます。これには、Scan オペレーションの代わりにグローバルセカンダリインデックスを使用する方が効率的な場合のモデリングが含まれる場合があります。このプロセスの詳細については、次の動画をご覧ください。

低速アクセスパターンのモデリング

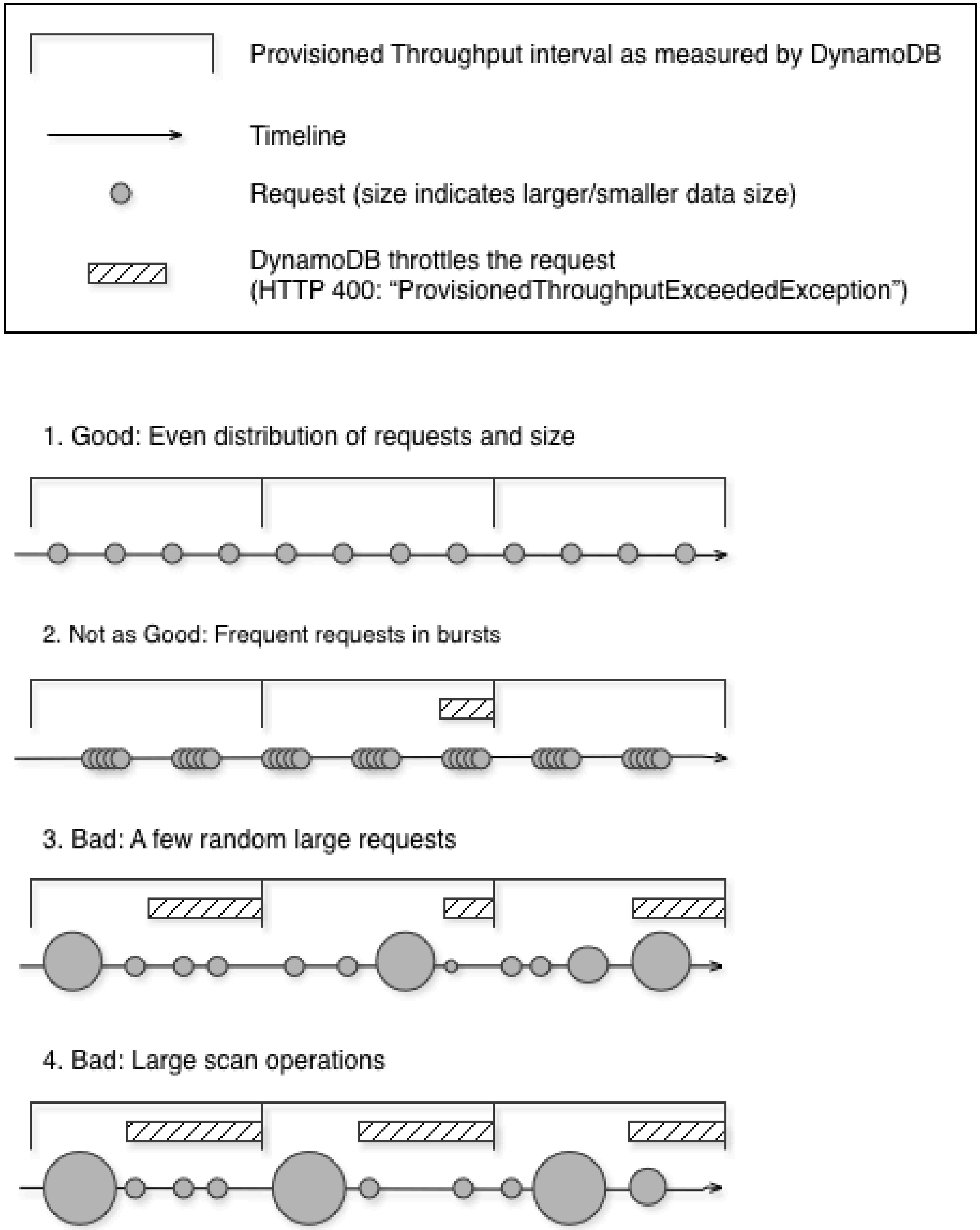
読み込みアクティビティの突然のスパイクの回避

テーブルを作成する際、読み込み容量ユニットと書き込み容量ユニットの要件を設定します。読み込みの場合、容量ユニットは、1 秒間に強い整合性のある 4 KB データ読み込みリクエストの数として表されます。結果整合性のある読み込みの場合、読み込み容量ユニットは 1 秒あたり 2 回の 4 KB 読み込みリクエストです。Scan オペレーションは、デフォルトで結果整合性のある読み込みを実行し、最大 1 MB (1 ページ) のデータを返すことができます。したがって、単一の Scan リクエストは、(1 MB ページサイズ/ 4 KB 項目サイズ)/2 (結果整合性のある読み込み) = 128 の読み込みオペレーションを使用できます。強力な整合性のある読み込みをリクエストした場合、Scan オペレーションでは、プロビジョニングされたスループット (256 回の読み込みオペレーション) が 2 倍消費されます。

これは、テーブルに設定されている読み込み容量と比較して使用量が急増することを表します。スキャンによる容量ユニットの使用により、同じテーブルに対する他の潜在的に重要なリクエストで利用可能な容量ユニットが使用できなくなります。その結果、これらのリクエストで ProvisionedThroughputExceeded 例外が発生する可能性が高くなります。

問題は、Scan が使用する容量ユニットの急激な増加だけではありません。スキャンは、パーティション上で隣り合うアイテムの読み込みをリクエストするため、同じパーティションのすべての容量ユニットを消費する可能性もあります。これは、リクエストが同じパーティションに到達し、そのすべての容量ユニットが消費され、そのパーティションへの他のリクエストがスロットリングされることを意味します。データの読み込みリクエストが複数のパーティションに分散されている場合、オペレーションは特定のパーティションをスロットリングしません。

次の図は、Query オペレーションと Scan オペレーションによって容量ユニットの使用率が急増したことによる影響、および同じテーブルに対するその他のリクエストへの影響を示しています。


Provisioned Throughput interval as measured by DynamoDB
Timeline
Request (size indicates larger/smaller data size)
DynamoDB throttles the request
(HTTP 400: “ProvisionedThroughputExceededException”)
1. Good: Even distribution of requests and size
2. Not as Good: Frequent requests in bursts
3. Bad: A few random large requests
4. Bad: Large scan operations

この図に示すように、使用率の急増は、テーブルのプロビジョニングされたスループットにいくつかの点で影響する可能性があります。

1. 良い: リクエストとサイズの均等な分散

2. あまり良くない: バーストでの頻繁なリクエスト
3. 悪い: いくつかのランダムな大きなリクエスト
4. 悪い: 大規模なスキャンオペレーション

大きな Scan オペレーションを使用する代わりに、次のテクニックを使用して、テーブルのプロビジョニングされたスループットに対するスキャンの影響を最小限に抑えることができます。

- ページサイズの削減

  スキャンオペレーションはページ全体 (デフォルトでは 1 MB) を読み込むため、ページサイズを小さくすることで、スキャンオペレーションの影響を軽減できます。Scan オペレーションは、リクエストのページサイズを設定するために使用できる Limit パラメータを提供します。同じようなページサイズの個々の Query または Scan リクエストでは使用される読み込みオペレーションの数が少なくなり、各リクエストの間で「一時停止」が作成されます。例えば、各項目が 4 KB で、ページサイズを 40 項目に設定するとします。Query リクエストは、結果整合性のある読み込み操作を 20 回しか消費せず、強く整合性のある読み込み操作を 40 回しか消費しません 多くの小さい Query または Scan オペレーションを使用すると、他の重要なリクエストをスロットリングなしで正常に完了できます。

- 複数のスキャンオペレーションの分離

  DynamoDB は、簡単にスケーリングできるように設計されています。その結果、アプリケーションは、複数の異なる目的のためにテーブルを作成することができます。場合によっては、複数のテーブル間でコンテンツを複製することも可能です。「ミッションクリティカル」なトラフィックを取らないテーブルでスキャンを実行する場合、アプリケーションによっては、2 つのテーブル間で、重要なトラフィックと記録用トラフィックなど、トラフィックを 1 時間ごとにローテーションすることで、このロードを処理します。他のアプリケーションでは、「ミッションクリティカル」なテーブルと「シャドウ」テーブルの 2 つのテーブルに対してすべての書き込みを実行することで、これを実行できます。

プロビジョニングされたスループットを超えたことを示すレスポンスコードを受け取ったリクエストを再試行するようにアプリケーションを設定します。または、UpdateTable オペレーションを使用して、テーブルのプロビジョニングされたスループットを増やします。ワークロードの一時的なスパイクが発生して、スループットがプロビジョニングされたレベルを超えた場合、エクスポネンシャルバックオフを使用してリクエストを再試行します。エクスポネンシャルバックオフの実装の詳細については、「エラーの再試行とエクスポネンシャルバックオフ」を参照してください。

並列スキャンを活用する

多くのアプリケーションでは、シーケンシャルスキャンよりも、並列 Scan オペレーションを使用することで利点が得られます。例えば、履歴データの大きなテーブルを処理するアプリケーションは、シーケンシャルデータよりもはるかに高速に並列スキャンを実行できます。バックグラウンドの「sweeper」プロセスに複数のワーカースレッドがあると、本番トラフィックに影響を与えずに、低い優先度でテーブルをスキャンできる可能性があります。これらの例では、並列 Scan は、プロビジョニングされたスループットリソースの他のアプリケーションが枯渇しないように使用されます。

並列スキャンは有益ですが、プロビジョニングされたスループットに大量のリクエストが発生する可能性があります。並列スキャンでは、アプリケーションには、Scan オペレーションが同時実行する複数のワーカーがあります。これにより、テーブルのプロビジョニングされた読み込み容量が短時間で消費されることがあります。その場合、テーブルにアクセスする必要がある他のアプリケーションでスロットリングが発生する可能性があります。

次の条件が満たされている場合、並列スキャンは正しい選択と考えられます。

- テーブルのサイズが 20 GB 以上である。
- テーブルのプロビジョニングされた読み込みスループットが完全に使用されていない。
- シーケンシャル Scan オペレーションが遅すぎる。

TotalSegments の選択

TotalSegments の最適な設定は、特定のデータ、テーブルのプロビジョニングされたスループット設定、およびパフォーマンス要件によって異なります。正しい設定を行うには、実験が必要な場合があります。データの 2 GB あたり 1 セグメントなど、単純な比率から始めることをお勧めします。例えば、30 GB のテーブルの場合、TotalSegments を 15 に設定してみます (30 GB/2 GB)。アプリケーションは 15 のワーカーを使用し、各ワーカーは異なるセグメントをスキャンします。

クライアントリソースに基づいて TotalSegments の値を選択することもできます。TotalSegments を 1～1000000 の任意の数に設定し、その数のセグメントを DynamoDB でスキャンできます。例えば、クライアントが同時に実行できるスレッドの数を制限している場合、アプリケーションで最高の Scan パフォーマンスを得るまで TotalSegments を段階的に増やすことができます。

並列スキャンをモニタリングして、プロビジョニングされたスループットの使用を最適化し、他のアプリケーションのリソースが不足していないことを確認します。プロビジョニングされたス

ループットがすべて消費されなくても Scan リクエストで引き続きスロットリングが発生する場合、TotalSegments の値を増やします。Scan リクエストが意図したものよりも多くのプロビジョニング済みスループットを消費する場合、TotalSegments の値を減らします。

DynamoDB テーブル設計のベストプラクティス

Amazon DynamoDB の一般的な設計原則では、使用するテーブルの数を最小限に抑えることをお勧めします。大部分のケースで、1 つのテーブルを使用することをお勧めします。ただし、1 つまたは少数のテーブルで対応できない場合は、以下のガイドラインを参考にしてください。

- 1 アカウントあたりのテーブル数は 10,000 が上限です。これを超えることはできません。アプリケーションでさらにテーブルが必要な場合は、テーブルを複数のアカウントに分散するように計画してください。詳細については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。
- テーブル管理に影響する可能性がある同時実行コントロールプレーンオペレーションについては、コントロールプレーンの制限を考慮します。
- AWS ソリューションアーキテクトと協力して、マルチテナント設計の設計パターンを検証します。

DynamoDB グローバルテーブル設計のベストプラクティス

グローバルテーブルは、DynamoDB のグローバルフットプリントに基づいて構築され、フルマネージド、マルチリージョン、マルチアクティブのデータベースにより、大規模にスケールされたグローバルアプリケーションの高速なローカルの読み取り/書き込みパフォーマンスを実現します。グローバルテーブルでは、選択した AWS リージョン間でデータが自動的にレプリケートされます。グローバルテーブルは既存の DynamoDB API を使用するため、アプリケーションを変更する必要はありません。グローバルテーブルの使用に伴う前払い料金やコミットメントはなく、使用したリソース分のみの支払いで済みます。

トピック

- グローバルテーブルを使用したリージョンからの退避
- グローバルテーブルのスループットキャパシティプランニング
- グローバルテーブルの準備チェックリストとよくある質問

DynamoDB グローバルテーブル設計の規範的ガイダンス

グローバルテーブルを効率的に使用するには、使用する書き込みモード、ルーティングモデル、退避プロセスなどの要素を慎重に検討する必要があります。すべてのリージョンにわたってアプリケーションを計測し、グローバルな健全性を維持するためのルーティングの調整や退避に備える必要があります。これにより、読み取りと書き込みが低レイテンシーで、サービスレベルアグリーメントが 99.999% のグローバルに分散されたデータセットが実現します。

DynamoDB グローバルテーブル設計に関する重要な事実

- グローバルテーブルには、現行バージョンであるグローバルテーブルバージョン 2019.11.21 (現行) (「V2」とも呼ばれます) と グローバルテーブルバージョン 2017.11.29 (レガシー) (「V1」とも呼ばれます) の 2 つのバージョンがあります。このガイドは、現行バージョンの V2 のみを対象としています。
- グローバルテーブルを使用しない場合、DynamoDB はリージョン別のサービスとなります。サービスは可用性が高く、リージョンでのインフラストラクチャの障害 (アベイラビリティーゾーン (AZ) 全体の障害を含む) に対する本質的な回復力を備えています。単一リージョンの DynamoDB テーブルは 99.99% の可用性 https://aws.amazon.com/dynamodb/sla/サービスレベルアグリーメント (SLA) を提供します。
- DynamoDB は、グローバルテーブルを使用することで、1 つのテーブルのデータを複数のリージョン間でレプリケートできます。マルチリージョンの DynamoDB テーブルは、99.999% の可用性 SLA を提供します。グローバルテーブルを適切に計画することは、リージョンの障害に対する回復力と耐性を備えたアーキテクチャの構築に役立ちます。
- グローバルテーブルは、アクティブ-アクティブレプリケーションモデルを採用しています。DynamoDB の観点から見ると、各リージョンのテーブルは読み取りと書き込みのリクエストを同じように受け入れることができます。ローカルレプリカテーブルは、書き込みリクエストを受け取ると、バックグラウンドで他の参加リージョンに書き込みをレプリケートします。
- 項目は個別にレプリケートされます。1 回のトランザクションで更新した複数の項目を、まとめてレプリケートすることはできません。
- ソースリージョンの各テーブルパーティションは、他のすべてのパーティションと並行して書き込みをレプリケートします。リモートリージョン内の書き込みの順序は、ソースリージョン内で発

生した書き込みの順序と一致しない場合があります。テーブルパーティションの詳細については、ブログ記事「DynamoDB のスケーリング: パーティション、ホットキー、ヒートに応じた分割がパフォーマンスに与える影響」を参照してください。

- 新しく書き込まれた項目は、通常、1 秒以内にすべてのレプリカテーブルに伝播されます。近くのリージョンにはより速く伝播される傾向があります。
- Amazon CloudWatch は、リージョンのペアごとに `ReplicationLatency` メトリクスを提供します。到着した項目の到着時間と最初の書き込み時間を比較し、平均を算出することでメトリクスが計算されます。タイミングはソースリージョンの CloudWatch 内に保存されます。平均タイミングと最大タイミングを表示すると、平均および最悪の場合のレプリケーションラグを判断するのに役立ちます。このレイテンシーには SLA はありません。
- 同じ項目が 2 つの異なるリージョンでほぼ同時に (この `ReplicationLatency` ウィンドウ内で) 更新され、最初の書き込みがレプリケートされる前に次の書き込みが行われると、書き込みが競合する可能性があります。グローバルテーブルは、書き込みのタイムスタンプに基づく最終書き込み者優先メカニズムにより、このような競合を解決します。1 回目の書き込みよりも 2 回目の書き込みが「優先」されます。これらの競合は CloudWatch または AWS CloudTrail には記録されません。
- 各項目には、最終書き込みタイムスタンプがプライベートシステムプロパティとして保持されます。最終書き込み者優先アプローチを実装するには、着信する項目のタイムスタンプが既存の項目のタイムスタンプよりも大きいことを要求する条件付き書き込みを使用します。
- グローバルテーブルは、すべての項目をすべての参加リージョンにレプリケートします。複数の異なるレプリケーションスコープが必要な場合は、複数の異なるテーブルを作成し、テーブルごとに異なる参加リージョンを割り当てることができます。
- レプリカリージョンがオフラインであったり、`ReplicationLatency` が高くなったりしても、ローカルリージョンへの書き込みは受け入れられます。ローカルテーブルは、各項目が成功するまで、リモートテーブルへの項目のレプリケーションを試行し続けます。
- 万が一、リージョンが完全にオフラインになった場合でも、後でオンラインに戻ったときに、すべての保留中のアウトバウンドレプリケーションとインバウンドレプリケーションが再試行されます。テーブルを同期状態に戻すために特別なアクションは必要ありません。最終書き込み者優先メカニズムにより、データは結果的に整合します。
- DynamoDB テーブルには、いつでも新しいリージョンを追加できます。DynamoDB は、初期同期と進行中のレプリケーションを処理します。リージョンを削除すると、それが元のリージョンであっても、そのリージョンのテーブルのみが削除されます。
- DynamoDB にはグローバルエンドポイントはありません。すべてのリクエストは、リージョンのエンドポイントに対して行います。このエンドポイントからリージョンのローカルなグローバルテーブルインスタンスにアクセスします。

- DynamoDB への呼び出しはリージョンを越えないようにします。ベストプラクティスとして、1 つのリージョンのコンピューティングレイヤーは、そのリージョンのローカル DynamoDB エンドポイントにのみ直接アクセスするようにします。リージョン内で問題が検出された場合は、問題が DynamoDB レイヤーにあるか周囲のスタックにあるかにかかわらず、エンドユーザーのトラフィックを別のリージョンでホストされている別のコンピューティングレイヤーにルーティングする必要があります。グローバルテーブルのレプリケーションのおかげで、別のリージョンには、ローカルで使用する同じデータのローカルコピーが既に存在します。状況によっては、あるリージョンのコンピューティングレイヤーから別のリージョンのコンピューティングレイヤーにリクエストを渡して処理することがありますが、この場合、リモート DynamoDB エンドポイントに直接アクセスすべきではありません。この特別なユースケースの詳細については、「コンピューティングレイヤーのリクエストルーティング」を参照してください。

ユースケース

グローバルテーブルには、以下のような一般的な利点があります。

- 読み取りレイテンシーの短縮。データのコピーをエンドユーザーの近くに配置して、読み取り時のネットワークレイテンシーを短縮できます。キャッシュは `ReplicationLatency` 値と同じ最新の状態に保たれます。
- 書き込みレイテンシーの短縮。近くのリージョンに書き込むことで、ネットワークレイテンシーと書き込みにかかる時間を短縮できます。書き込みトラフィックは、競合が発生しないように慎重にルーティングする必要があります。ルーティングの手法の詳細については、「グローバルテーブルを使用したリクエストルーティング」を参照してください。
- 回復力とディザスタリカバリの向上。リージョンのパフォーマンス低下や完全な停止が発生した場合、リージョンから退避する (そのリージョンへのリクエストの一部またはすべてを移動する) ことができます。これに伴う目標復旧時点 (RPO) と目標復旧時間 (RTO) は秒単位で測定されます。また、グローバルテーブルを使用すると、DynamoDB の SLA が 99.99% から 99.999% に向上します。
- シームレスなリージョンの移行。新しいリージョンを追加してから古いリージョンを削除し、リージョン間でデプロイを移行できます。これに伴うデータレイヤーでのダウンタイムは発生しません。例えば、注文管理システムに DynamoDB グローバルテーブルを使用すると、AZ やリージョンの障害に対する回復力を維持しながら、低レイテンシーの処理を高い信頼性とスケールで実現できます。

グローバルテーブルによる書き込みモード

グローバルテーブルは、テーブルレベルでは常にアクティブ/アクティブです。ただし、書き込みリクエストのルーティング方法を制御して、アクティブ/パッシブとして扱うこともできます。例えば、書き込みが競合しないように、書き込みリクエストを単一のリージョンにルーティングすることを決定する場合があります。

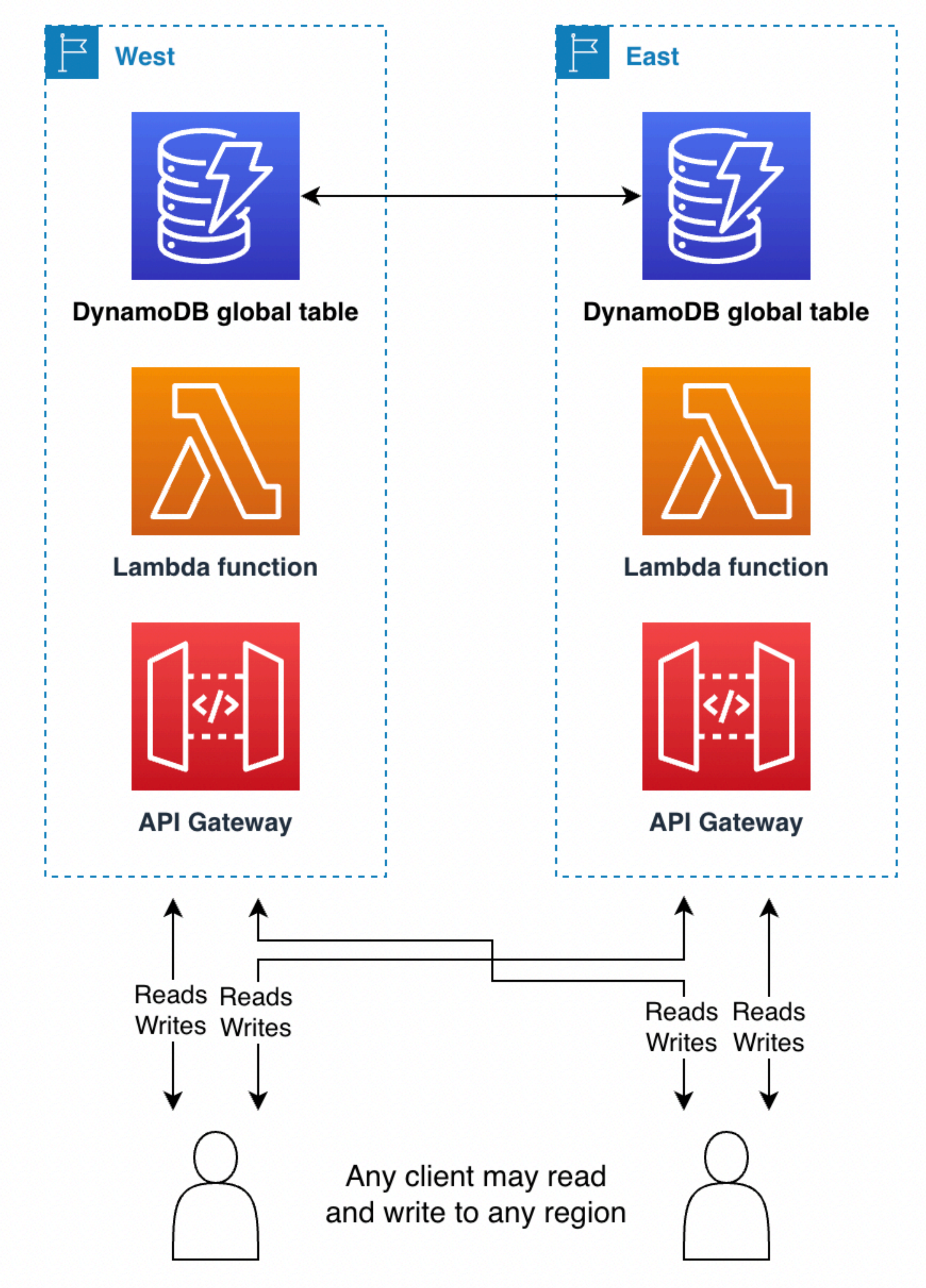
マネージドの書き込みパターンは、主に以下の 3 つに分類されます。

- 任意のリージョンへの書き込みモード (プライマリなし)
- 1 つのリージョンへの書き込みモード (単一プライマリ)
- 自分のリージョンへの書き込みモード (混合プライマリ)

どの書き込みパターンがユースケースに適しているかを検討する必要があります。この選択は、リクエストのルーティング、リージョンの退避、ディザスタリカバリの処理方法に影響します。全体的なベストプラクティスは、アプリケーションの書き込みモードによって異なる場合があります。

任意のリージョンへの書き込みモード (プライマリなし)

任意のリージョンへの書き込みモードは、完全にアクティブ/アクティブであり、書き込みが発生する場所に制限はありません。任意のリージョンにいつでも書き込むことができます。これは最も単純なモードです。このモードは、一部のタイプのアプリケーションでのみ使用できます。すべてのライターが冪等であるため、安全に繰り返し実行可能であり、リージョンをまたいでの同時または繰り返しの書き込みオペレーションが競合しないという場合に適しています。例えば、ユーザーが連絡先データを更新する場合などに使用できます。このモードは、冪等である特殊なケースとして、確定的なプライマリキーにおけるすべての書き込みが一意の挿入であるという、追加専用のデータセットにも適しています。最後に、このモードは、書き込みの競合リスクを許容できる場合に適しています。



任意のリージョンへの書き込みモードは、実装が最も簡単なアーキテクチャです。いつでも任意のリージョンを書き込み先にすることができるため、ルーティングが容易になります。最近の書き込

みを任意のセカンダリリージョンに何度でもリプレイできるため、フェイルオーバーが容易になります。可能な限り、この書き込みモード向けの設計を行うようにします。

例えば、ビデオストリーミングサービスでは、ブックマーク、レビュー、視聴ステータスフラグなどの追跡にグローバルテーブルをよく使用します。これらのデプロイで任意のリージョンへの書き込みモードを使用できるのは、すべての書き込みが冪等で、項目の次の正しい値が現在の値に依存しない場合に限ります。これは、最新のタイムコードの新規設定、新しいレビューの割り当て、新しい視聴ステータスの設定など、ユーザーの新しいステータスを直接割り当ててユーザーを更新する場合に当てはまります。ユーザーの書き込みリクエストが複数の異なるリージョンにルーティングされた場合、最後の書き込みオペレーションが保持され、最後の割り当てに従ってグローバル状態が確定します。このモードでの読み取りオペレーションは、最新の `ReplicationLatency` 値の遅延後に、結果的に整合します。

別の例として、ある金融サービス会社では、システムの一部としてグローバルテーブルを使用し、各顧客のデビットカード購入の累計を管理して、その顧客のキャッシュバック特典を計算しています。新しいトランザクションは世界中から流入し、複数のリージョンに送られます。グローバルテーブルを利用しない現在の設計では、顧客ごとに 1 つの収支項目を使用しています。顧客のアクションに応じて残高が ADD 式で更新されます。ただし、新しい正しい値は現在の値によって異なるため、この式は冪等ではありません。つまり、複数の異なるリージョンで同じ残高に対してほぼ同時に 2 つの書き込みオペレーションが行われた場合、残高が同期しなくなります。

この同じ会社は、DynamoDB のグローバルテーブルを使用して慎重に再設計することで、任意のリージョンへの書き込みモードを実現できます。新しい設計では、「イベントストリーミング」モデル、つまり実質的には追加専用のワークフローを持つ台帳を採用できます。顧客のアクションごとに、その顧客用に管理されている項目コレクションに新しい項目が追加されます。項目コレクションは、プライマリキーを共有し、異なるソートキーを持つ項目のセットです。顧客のアクションを追加する各書き込みアクションは、冪等挿入であり、顧客 ID をパーティションキーとして使用し、トランザクション ID をソートキーとして使用します。この設計では、項目をプルしてからクライアント側で計算を行うために `Query` が必要になり、残高の計算がより複雑になります。ただし、すべての書き込みが冪等になるため、ルーティングとフェイルオーバーが大幅に簡素化されるという利点があります。詳細については、「グローバルテーブルを使用したリクエストルーティング」を参照してください。

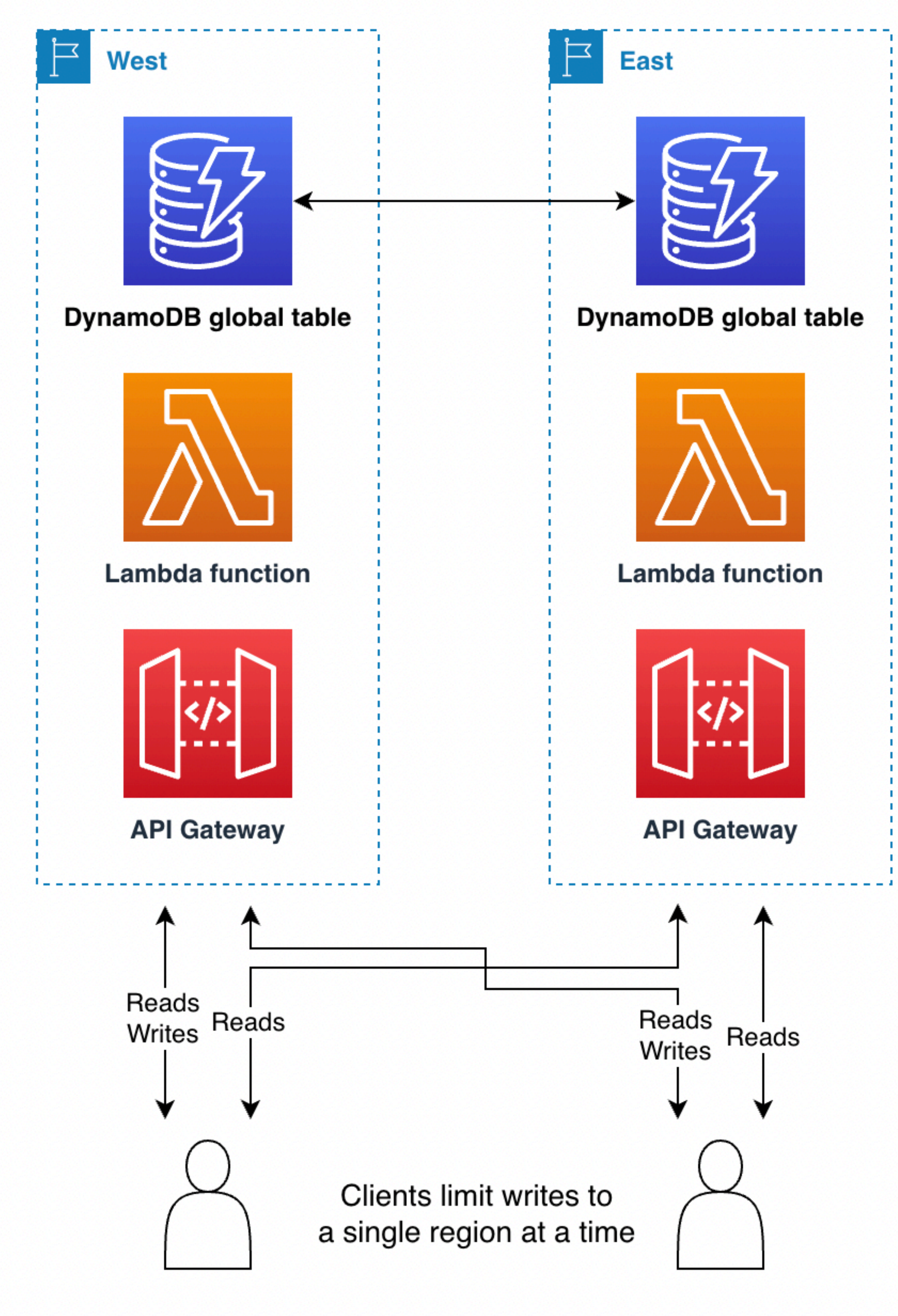
3 番目の例として、オンライン広告を掲載している企業があるとします。この企業は、任意のリージョンへの書き込みモードによる設計の簡素化を実現するには、低リスクのデータ損失は許容できると判断しました。広告を配信する際には、わずか数ミリ秒で十分なメタデータを取得してどの広告を表示するか決定し、広告インプレッションを記録して、ユーザーに同じ広告が繰り返し表示されないようにしています。グローバルテーブルを使用すると、世界中のエンドユーザーに対して低レイテ

ンシーの読み取りと低レイテンシーの書き込みの両方を提供できます。ユーザーのすべての広告インプレッションを 1 つの項目内に記録し、それを成長するリストとすることができます。項目コレクションに追加する代わりに 1 つの項目を使用できます。これにより、各書き込みの一部として古い広告インプレッションを削除できるため、削除に料金を支払う必要がありません。この書き込みオペレーションは冪等ではないため、同じエンドユーザーが複数のリージョンからほぼ同時に配信された広告を見た場合、ある広告インプレッションの書き込みが他の広告インプレッションの書き込みを上書きする可能性があります。オンライン広告の場合、ユーザーが広告を繰り返し見るリスクがあるとしても、このシンプルで効率的な設計を採用する価値があります。

単一プライマリ (「1 つのリージョンへの書き込み」)

1 つのリージョンへの書き込みモードはアクティブ/パッシブで、すべてのテーブル書き込みを 1 つのアクティブリージョンにルーティングします。DynamoDB には 1 つのアクティブリージョンという概念がないことに注意してください。DynamoDB の外部のアプリケーションルーティングがこれを管理します。1 つのリージョンへの書き込みモードは、一度に 1 つのリージョンにのみ書き込むようにすることで、書き込みの競合を回避します。この書き込みモードは、条件式やトランザクションを使用する場合に役立ちます。条件式やトランザクションは、最新であることが明白なデータに適応しないと、意味がないためです。したがって、条件式とトランザクションを使用するには、すべての書き込みリクエストを最新のデータがある 1 つのリージョンに送信する必要があります。

結果整合性のある読み込みは、任意のレプリカリージョンに送信してレイテンシーを短縮できます。強力な整合性のある読み込みは、単一のプライマリリージョンに送信する必要があります。


West
DynamoDB global table
Lambda function
API Gateway
East
DynamoDB global table
Lambda function
API Gateway
Reads
Writes
Reads
Reads
Writes
Reads
Clients limit writes to
a single region at a time

リージョンの障害に対応するために、アクティブなリージョンを変更してデータを処理することが必要になる場合があります。グローバルテーブルを使用したリージョンからの退避は、このユースケースの一例です。「フォローザサン」デプロイなどで、現在アクティブなリージョンを定期的に変更する場合もあります。これにより、アクティブなリージョンが最もアクティビティの多い地域の近くに配置され、読み取りと書き込みのレイテンシーが最小限に抑えられます。これには、リージョンを変更するコードパスを毎日呼び出すことで、ディザスタリカバリの前にコードパスが十分にテスト済みであることを確認できるという、副次的な利点もあります。

パッシブリージョンでは、DynamoDB を取り巻くインフラストラクチャをダウンスケールしたままにして、アクティブリージョンになったときにのみ構築を行う場合があります。パイロットライトとウォームスタンバイの設計の詳細については、「AWS でのディザスタリカバリ (DR) アーキテクチャ、パート III: パイロットライトとウォームスタンバイ」を参照してください。

グローバルテーブルを活用して低レイテンシーのグローバル分散読み取りを行う場合は、1 つのリージョンへの書き込みモードを使用すると効果的です。例えば、大規模なソーシャルメディア会社には、数百万のユーザーと数十億の投稿があります。各ユーザーは、アカウント作成時に、自分の所在地と地理的に近いリージョンに割り当てられます。ユーザーのすべてのデータは、そのリージョンの非グローバルテーブルに格納されます。同社では、1 つのリージョンへの書き込みモードを使用し、別個のグローバルテーブルでユーザーのホームリージョンへのマッピングを保持しています。読み取り専用のコピーが世界中に保存されるため、レイテンシーを最小限に抑えながら、各ユーザーのデータを直接検索できます。更新はまれであり (ユーザーのホームリージョンを別のリージョンに移動する場合のみ)、必ず 1 つのリージョンを経由して書き込むため、書き込みの競合を回避できます。

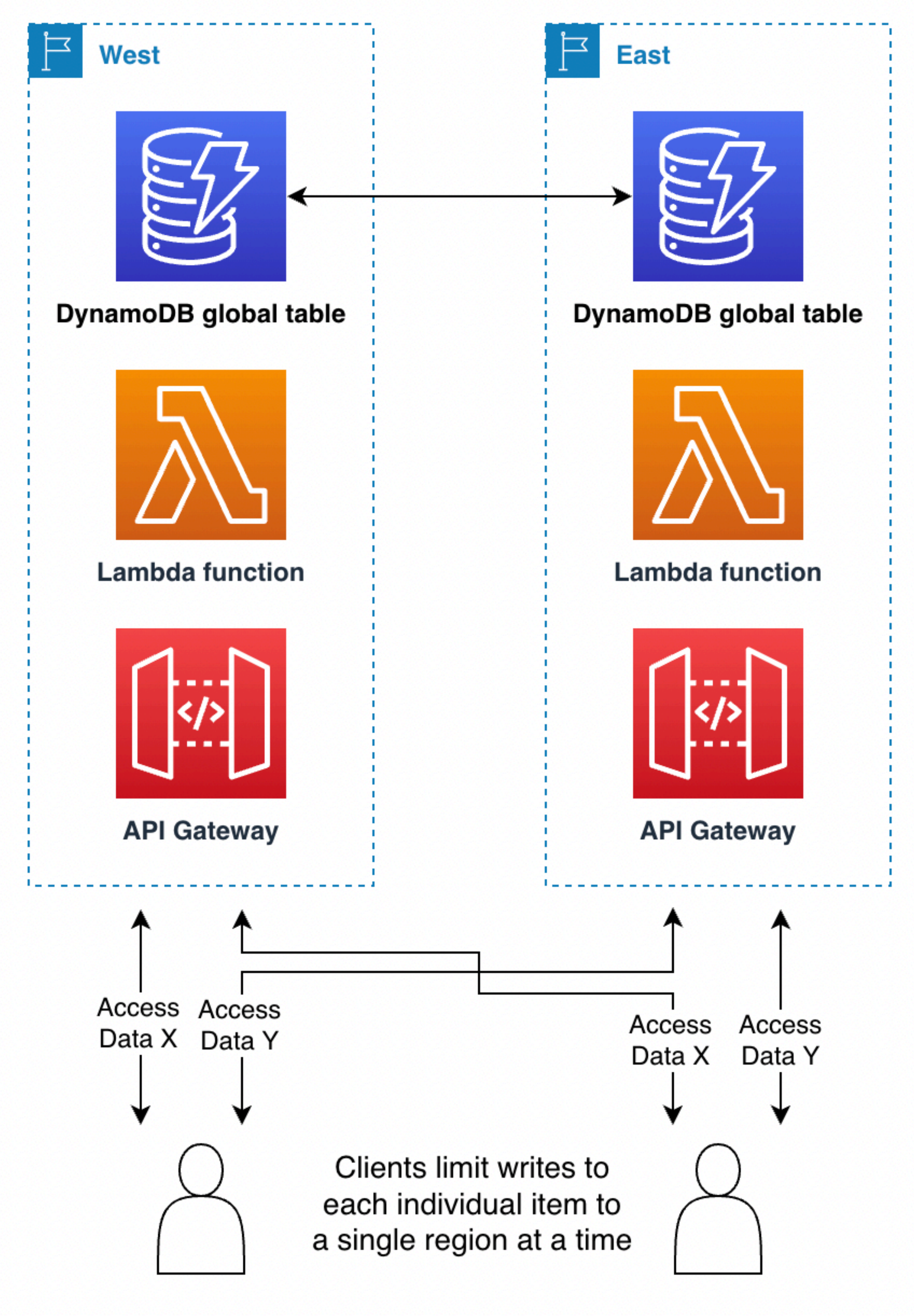
別の例として、毎日のキャッシュバック計算を実装した金融サービス会社を考えてみましょう。同社は、残高の計算には任意のリージョンへの書き込みモードを使用していますが、実際のキャッシュバック支払いの追跡には 1 つのリージョンへの書き込みモードを使用しています。1 日に 10 ドル使うごとに 1 ペニーの報酬を顧客に提供するには、前日のすべてのトランザクションに対して `Query` を実行して、支出総額を計算し、キャッシュバックを決定して新しいテーブルに書き込み、クエリした項目のセットを削除する必要があります。これを消費済みとしてマークし、単一の項目に置き換えて翌日の計算に使用する残額を格納します。この作業にはトランザクションが必要であるため、1 つのリージョンへの書き込みモードを使用すると、より効率的です。ワークロードが重複する可能性がない限り、アプリケーションは同じテーブルに対しても複数の書き込みモードを一緒に使用できます。

混合プライマリ (「自分のリージョンへの書き込み」)

自分のリージョンへの書き込みモードでは、データサブセット別に異なるリージョンに割り当て、ホームリージョンを介した項目への書き込みオペレーションのみを許可します。このモードはアク

ティブ/パッシブですが、項目に応じてアクティブリージョンを割り当てます。各リージョンは重複しない独自のデータセットのプライマリであり、適切なローカリティを確保するために書き込みを保護する必要があります。

このモードは 1 つのリージョンへの書き込みと似ていますが、各エンドユーザーに関連するデータを、そのユーザーに近いネットワークに配置できるため、書き込みのレイテンシーを短縮できる点が異なります。また、周囲のインフラストラクチャがリージョン間でより均等に分散され、すべてのリージョンでインフラストラクチャの一部が既にアクティブになっているため、フェイルオーバーシナリオ時にインフラストラクチャを構築する作業が少なくなります。



項目のホームリージョンは、さまざまな方法で決定できます。

- 固有: パーティションキーなど、データの一部の側面により、データがどのリージョンをホームとするかが明確になります。例えば、顧客とその顧客に関するすべてのデータには、顧客データ内で、特定のリージョンをホームとすることがマークされます。この手法については、「リージョンのピン留めを使用して Amazon DynamoDB グローバルテーブル内の項目のホームリージョンを設定する」で説明しています。
- ネゴシエート済み: 各データセットのホームリージョンは、割り当てを管理する別のグローバルサービスなど、何らかの外部的な方法でネゴシエート済みです。割り当てには有限の期間があり、その後は再ネゴシエーションの対象となります。
- テーブル指向: レプリケートするグローバルテーブルを 1 つ作成するのではなく、レプリケートするリージョンの数だけグローバルテーブルを作成します。各テーブルの名前はホームリージョンを示します。標準オペレーションでは、すべてのデータがホームリージョンに書き込まれ、他のリージョンは読み取り専用のコピーを保持します。フェイルオーバー中、別のリージョンがそのテーブルの書き込みタスクを一時的に引き継ぎます。

例えば、ゲーム会社で働いているとしましょう。世界中のすべてのゲーマーに、低レイテンシーの読み取りと書き込みが必要です。各ゲーマーのホームは、最も近いリージョンに設定できます。このリージョンですべての読み取りと書き込みが行われるため、強力な整合性のある読み込みと書き込みが常に確保されます。ただし、ゲーマーが旅行したり、ホームリージョンが停止したりした場合は、データの完全なコピーが別のリージョンで利用できるようになります。したがって、必要に応じてゲーマーを別のホームリージョンに割り当てることができます。

別の例として、ビデオ会議会社で働いているとしましょう。各電話会議のメタデータは特定のリージョンに割り当てられます。発信者は、最も近いリージョンを利用してレイテンシーを最小限に抑えることができます。リージョンが停止した場合、グローバルテーブルを使用すると、システムは、データのレプリケートされたコピーが既に存在する別のリージョンに呼び出しの処理を移動できるため、すばやく回復できます。

グローバルテーブルを使用したリクエストルーティング

グローバルテーブルのデプロイで最も複雑な部分は、おそらくリクエストルーティングの管理です。リクエストは、まずエンドユーザーから、何らかの方法で経路が選定されたリージョンに送る必要があります。リクエストは、そのリージョンでサービスのスタックに遭遇します。これには、AWS Lambda 関数、コンテナ、または Amazon Elastic Compute Cloud (Amazon EC2) ノードでバッキングされたロードバランサーを構成要素とするコンピューティングレイヤーや、場合によっては他のサービス (別のデータベースなど) が含まれます。このコンピューティングレイヤーは DynamoDB と通信します。この通信は、そのリージョンのローカルエンドポイントを使用して行う必要がありま

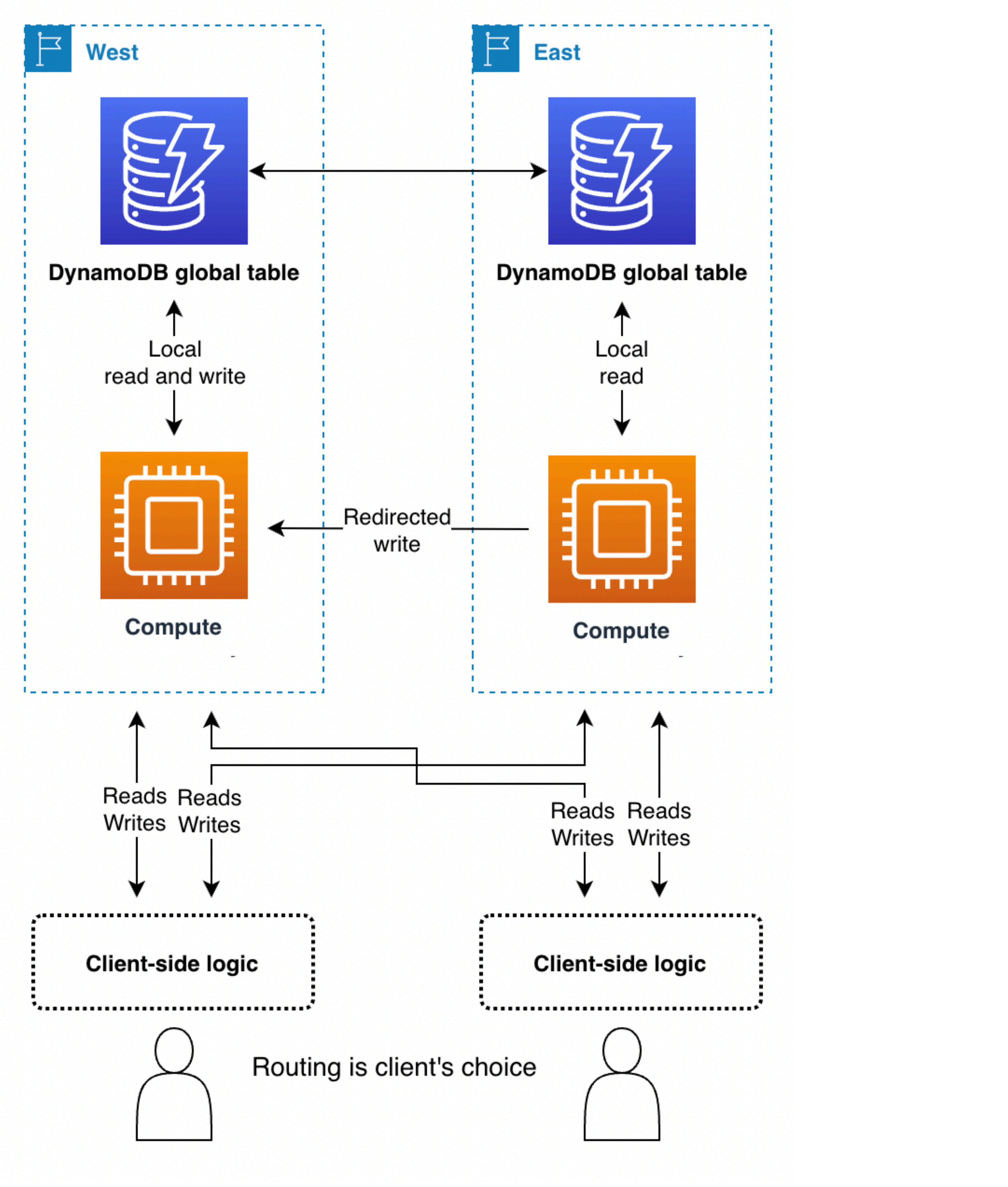
す。グローバルテーブルのデータは、参加している他のすべてのリージョンにレプリケートされ、各リージョンには DynamoDB テーブルを中心に同様のサービスのスタックがあります。

グローバルテーブルは、さまざまなリージョンの各スタックに同じデータのローカルコピーを提供します。1 つのリージョンの 1 つのスタックを対象とした設計を行い、ローカルの DynamoDB テーブルに問題が発生した場合は、セカンダリリージョンの DynamoDB エンドポイントにリモート呼び出しを行うということも考えられます。しかし、これはベストプラクティスではありません。リージョンをまたいだ場合のレイテンシーは、ローカルアクセスの場合より 100 倍も高くなる可能性があります。5 回のリクエストを往復する場合、ローカルの実行は数ミリ秒で済みますが、地球を横断する場合は数秒かかることがあります。エンドユーザーを別のリージョンにルーティングして処理することをお勧めします。レジリエンスを確保するには、マルチリージョンのレプリケーションが必要であり、データレイヤーだけでなくコンピューティングレイヤーのレプリケーションも必要です。

エンドユーザーのリクエストをリージョンにルーティングして処理するための代替手法は数多くあります。最適な選択は、書き込みモードとフェイルオーバーに関する考慮事項によって異なります。このセクションでは、クライアント主導、コンピューティングレイヤー、Route 53、Global Accelerator の 4 つのオプションについて説明します。

クライアント主導のリクエストルーティング

クライアント主導のリクエストルーティングでは、アプリケーションなどのエンドユーザークライアント、JavaScript を使用したウェブページ、または別のクライアントにより、有効なアプリケーションエンドポイントを追跡します。この場合、リテラルの DynamoDB エンドポイントではなく、Amazon API Gateway などのアプリケーションエンドポイントが対象になります。エンドユーザークライアントは、独自の組み込みロジックを使用して、どのリージョンと通信するかを選択します。この選択は、ランダムな選択、観測された最低のレイテンシー、観測された最大の帯域幅測定値、またはローカルで実行されたヘルスチェックに基づいて行うことができます。


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Reads
Writes
Reads
Writes
Reads
Writes
Reads
Writes
Client-side logic
Client-side logic
Routing is client's choice

クライアント主導のリクエストルーティングの利点は、パフォーマンスの低下に気付いた場合に、実際のパブリックインターネットのトラフィック状況などに対応して、リージョンを切り替えることができることです。クライアントは、すべての潜在的なエンドポイントを認識している必要がありますが、新しいリージョンエンドポイントの立ち上げは頻繁に起こることではありません。

任意のリージョンへの書き込みモードでは、クライアントが優先エンドポイントを一方的に選択できます。1 つのリージョンへのアクセスが損なわれた場合、クライアントは別のエンドポイントにルーティングできます。

1 つのリージョンへの書き込みモードでは、クライアントは現在アクティブなリージョンに書き込みをルーティングするメカニズムが必要になります。これは、どのリージョンが現在書き込みを受け入れているかを経験的にテストする (書き込み拒否を確認して別のリージョンにフォールバックする) という基本的なものから、グローバルコーディネーターを呼び出して現在のアプリケーションの状態をクエリするという複雑なものまであります。後者の場合は、このようなニーズに対応してグローバル状態を維持するために、Route 53 Application Recovery Controller (ARC) ルーティング制御に基づいて構築し、5 リージョンのクォーラム駆動型システムを提供するようなメカニズムとなります。クライアントは、読み取りを任意のリージョンに送って結果整合性を確保するのか、アクティブなリージョンにルーティングして強力な整合性を確保するのかを決定できます。詳細については、「Route 53 の仕組み」を参照してください。

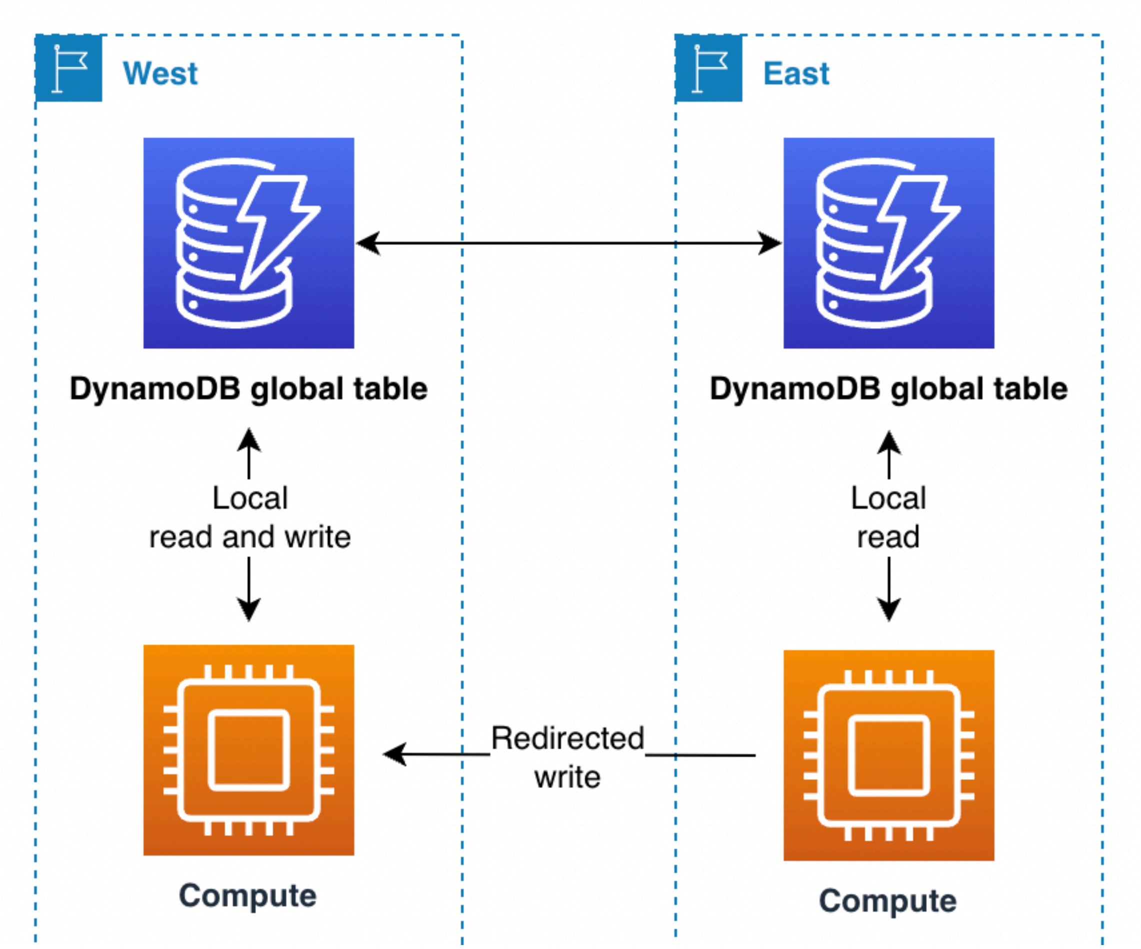
自分のリージョンへの書き込みモードでは、クライアントは使用するデータセットのホームリージョンを決定する必要があります。例えば、クライアントがユーザーアカウントと一致し、ユーザーアカウントごとに 1 つのリージョンをホームとする場合、クライアントはグローバルログインシステムに対して適切なエンドポイントをリクエストできます。

例えば、ユーザーがウェブ経由で財務を管理できるよう支援する金融サービス会社では、自分のリージョンへの書き込みモードでグローバルテーブルを使用できます。各ユーザーは中央サービスにログインする必要があります。このサービスは、認証情報と共に、認証情報が機能するリージョンのエンドポイントを返します。認証情報の有効期間は短期です。有効期間後に、ウェブページは新しいログインを自動ネゴシエートし、ユーザーのアクティビティを新しいリージョンにリダイレクトする機会を提供します。

コンピューティングレイヤーのリクエストルーティング

コンピューティングレイヤーのリクエストルーティングでは、コンピューティングレイヤーで実行されているコードが、リクエストをローカルで処理するか、別のリージョンで実行されているそれ自体のコピーに渡すかを決定します。1 つのリージョンへの書き込みモードを使用する場合、コンピューティングレイヤーは、そのリージョンがアクティブリージョンではないことを検出すると、ローカルの読み取りオペレーションを許可する一方で、すべての書き込みオペレーションを別のリージョンに

転送する場合があります。このコンピューティングレイヤーコードは、データトポロジとルーティングルールを認識する必要があります。また、どのリージョンをどのデータに対してアクティブにするかを指定する最新の設定に基づいて、これらのデータトポロジとルーティングルールを確実に適用する必要があります。リージョン内の外側のソフトウェアスタックは、読み取りリクエストと書き込みリクエストがマイクロサービスによってどのようにルーティングされるかを認識する必要はありません。堅牢な設計では、受信側リージョンが書き込みオペレーションの現在のプライマリであるかどうかを検証します。プライマリでない場合は、グローバル状態を修正する必要があることを示すエラーが生成されます。プライマリリージョンが変更中の場合、受信側リージョンが書き込みオペレーションをしばらくバッファリングすることもあります。いずれの場合も、リージョン内のコンピューティングスタックはローカルの DynamoDB エンドポイントにのみ書き込みますが、コンピューティングスタック間で相互に通信する可能性があります。

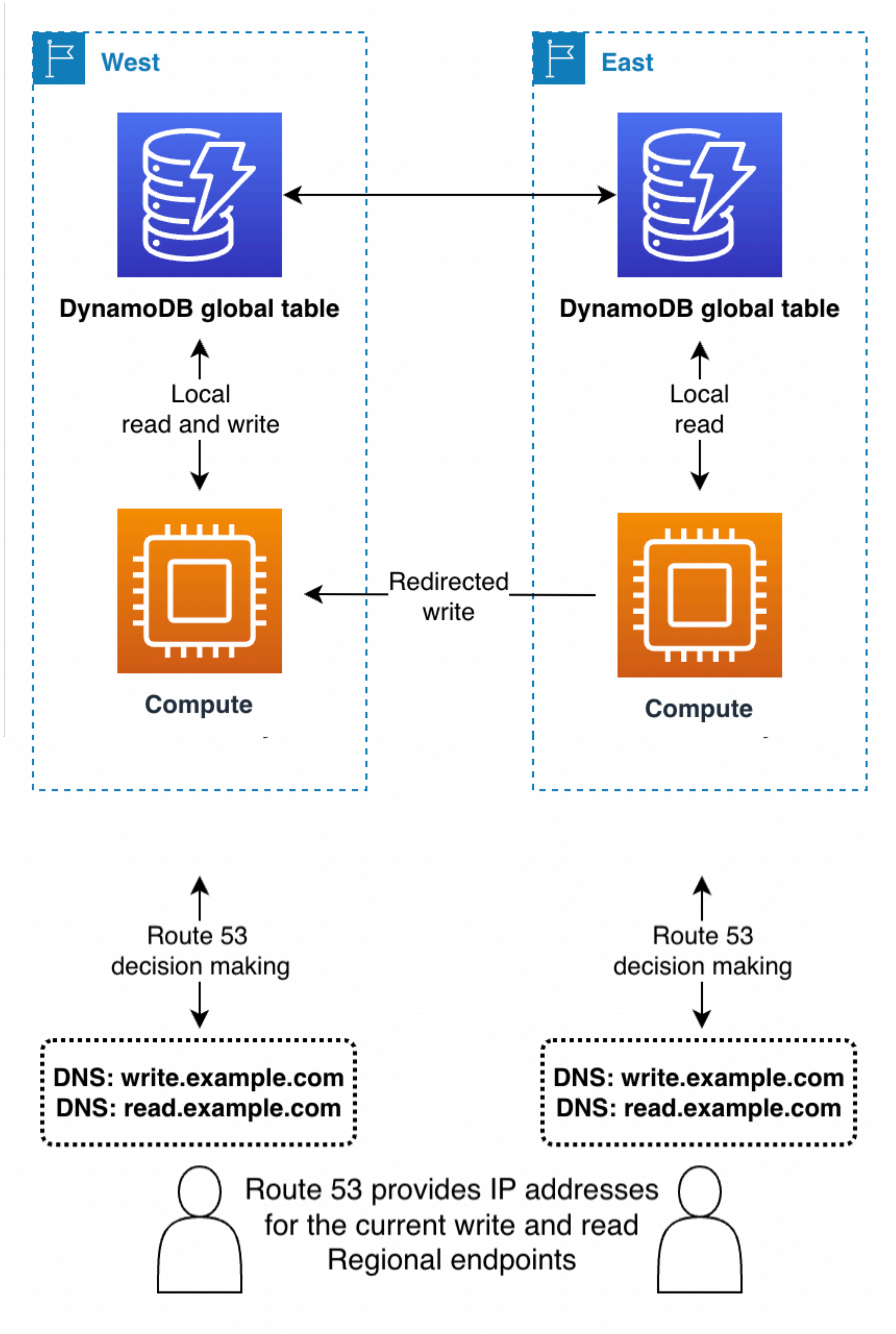


このシナリオでは、金融サービス会社がフォローザサン型の単一プライマリモデルを使用しているとします。このルーティングプロセスにはシステムとライブラリを使用します。システム全体は、AWS の Route 53 ARC ルーティング制御と同様に、グローバル状態を維持します。グローバルテーブルを使用して、どのリージョンがプライマリリージョンで、次のプライマリの切り替えがいつ予定されているかを追跡します。すべての読み取りおよび書き込みオペレーションは、システム

と連携するライブラリを経由します。このライブラリを使用すると、読み取りオペレーションを低レイテンシーでローカルに実行できます。書き込みオペレーションの場合、アプリケーションはローカルリージョンが現在のプライマリリージョンかどうかをチェックします。プライマリリージョンである場合、書き込みオペレーションは直接完了します。そうでない場合、書き込みタスクは現在のプライマリリージョンにあるライブラリに転送されます。受信側のライブラリは、自身もプライマリリージョンであると認識したことを確認します。そうでない場合は、エラーを生成します。これにより、グローバル状態の伝播が遅延します。このアプローチでは、リモート DynamoDB エンドポイントに直接書き込まないため、検証上の利点があります。

Route 53 のリクエストルーティング

Amazon Route 53 Application Recovery Controller は、ドメインネームサービス (DNS) テクノロジーです。Route 53 の場合、クライアントは既知の DNS ドメイン名を検索してエンドポイントをリクエストし、Route 53 は最も適切と思われるリージョンエンドポイントに対応する IP アドレスを返します。Route 53 には、適切なリージョンを決定するために使用するルーティングポリシーのリストがあります。Route 53 は、フェイルオーバールーティングを実行して、ヘルスチェックに失敗したリージョンからトラフィックをルーティングすることもできます。


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Compute
Compute
Redirected
write
Route 53
decision making
Route 53
decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Route 53 provides IP addresses
for the current write and read
Regional endpoints

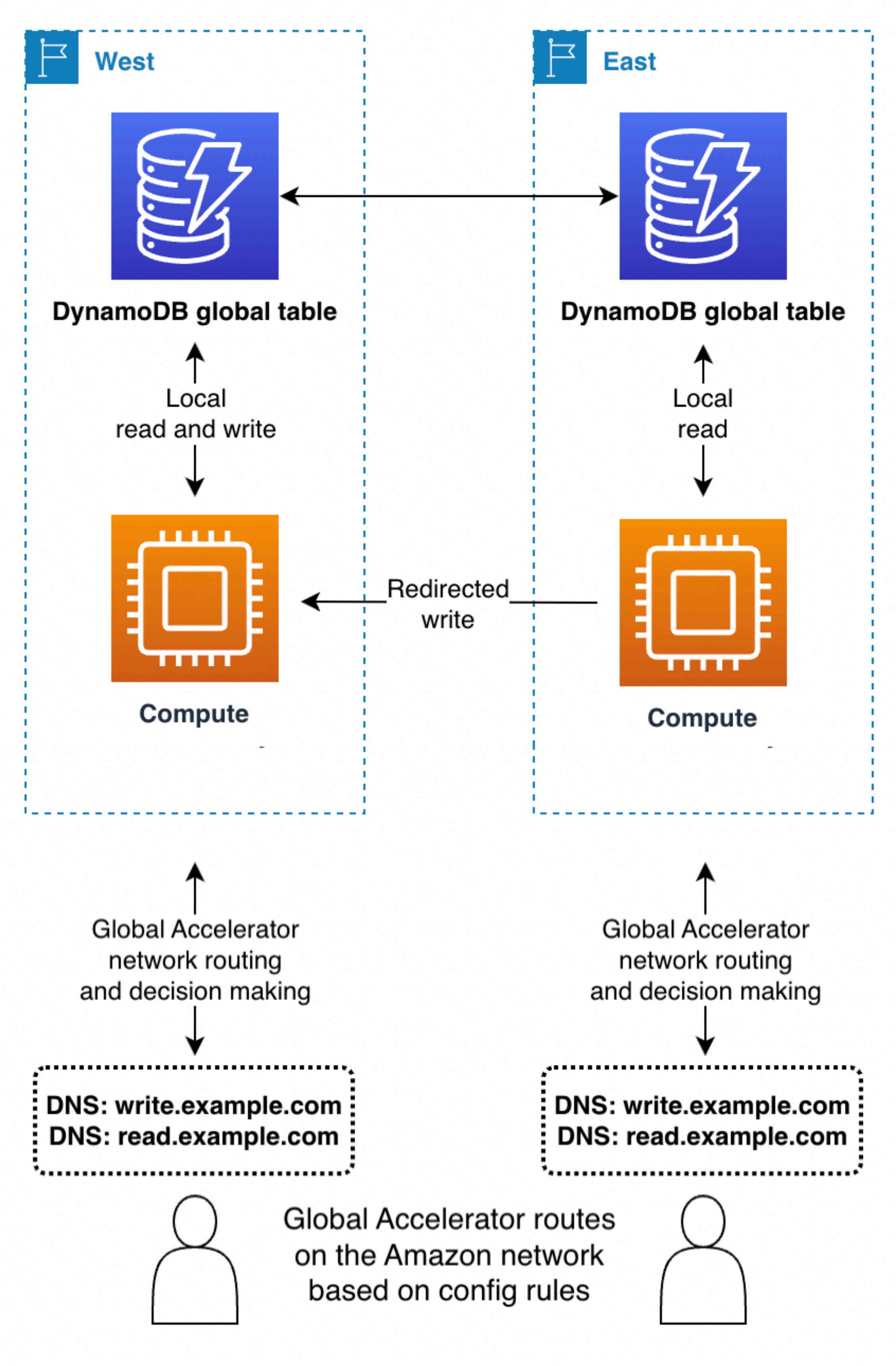
- 任意のリージョンへの書き込みモードを使用するか、バックエンドでコンピューティングレイヤーのリクエストルーティングと組み合わせることで、ネットワークに最も近いリージョン、地理的に最も近いリージョンなど、複雑な内部ルールに基づいてリージョンを返すフルアクセスを Route 53 に許可できます。
- 1 つのリージョンへの書き込みモードでは、(Route 53 ARC を使用して) 現在アクティブなリージョンを返すように Route 53 を設定できます。

> Note
>
> クライアントは、ドメイン名の有効期間 (TTL) 設定で指定された期間にわたって、Route 53 からの応答に含まれている IP アドレスをキャッシュします。TTL を長くすると、すべてのクライアントが新しいエンドポイントを認識するまでの目標復旧時間 (RTO) が長くなります。フェイルオーバーを使用する場合は通常 60 秒です。すべてのソフトウェアが DNS TTL の有効期限を完全に順守しているわけではありません。

- 自分のリージョンへの書き込みモードでは、コンピューティングレイヤーのリクエストルーティングを一緒に使用する場合を除いて、Route 53 を避けるのが最善です。

Global Accelerator のリクエストルーティング

クライアントは、AWS Global Accelerator を使用して Route 53 の既知のドメイン名を検索します。ただし、クライアントはリージョンのエンドポイントに対応する IP アドレスを取得する代わりに、最も近い AWS エッジロケーションにルーティングするエニーキャスト静的 IP アドレスを受け取ります。このエッジロケーションを起点に、すべてのトラフィックはプライベート AWS ネットワークでルーティングされます。ルーティング先は、Global Accelerator 内で維持されるルーティングルールで選択された、リージョンの一部のエンドポイント (ロードバランサーや API ゲートウェイなど) です。Global Accelerator のリクエストルーティングは、Route 53 ルールに基づくルーティングと比較して、パブリックインターネットのトラフィック量を削減するため、レイテンシーが短縮されます。さらに、Global Accelerator はルーティングルールを変更するために DNS TTL の有効期限に依存しないため、ルーティングをより迅速に調整できます。


West
East
DynamoDB global table
DynamoDB global table
Local
read and write
Local
read
Redirected
write
Compute
Compute
Global Accelerator
network routing
and decision making
Global Accelerator
network routing
and decision making
DNS: write.example.com
DNS: read.example.com
DNS: write.example.com
DNS: read.example.com
Global Accelerator routes
on the Amazon network
based on config rules

- 任意のリージョンへの書き込みモード、またはバックエンドのコンピューティングレイヤーのリクエストルーティングと組み合わせることで、Global Accelerator はシームレスに動作します。クライアントは最も近いエッジロケーションに接続するため、どのリージョンがリクエストを受け取るか気にする必要はありません。
- 1 つのリージョンに書き込む場合、Global Accelerator のルーティングルールにより、現在アクティブなリージョンにリクエストを送信する必要があります。ヘルスチェックを使用すると、グローバルシステムによってアクティブなリージョンとは見なされていないリージョンの障害を人為的に報告できます。DNS と同様に、読み取りリクエストがどのリージョンからでも可能な場合は、代替の DNS ドメイン名を使用して読み取りリクエストをルーティングできます。
- 自分のリージョンへの書き込みモードでは、コンピューティングレイヤーのリクエストルーティングを一緒に使用する場合を除いて、Global Accelerator を避けるのが最善です。

グローバルテーブルを使用したリージョンからの退避

リージョンからの退避とは、リージョンから読み取り/書き込みアクティビティを移行するプロセスです。ほとんどの場合、これは書き込みアクティビティです。ときに読み取りアクティビティが含まれる場合もあります。

ライブリージョンからの退避

さまざまな理由で、ライブリージョンから退避することを決定する場合があります。退避は、1 つのリージョンへのフォローザサン書き込みモードを使用している場合など、通常のビジネス活動の一部である場合もあります。また、DynamoDB 外部のソフトウェアスタックの障害に対応して現在のアクティブなリージョンを変更するというビジネス上の決定や、リージョン内で通常よりも高いレイテンシーなどの一般的な問題が発生したことが原因で、退避を行う場合もあります。

任意のリージョンへの書き込みモードでは、ライブリージョンから簡単に退避できます。任意のルーティングシステムを介してトラフィックを代替リージョンにルーティングし、退避したリージョンで実行済みの書き込みオペレーションを通常どおりレプリケートできます。

1 つのリージョンへの書き込みモードと自分のリージョンへの書き込みモードでは、新しいアクティブなリージョンへの書き込みを開始する前に、現在のアクティブリージョンへのすべての書き込みが完全に記録されてストリーム処理され、グローバルに伝播されていることを確認する必要があります。これは、今後の書き込みが最新バージョンのデータに対して行われるようにするために必要です。

例えば、リージョン A がアクティブ、リージョン B がパッシブだとします (リージョン A をホームとするテーブル全体または項目のいずれかが対象)。退避を実行する一般的なメカニズムは、A へ

の書き込みオペレーションを一時停止し、そのオペレーションが B に完全に伝播されるまで待ち、アーキテクチャスタックを更新して B がアクティブであることを認識してから、B への書き込みオペレーションを再開することです。リージョン A がリージョン B にデータを完全にレプリケートしたことを確実に示すメトリクスはありません。リージョン A が正常であれば、リージョン A への書き込みオペレーションを一時停止し、ReplicationLatency メトリクスの最新の最大値の 10 倍を待てば、通常はレプリケーションが完了したことを十分に確認できます。リージョン A に異常があり、他の領域でのレイテンシーの増加も示している場合は、待機時間の倍数を大きくします。

オフラインリージョンからの退避

考慮すべき特別なケースがあります。リージョン A が予告なしに完全にオフラインになった場合はどうなるでしょうか。これは非常に可能性が低いことですが、用心に越したことはありません。これが発生した場合、リージョン A でまだ伝播されていない書き込みオペレーションはすべて保持され、リージョン A がオンラインに戻った後に伝播されます。書き込みオペレーションは失われませんが、その伝播は無期限に遅延します。

このイベントの処理方法は、アプリケーションの決定によります。ビジネスの継続性を確保するために、書き込みオペレーションを新しいプライマリリージョン B に移行することが必要になる場合があります。ただし、リージョン A の項目に対する書き込みオペレーションの伝播が保留されているときに、リージョン B でその項目が更新を受け取ると、最終書き込み者優先モデルに従ってその伝播は抑制されます。リージョン B での更新は、着信する書き込みリクエストを抑制する可能性があります。

任意のリージョンへの書き込みモードでは、リージョン A の項目が最終的にリージョン B に伝播されることを信頼し、リージョン A がオンラインに戻るまで項目が欠落する可能性を認識しながら、リージョン B で読み取りと書き込みを続行できます。可能な場合は、最近の書き込みトラフィックを (アップストリームのイベントソースを使用するなどして) 再生することを検討してください。これにより、欠落している可能性のある書き込みオペレーションのギャップを埋め、最終書き込み者優先の競合解決により、着信する書き込みオペレーションの最終的な伝播を抑制します。

他の書き込みモードでは、どの程度の作業を継続できるかを、少し時代遅れの方法で考慮する必要があります。リージョン A がオンラインに戻るまで、ReplicationLatency で追跡する書き込みオペレーションの一部の短い期間は失われます。ビジネスを継続できるでしょうか。一部のユースケースでは可能ですが、他のユースケースでは追加の緩和メカニズムがないと難しい場合があります。

例えば、リージョンに障害が発生した後でも、利用可能なクレジット残高を中断することなく維持する必要があるとします。残高を 2 つの異なる項目に分割し、1 つはリージョン A に、もう 1 つはリージョン B に置き、それぞれ利用可能な残高の半分から開始できます。この場合は、自分のリージョンへの書き込みモードを使用します。各リージョンで処理されたトランザクションの更新は、残

高のローカルコピーに対して書き込まれます。リージョン A が完全にオフラインになっても、リージョン B でトランザクション処理を続行でき、書き込みオペレーションはリージョン B に保持されている残高部分に制限されます。このように残高を分割すると、残高が少なくなった場合やクレジットの再調整が必要になった場合に複雑になりますが、保留中の書き込みオペレーションが不安定でも安全にビジネスを回復できる一例となります。

別の例として、ウェブフォームのデータをキャプチャするとします。オプティミスティック同時実行制御 (OCC) を使用してデータ項目にバージョンを割り当て、最新バージョンを非表示フィールドとしてウェブフォームに埋め込むことができます。送信するたびに、データベース内のバージョンがフォーム作成時のバージョンとまだ一致する場合にのみ、書き込みオペレーションが成功します。バージョンが一致しない場合、データベース内の現在のバージョンに基づいてウェブフォームを更新 (または慎重にマージ) することで、ユーザーは再度続行できます。OCC モデルは通常、別のクライアントがデータを上書きして新しいバージョンを生成するのを防ぎますが、フェイルオーバー中にクライアントが古いバージョンのデータに遭遇する可能性がある場合にも役立ちます。

タイムスタンプをバージョンとして使用しているとします。例えば、フォームを当初 12:00 にリージョン A に対して作成しましたが、(フェイルオーバー後に) リージョン B に書き込もうとして、データベースの最新バージョンが 11:59 であることに気付いたとします。このシナリオの場合、クライアントは 12:00 バージョンがリージョン B に伝播されるのを待ってからそのバージョンに書き込むか、11:59 バージョンに基づいて構築し、新しい 12:01 バージョンを作成することができます (書き込み後、リージョン A の回復後に着信したバージョンは抑制されます)。

最後の例として、金融サービス会社が顧客アカウントおよび金融取引に関するデータを DynamoDB データベースに保持しているとします。同社は、リージョン A が完全に停止した場合、アカウントに関連する書き込みアクティビティのすべてがリージョン B で完全に利用可能であることを確認するか、リージョン A がオンラインに戻るまでアカウントを部分的なものとして隔離したいと考えていました。すべての業務を一時停止するのではなく、伝播されていないトランザクションがあると判断したごく一部のアカウントに対してのみ業務を一時停止することにしました。これを実現するために、同社は 3 番目のリージョン (リージョン C と呼びます) を使用しました。リージョン A で書き込みオペレーションを処理する前に、保留中のオペレーション (アカウントの新規トランザクション数など) の簡潔な要約をリージョン C に配置しました。この要約は、リージョン B のビューが完全に最新であるかどうかを判断するのに十分でした。このアクションにより、リージョン C での書き込み時点から、リージョン A が書き込みオペレーションを受け入れてリージョン B がそれを受け取るまでの間、アカウントは事実上ロックされました。リージョン C のデータは、フェイルオーバープロセスの一部として以外は使用されませんでした。その後、リージョン B はリージョン C とデータを相互チェックして、古いアカウントがないかどうかを確認できました。これらのアカウントは、リージョン A の回復に伴って部分的なデータがリージョン B に伝播されるまで、隔離済みとしてマークされます。

リージョン C に障害が発生した場合は、代わりに新しいリージョン D をスピンアップして使用できます。リージョン C のデータはごく一時的なもので、数分後には処理中の書き込みオペレーションの最新の記録がリージョン D に反映されて十分に役立つようになります。リージョン B に障害が発生しても、リージョン A はリージョン C と協力して書き込みリクエストを引き続き受け入れることができます。同社は、より高いレイテンシーの書き込み (C に続けて A という 2 つのリージョンへの書き込み) をあえて受け入れ、アカウントの状態を簡潔に要約できるデータモデルを持つことができたことを喜んでいます。

グローバルテーブルのスループットキャパシティプランニング

リージョン間でトラフィックを移行する場合は、キャパシティに関する DynamoDB テーブルの設定を慎重に考慮する必要があります。

書き込みキャパシティの管理に関する考慮事項は以下のとおりです。

- グローバルテーブルは、オンデマンドモードにするか、自動スケーリングを有効にしてプロビジョニングする必要があります。
- 自動スケーリングでプロビジョニングした場合、書き込み設定 (最小使用率、最大使用率、およびターゲット使用率) はリージョン間でレプリケートされます。自動スケーリングの設定は同期されますが、実際にプロビジョニングされた書き込みキャパシティは、リージョン間で独立して変動する可能性があります。
- プロビジョニングされた書き込みキャパシティが異なる場合がある理由の 1 つは、TTL 機能によるものです。DynamoDB で TTL を有効にすると、項目の有効期限を秒単位の Unix エポック時間形式の値として示す属性名を指定できます。その後、DynamoDB は書き込みコストを発生させずに項目を削除できます。グローバルテーブルを使用すると、任意のリージョンで TTL を設定でき、その設定はグローバルテーブルに関連付けられている他のリージョンに自動的にレプリケートされます。項目が TTL ルールで削除の対象となった場合、削除はどのリージョンでも実行できます。削除オペレーションはソーステーブルの書き込みユニットを消費せずに実行されますが、レプリカテーブルは削除オペレーションのレプリケートされた書き込みを受け取るため、レプリケートされた書き込みユニットのコストが発生します。
- 自動スケーリングを使用する場合は、プロビジョニングされた最大書き込みキャパシティの設定が、すべての書き込みオペレーションとすべての潜在的な TTL 削除オペレーションを処理するのに十分な大きさであることを確認してください。自動スケーリングは、書き込み消費量に応じて各リージョンを調整します。オンデマンドテーブルには、プロビジョニングされた最大書き込みキャパシティの設定はありませんが、テーブルレベルの最大書き込みスループット制限により、オンデマンドテーブルが許可する最大持続書き込みキャパシティが指定されます。デフォルトの制限は 40,000 ですが、調整可能です。オンデマンドテーブルに必要となるすべての書き込みオペレー

ション (TTL 書き込みオペレーションを含む) を処理できるように、この値を十分に高く設定することをお勧めします。グローバルテーブルを設定する場合、この値は参加しているすべてのリージョンで同じでなければなりません。

読み取りキャパシティの管理に関する考慮事項は以下のとおりです。

- 読み取りキャパシティの管理に関する設定は、リージョンごとに読み取りパターンが異なることが想定されているため、リージョン間で異なることが許容されます。グローバルレプリカを初めてテーブルに追加すると、ソースリージョンのキャパシティが反映されます。作成後、読み取りキャパシティの設定は調整できますが、この調整した設定は反対側には転送されません。
- DynamoDB 自動スケーリングを使用するときは、プロビジョニングされた最大読み取りキャパシティの設定が、すべてのリージョンのすべての読み取りオペレーションを処理するのに十分な大きさであることを確認してください。標準的なオペレーション時には、読み取りキャパシティがリージョン全体に分散される可能性がありますが、フェイルオーバー時には、テーブルが読み取りワークロードの増大に自動的に対応できなければなりません。オンデマンドテーブルにはプロビジョニングされた最大読み取りキャパシティの設定はありませんが、テーブルレベルの最大読み取りスループット制限により、オンデマンドテーブルが許容する最大持続読み取りキャパシティが指定されます。デフォルトの制限は 40,000 ですが、調整可能です。すべての読み取りオペレーションをこの単一のリージョンにルーティングする場合、テーブルが必要とするすべての読み取りオペレーションを処理できるように、この値を十分に高く設定することをお勧めします。
- あるリージョンのテーブルが通常は読み取りトラフィックを受信しないが、フェイルオーバー後に大量の読み取りトラフィックを吸収しなければならない場合は、テーブルのプロビジョニングされた読み取りキャパシティを増やし、テーブルの更新が完了するのを待ってから、テーブルを再度プロビジョニングできます。テーブルをプロビジョニングモードのままにすることも、オンデマンドモードに切り替えることもできます。これにより、テーブルが事前にウォームアップされ、より高いレベルの読み取りトラフィックを受け入れることができます。

Route 53 ARC には準備状況チェック機能があり、Route 53 を使用してリクエストをルーティングするかどうかにかかわらず、DynamoDB リージョン間でテーブル設定とアカウントクォータが同じであることを確認するのに役立ちます。準備状況チェックは、アカウントレベルのクォータを調整して相互に一致させる場合にも役立ちます。

グローバルテーブルの準備チェックリストとよくある質問

グローバルテーブルをデプロイするときの決定事項とタスクには、次のチェックリストを使用してください。

- グローバルテーブルに参加するリージョンの数と各リージョンを決定します。
- アプリケーションの書き込みモードを決定します。詳細については、「グローバルテーブルによる書き込みモード」を参照してください。
- 書き込みモードに基づいて「グローバルテーブルを使用したリクエストルーティング」戦略を計画します。
- 書き込みモードとルーティング戦略に基づいて、

リージョンからの退避とは、リージョンから読み取り/書き込みアクティビティを移行するプロセスです。ほとんどの場合、これは書き込みアクティビティです。ときに読み取りアクティビティが含まれる場合もあります。

ライブリージョンからの退避

さまざまな理由で、ライブリージョンから退避することを決定する場合があります。退避は、1つのリージョンへのフォローザサン書き込みモードを使用している場合など、通常のビジネス活動の一部である場合もあります。また、DynamoDB外部のソフトウェアスタックの障害に対応して現在のアクティブなリージョンを変更するというビジネス上の決定や、リージョン内で通常よりも高いレイテンシーなどの一般的な問題が発生したことが原因で、退避を行う場合もあります。

任意のリージョンへの書き込みモードでは、ライブリージョンから簡単に退避できます。任意のルーティングシステムを介してトラフィックを代替リージョンにルーティングし、退避したリージョンで実行済みの書き込みオペレーションを通常どおりレプリケートできます。

1つのリージョンへの書き込みモードと自分のリージョンへの書き込みモードでは、新しいアクティブなリージョンへの書き込みを開始する前に、現在のアクティブリージョンへのすべての書き込みが完全に記録されてストリーム処理され、グローバルに伝播されていることを確認する必要があります。これは、今後の書き込みが最新バージョンのデータに対して行われるようにするために必要です。

例えば、リージョンAがアクティブ、リージョンBがパッシブだとします(リージョンAをホームとするテーブル全体または項目のいずれかが対象)。退避を実行する一般的なメカニズムは、Aへの書き込みオペレーションを一時停止し、そのオペレーションがBに完全に伝播されるまで待ち、アーキテクチャスタックを更新してBがアクティブであることを認識してから、Bへの書き込みオペレーションを再開することです。リージョンAがリージョンBにデータを完全にレプリケートしたことを確実に示すメトリクスはありません。リージョンAが正常であれば、リージョンAへの書き込みオペレーションを一時停止し、`ReplicationLatency`メトリ

クスの最新の最大値の 10 倍を待てば、通常はレプリケーションが完了したことを十分に確認できます。リージョン A に異常があり、他の領域でのレイテンシーの増加も示している場合は、待機時間の倍数を大きくします。

オフラインリージョンからの退避

考慮すべき特別なケースがあります。リージョン A が予告なしに完全にオフラインになった場合はどうなるでしょうか。これは非常に可能性が低いことですが、用心に越したことはありません。これが発生した場合、リージョン A でまだ伝播されていない書き込みオペレーションはすべて保持され、リージョン A がオンラインに戻った後に伝播されます。書き込みオペレーションは失われませんが、その伝播は無期限に遅延します。

このイベントの処理方法は、アプリケーションの決定によります。ビジネスの継続性を確保するために、書き込みオペレーションを新しいプライマリリージョン B に移行することが必要になる場合があります。ただし、リージョン A の項目に対する書き込みオペレーションの伝播が保留されているときに、リージョン B でその項目が更新を受け取ると、最終書き込み者優先モデルに従ってその伝播は抑制されます。リージョン B での更新は、着信する書き込みリクエストを抑制する可能性があります。

任意のリージョンへの書き込みモードでは、リージョン A の項目が最終的にリージョン B に伝播されることを信頼し、リージョン A がオンラインに戻るまで項目が欠落する可能性を認識しながら、リージョン B で読み取りと書き込みを続行できます。可能な場合は、最近の書き込みトラフィックを (アップストリームのイベントソースを使用するなどして) 再生することを検討してください。これにより、欠落している可能性のある書き込みオペレーションのギャップを埋め、最終書き込み者優先の競合解決により、着信する書き込みオペレーションの最終的な伝播を抑制します。

他の書き込みモードでは、どの程度の作業を継続できるかを、少し時代遅れの方法で考慮する必要があります。リージョン A がオンラインに戻るまで、`ReplicationLatency` で追跡する書き込みオペレーションの一部の短い期間は失われます。ビジネスを継続できるでしょうか。一部のユースケースでは可能ですが、他のユースケースでは追加の緩和メカニズムがないと難しい場合があります。

例えば、リージョンに障害が発生した後でも、利用可能なクレジット残高を中断することなく維持する必要があるとします。残高を 2 つの異なる項目に分割し、1 つはリージョン A に、もう 1 つはリージョン B に置き、それぞれ利用可能な残高の半分から開始できます。この場合は、自分のリージョンへの書き込みモードを使用します。各リージョンで処理されたトランザクションの更新は、残高のローカルコピーに対して書き込まれます。リージョン A が完全にオフライ

ンになっても、リージョン B でトランザクション処理を続行でき、書き込みオペレーションはリージョン B に保持されている残高部分に制限されます。このように残高を分割すると、残高が少なくなった場合やクレジットの再調整が必要になった場合に複雑になりますが、保留中の書き込みオペレーションが不安定でも安全にビジネスを回復できる一例となります。

別の例として、ウェブフォームのデータをキャプチャするとします。オプティミスティック同時実行制御 (OCC) を使用してデータ項目にバージョンを割り当て、最新バージョンを非表示フィールドとしてウェブフォームに埋め込むことができます。送信するたびに、データベース内のバージョンがフォーム作成時のバージョンとまだ一致する場合にのみ、書き込みオペレーションが成功します。バージョンが一致しない場合、データベース内の現在のバージョンに基づいてウェブフォームを更新 (または慎重にマージ) することで、ユーザーは再度続行できます。OCC モデルは通常、別のクライアントがデータを上書きして新しいバージョンを生成するのを防ぎますが、フェイルオーバー中にクライアントが古いバージョンのデータに遭遇する可能性がある場合にも役立ちます。

タイムスタンプをバージョンとして使用しているとします。例えば、フォームを当初 12:00 にリージョン A に対して作成しましたが、(フェイルオーバー後に) リージョン B に書き込もうとして、データベースの最新バージョンが 11:59 であることに気付いたとします。このシナリオの場合、クライアントは 12:00 バージョンがリージョン B に伝播されるのを待ってからそのバージョンに書き込むか、11:59 バージョンに基づいて構築し、新しい 12:01 バージョンを作成することができます (書き込み後、リージョン A の回復後に着信したバージョンは抑制されます)。

最後の例として、金融サービス会社が顧客アカウントおよび金融取引に関するデータを DynamoDB データベースに保持しているとします。同社は、リージョン A が完全に停止した場合、アカウントに関連する書き込みアクティビティのすべてがリージョン B で完全に利用可能であることを確認するか、リージョン A がオンラインに戻るまでアカウントを部分的なものとして隔離したいと考えていました。すべての業務を一時停止するのではなく、伝播されていないトランザクションがあると判断したごく一部のアカウントに対してのみ業務を一時停止することにしました。これを実現するために、同社は 3 番目のリージョン (リージョン C と呼びます) を使用しました。リージョン A で書き込みオペレーションを処理する前に、保留中のオペレーション (アカウントの新規トランザクション数など) の簡潔な要約をリージョン C に配置しました。この要約は、リージョン B のビューが完全に最新であるかどうかを判断するのに十分でした。このアクションにより、リージョン C での書き込み時点から、リージョン A が書き込みオペレーションを受け入れてリージョン B がそれを受け取るまでの間、アカウントは事実上ロックされました。リージョン C のデータは、フェイルオーバープロセスの一部として以外は使用されませんでした。その後、リージョン B はリージョン C とデータを相互チェックして、古い

アカウントがないかどうかを確認できました。これらのアカウントは、リージョン A の回復に伴って部分的なデータがリージョン B に伝播されるまで、隔離済みとしてマークされます。

リージョン C に障害が発生した場合は、代わりに新しいリージョン D をスピンアップして使用できます。リージョン C のデータはごく一時的なもので、数分後には処理中の書き込みオペレーションの最新の記録がリージョン D に反映されて十分に役立つようになります。リージョン B に障害が発生しても、リージョン A はリージョン C と協力して書き込みリクエストを引き続き受け入れることができます。同社は、より高いレイテンシーの書き込み (C に続けて A という 2 つのリージョンへの書き込み) をあえて受け入れ、アカウントの状態を簡潔に要約できるデータモデルを持つことができたことを喜んでいます。

退避計画を定義します。

- リージョンごとのヘルス、レイテンシー、エラーに関するメトリクスをキャプチャします。DynamoDB メトリクスのリストについては、AWS ブログ記事「運用状況を把握するための Amazon DynamoDB のモニタリング」で確認すべきメトリクスのリストを参照してください。また、合成 canary (炭鉱のカナリアにちなんで名付けられた、障害を検出するための人工的なリクエスト) の使用や、顧客トラフィックのライブ観察も必要です。すべての問題が DynamoDB メトリクスに反映されるわけではありません。
- `ReplicationLatency` の継続的な増加に対するアラームを設定します。この増加は、グローバルテーブルの書き込み設定がリージョンごとに異なるという偶発的な設定ミスを示している可能性があり、その結果、レプリケートされたリクエストが失敗し、レイテンシーが高くなります。また、リージョンに混乱が生じていることを示している可能性もあります。良い例としては、最近の平均が 180,000 ミリ秒を超えた場合にアラートを生成することが挙げられます。また、`ReplicationLatency` が 0 に下がり、レプリケーションが停止していることを示す場合もあります。
- 各グローバルテーブルに十分な最大読み取り/書き込み設定を割り当てます。
- リージョンから退避する理由を事前に特定します。決定に人間の判断が関与する場合は、すべての考慮事項を文書化します。この作業は、必要に迫られて行うのではなく、事前に慎重に行う必要があります。
- リージョンから退避する際に実行する必要があるすべてのアクションのランブックを用意しておきます。通常、グローバルテーブルに関係する作業はほとんどありませんが、スタックの残りの部分を移動するのは複雑な作業になる場合があります。

> **Note**
>
> リージョンに障害が発生すると、一部のコントロールプレーンオペレーションのパフォーマンスが低下する可能性があるため、コントロールプレーンオペレーションには依存せず、データプレーンオペレーションにのみ依存するのがベストプラクティスです。

詳細については、AWS ブログ記事「Amazon DynamoDB グローバルテーブルを使用した回復性の高いアプリケーションの構築: パート 4」を参照してください。

- リージョンの退避を含め、ランブックのあらゆる側面を定期的にテストします。テストしないと、ランブックの信頼性が低下します。
- Resilience Hub を使用して、アプリケーション全体 (グローバルテーブルを含む) のレジリエンスを評価することを検討します。ダッシュボードを通じて、アプリケーションポートフォリオ全体のレジリエンスステータスを包括的に確認できます。
- Route 53 ARC の準備状況チェックを使用して、アプリケーションの現在の設定を評価し、ベストプラクティスから逸脱していないか追跡することを検討します。
- Route 53 または Global Accelerator で使用するヘルスチェックを作成する場合、DynamoDB エンドポイントが稼働していることを ping するだけでは不十分です。IAM 設定エラー、コードデプロイの問題、DynamoDB 外部のスタック障害、平均を超える読み取り/書き込みレイテンシーなど、多くの障害モードは含まれていません。データベースフロー全体を実行する一連の呼び出しを実行するのが最善です。

グローバルテーブルのデプロイに関するよくある質問 (FAQ)

DynamoDB グローバルテーブルの使用全般に役立つ原則にはどのようなものがありますか?

DynamoDB グローバルテーブルにはコントロールノブがほとんどありませんが、それでもいくつかの考慮事項が必要です。書き込みモード、ルーティングモデル、および退避プロセスを決定する必要があります。すべてのリージョンにわたってアプリケーションをインストルメントし、グローバルヘルスを維持するためのルーティングの調整や退避に備える必要があります。これにより、読み取りと書き込みが低レイテンシーで、サービスレベルアグリーメントが 99.999% のグローバルに分散されたデータセットが実現します。

グローバルテーブルの料金はいくらですか?

従来の DynamoDB テーブルへの書き込みは、書き込みキャパシティユニット (WCU、プロビジョニングされたテーブルの場合) または書き込みリクエストユニット (WRU、オンデマンドテーブルの場合) で料金が設定されます。5 KB の項目を書き込むと、5 ユニットの料金が発生します。グローバルテーブルへの書き込みは、レプリケートされた書き込みキャパシティユニット (rWCU、プロビジョニングされたテーブルの場合) またはレプリケートされた書き込みリクエストユニット (rWRU、オンデマンドテーブルの場合) で料金が設定されます。

rWCU と rWRU には、レプリケーションの管理に必要なストリーミングインフラストラクチャのコストが含まれます。そのため、WCU や WRU よりも料金が 50% 高くなります。リージョン間のデータ転送料金がかかります。

レプリケートされた書き込みユニットの料金は、項目が直接書き込まれたか、レプリケートされて書き込まれた、すべてのリージョンで発生します。

グローバルセカンダリインデックス (GSI) への書き込みはローカル書き込みと見なされ、通常の書き込みユニットを使用します。

現在、rWCU に利用できるリザーブドキャパシティはありません。リザーブドキャパシティの購入は、GSI が書き込みユニットを消費するテーブルでは依然として有益な場合があります。

グローバルテーブルに新しいリージョンを追加するときの初期ブートストラップには、復元されたデータの GB あたりのリストアと同様の料金に加えて、リージョン間のデータ転送料金がかかります。

グローバルテーブルはどのリージョンをサポートしていますか?

グローバルテーブルバージョン 2019.11.21 (現行) は、ほとんどのリージョンで使用できます。レプリカを追加すると、DynamoDB コンソールの [リージョン] ドロップダウンリストに最新のリストが表示されます。

GSI はグローバルテーブルでどのように処理されますか?

グローバルテーブルバージョン 2019.11.21 (現行) では、あるリージョンで GSI を作成すると、他の参加リージョンでも自動的に作成され、自動的にバックフィルされます。

グローバルテーブルのレプリケーションを停止するにはどうしたらいいですか?

レプリカテーブルは、他のテーブルを削除するのと同じ方法で削除できます。グローバルテーブルを削除すると、そのリージョンへのレプリケーションが停止し、そのリージョンに保持されているテーブルのコピーが削除されます。ただし、テーブルのコピーを独立したエンティティとして保持している間は、レプリケーションを停止または一時停止することはできません。

DynamoDB Streams はグローバルテーブルとどのようにやり取りするのですか?

各グローバルテーブルは、書き込み元に関係なく、すべての書き込みに基づいて独立したストリームを生成します。DynamoDB ストリームを 1 つのリージョンで使用するか、すべてのリージョンで (個別に) 使用するかを選択できます。レプリケートされた書き込みオペレーションではなく、ローカル書き込みオペレーションを処理する場合は、各項目に独自のリージョン属性を追加して、書き込みリージョンを識別できます。次に、Lambda イベントフィルターを使用して、ローカルリージョンでの書き込みオペレーションに対してのみ Lambda 関数を呼び出すことができます。これは挿入および更新オペレーションには役立ちますが、残念ながら削除オペレーションには役立ちません。

グローバルテーブルはトランザクションをどのように処理するのですか?

トランザクションオペレーションは、書き込みオペレーションが最初に発生したリージョン内でのみ、不可分性、一貫性、分離性、および耐久性 (ACID) を保証します。グローバルテーブルのリージョン間では、トランザクションはサポートされていません。例えば、米国東部 (オハイオ) および米国西部 (オレゴン) リージョンにレプリカを持つグローバルテーブルがあり、米国東部 (オハイオ) リージョンで `TransactWriteItems` オペレーションを実行すると、変更がレプリケートされたときに米国西部 (オレゴン) では部分的に完了したトランザクションが観察されることがあります。変更は、ソースリージョンでコミットされた後でのみ、他のリージョンにレプリケートされます。

グローバルテーブルは DynamoDB Accelerator キャッシュ (DAX) とどのようにやり取りするのですか?

グローバルテーブルは DynamoDB を直接更新することで DAX をバイパスするため、DAX はそれが古いデータを保持していることを認識しません。DAX キャッシュは、キャッシュの TTL が期限切れになったときにのみ更新されます。

テーブルのタグは伝播されますか?

いいえ、タグは自動的には伝播されません。

すべてのリージョンのテーブルをバックアップすべきですか? 1 つのリージョンのテーブルのみをバックアップすべきですか?

答えはバックアップの目的によって異なります。データの耐久性を確保したい場合、DynamoDB には既に保護手段が用意されています。このサービスにより、耐久性が保証されます。履歴記録のスナップショットを保持する場合 (規制要件を満たすためなど)、1 つのリージョンでバックアップすれば十分です。AWS Backup を使用してバックアップを別のリージョンにコピーできます。誤って削除または変更したデータを復元する場合は、1 つのリージョンで DynamoDB ポイントインタイムリカバリ (PITR) を使用します。

AWS CloudFormation を使用してグローバルテーブルをデプロイするにはどうすればいいですか?

CloudFormation は、DynamoDB テーブルとグローバルテーブルを 2 つの独立したリソース (`AWS::DynamoDB::Table` と `AWS::DynamoDB::GlobalTable`) として表します。1 つの方法としては、`GlobalTable` コンストラクトを使用して、グローバルになる可能性のあるすべてのテーブルを作成します。その後、最初はスタンドアロンテーブルとして保持し、必要に応じて後でリージョンを追加できます。

CloudFormation では、レプリカの数に関係なく、各グローバルテーブルは単一リージョン内の単一スタックによって制御されます。テンプレートをデプロイすると、CloudFormation はすべてのレプリカを単一スタックのオペレーションの一部として作成および更新します。同じ `AWS::DynamoDB::GlobalTable` リソースを複数のリージョンにデプロイしないでください。これは、エラーとなるため、サポートされていません アプリケーションテンプレートを複数のリージョンにデプロイする場合は、条件を使用して単一リージョンに `AWS::DynamoDB::GlobalTable` リソースを作成できます。または、アプリケーションスタックとは別のスタック内に `AWS::DynamoDB::GlobalTable` リソースを定義し、それが単一リージョンにのみデプロイされるようにします。

通常のテーブルがあり、それを CloudFormationtion で管理したままグローバルテーブルに変換する場合は、削除ポリシーを [保持] に設定して、テーブルをスタックから削除し、コンソールでテーブルをグローバルテーブルに変換します。次に、グローバルテーブルを新しいリソースとしてスタックにインポートします。

クロスアカウントレプリケーションは現在サポートされていません。

DynamoDB でコントロールプレーンを管理するためのベストプラクティス

> **Note**
>
> DynamoDB では、1 秒あたり 2,500 リクエストというコントロールプレーンのスロットル制限を導入しており、再試行のオプションもあります。詳細については、以下を参照してください。

DynamoDB コントロールプレーンオペレーションでは、DynamoDB テーブルだけでなく、インデックスなどのテーブルに依存するオブジェクトも管理できます。これらのオペレーションの詳細については、コントロールプレーン を参照してください。

状況によっては、アクションを実行し、コントロールプレーンの呼び出しによって返されたデータをビジネスロジックの一部として使用する必要がある場合があります。例えば、`DescribeTable` によって返される `ProvisionedThroughput` の値を知っておく必要がある場合があります。このような場合は、次のベストプラクティスに従ってください。

- DynamoDB コントロールプレーンに過度にクエリを実行しないでください。
- 同じコード内でコントロールプレーンの呼び出しとデータプレーンの呼び出しを混在させないでください。
- コントロールプレーンのリクエストでスロットリングを処理し、バックオフして再試行します。
- 1 つのクライアントから特定のリソースを呼び出して変更を追跡します。
- 同じテーブルのデータを短い間隔で複数回取得する代わりに、データをキャッシュして処理します。

AWS の請求および使用状況レポートを理解するためのベストプラクティス

このドキュメントでは、DynamoDB に関連する `UsageType` 料金の請求コードについて説明します。

AWS は、使用したサービスのデータを含む請求および使用状況レポート (CUR) を提供します。AWS Cost and Usage Report を使用して、請求レポートを CSV 形式で Amazon S3 に公開できます。CUR を設定する際は、時間単位、日単位、または月単位で期間を分割したり、使用量をリソース ID ごとに分割するかどうかを選択できます。CUR の生成の詳細については、「Creating Cost and Usage Reports」を参照してください。

CSV エクスポートでは、各行に関連する属性が一覧表示されます。レポートに含まれる属性の例を以下に示します。

- lineitem/UsageStartDate: 明細項目の開始日時 (UTC、その時刻を含む)。
- lineitem/UsageEndDate: 対応する明細項目の終了日時 (UTC、その時刻は含まない)。
- lineitem/ProductCode: DynamoDB では「AmazonDynamoDB」。
- lineitem/UsageType: このドキュメントに列挙されている使用タイプの説明コード。
- lineitem/Operation: 料金が発生したオペレーション名などの請求項目の情報 (オプション)。
- lineitem/ResourceId: 使用されたリソースの識別子。CUR にリソース ID 別の内訳が含まれている場合に表示されます。

- lineitem/UsageAmount: 指定した期間に発生した使用量。
- lineitem/UnblendedCost: この使用量の料金。
- lineitem/LineItemDescription: 各明細項目の説明。

CUR データディクショナリの詳細については、「Cost and Usage Report (CUR) 2.0」を参照してください。正確な名前はコンテキストによって異なることに注意してください。

`UsageType` は、`ReadCapacityUnit-Hrs`、`USW2-ReadRequestUnits`、`EU-WriteCapacityUnit-Hrs`、`USE1-TimedPITRStorage-ByteHrs` などの値を持つ文字列です。各使用タイプは、オプションのリージョンプレフィックスで始まります。プレフィックスがない場合、us-east-1 リージョンであることを示します。以下の表では、簡易的な請求リージョンコードを従来のリージョンコードとリージョン名にマッピングしています。

例えば、`USW2-ReadRequestUnits` という使用は、us-west-2 で消費される読み取りリクエストユニットを示します。

請求リージョンコード	リージョンコード	リージョン名
AFS1	af-south-1	アフリカ (ケープタウン)
APE1	ap-east-1	アジアパシフィック (香港)
APN1	ap-northeast-1	アジアパシフィック (東京)
APN2	ap-northeast-2	アジアパシフィック (ソウル)
APN3	ap-northeast-3	アジアパシフィック (大阪)
APS1	ap-south-1	アジアパシフィック (ムンバイ)
APS2	ap-south-2	アジアパシフィック (ハイデラバード)
APS3	ap-southeast-1	アジアパシフィック (シンガポール)
APS4	ap-southeast-2	アジアパシフィック (シドニー)

請求リージョンコード	リージョンコード	リージョン名
APS5	ap-southeast-3	アジアパシフィック (ジャカルタ)
APS6	ap-southeast-4	アジアパシフィック (メルボルン)
CAN1	ca-central-1	カナダ (中部)
EU	eu-central-1	欧州 (フランクフルト)
EUC1	eu-central-2	欧州 (チューリッヒ)
EUN1	eu-north-1	欧州 (ストックホルム)
EUS1	eu-south-1	欧州 (ミラノ)
EUS2	eu-south-2	欧州 (スペイン)
EUW1	eu-west-1	欧州 (アイルランド)
EUW2	eu-west-2	欧州 (ロンドン)
EUW3	eu-west-3	欧州 (パリ)
ILC1	il-central-1	イスラエル (テルアビブ)
MEC1	me-central-1	中東 (アラブ首長国連邦)
MES1	me-south-1	中東 (バーレーン)
SAE1	sa-east-1	南米（サンパウロ）
USE1 (デフォルト)	us-east-1	米国東部 (バージニア北部)
USE2	us-east-2	米国東部（オハイオ）
UGE1	us-gov-east-1	米国東部政府
UGW1	us-gov-west-1	米国西部政府

請求リージョンコード	リージョンコード	リージョン名
USW1	us-west-1	米国西部 (北カリフォルニア)
USW2	us-west-2	米国西部 (オレゴン)

以下のセクションでは、REG-UsageType パターンを使用して DynamoDB の料金を表示します。REG は使用が発生したリージョンを示し、usageType は請求タイプのコードを示します。例えば、CSV ファイルに USW1- ReadCapacityUnit-Hrs という明細項目がある場合、その項目は US-west-1 でプロビジョニングされた読み込みキャパシティの使用量が発生したことを示します。この場合、リストには「REG-ReadCapacityUnit-Hrs」と表示されます。

トピック

- スループット容量
- Streams
- [Storage (ストレージ)]
- バックアップと復元
- データ転送
- CloudWatch Contributor Insights
- DynamoDB Accelerator (DAX)

スループット容量

プロビジョニングされたキャパシティの読み取りと書き込み

DynamoDB テーブルをプロビジョンドキャパシティモードで作成する場合、アプリケーションに必要と想定される読み取りキャパシティを指定します。使用タイプはテーブルクラス (Standard または Standard-Infrequent Access) によって異なります。1 秒あたりの使用率に基づいて読み取りと書き込みをプロビジョニングしますが、料金はプロビジョニングされた容量に基づいて時間単位で課金されます。

UsageType	単位	詳細度	説明
REG-ReadCapacityUnit-Hrs	RCU 時	時	Standard テーブルクラスを使用するプロビジョンドキャパシティモードでの読み取りに対して発生する料金。
REG-IA-ReadCapacityUnit-Hrs	RCU 時	時	Standard-IA テーブルクラスを使用するプロビジョンドキャパシティモードでの読み取りに対して発生する料金。
REG-WriteCapacityUnit-Hrs	WCU 時	時	Standard テーブルクラスを使用するプロビジョンドキャパシティモードでの書き込みに対して発生する料金。
REG-IA-WriteCapacityUnit-Hrs	WCU 時	時	Standard-IA テーブルクラスを使用するプロビジョンドキャパシティモードでの書き込みに対して発生する料金。

リザーブドキャパシティの読み取りと書き込み

リザーブドキャパシティでは、1 回限りの前払い料金を支払い、期間中、プロビジョニングされた最小使用レベルを支払う契約を結びます。リザーブドキャパシティは、割引された時間単位の料金で請求されます。リザーブドキャパシティを超えてプロビジョニングしたキャパシティには、標準のプロビジョンされたキャパシティの料金が請求されます。リザーブドキャパシティは、Standard テーブルクラスを使用する DynamoDB テーブルの単一リージョンのプロビジョニングされた読み取りおよ

び書き込みキャパシティユニット (RCU および WCU) で使用できます。1 年および 3 年のリザーブドキャパシティはどちらも同じ SKU を使用して請求されます。

UsageType	単位	詳細度	説明
REG-HeavyUsage:dynamodb.read	RCU 時	前払い、その後月払い	リザーブドキャパシティの読み取り料金: 1 回限りの前払い料金と、割引されたコミット済みの RCU 時をすべてカバーする月額料金が毎月請求されます。対応するゼロコスト REG-ReadCapacityUnit-Hrs 明細項目が表示されます。
REG-HeavyUsage:dynamodb.write	WCU 時	前払い、その後月払い	リザーブドキャパシティの書き込み料金: 1 回限りの前払い料金と、割引されたコミット済みの WCU 時をすべてカバーする月額料金が毎月請求されます。対応するゼロコスト REG-WriteCapacityUnit-Hrs 明細項目が表示されます。

オンデマンドキャパシティの読み取りと書き込み

DynamoDB テーブルをオンデマンドキャパシティモードで作成する場合、アプリケーションが実行する読み取りと書き込みに対してのみ料金が発生します。読み取りリクエストと書き込みリクエストの料金は、テーブルクラスによって異なります。

UsageType	単位	詳細度	説明
REG-ReadRequestUnits	RRU	単位	Standard テーブルクラスを使用するオンデマンドキャパシティモードでの読み取りに対して発生する料金。
REG-IA-ReadRequestUnits	RRU	単位	Standard-IA テーブルクラスを使用するオンデマンドキャパシティモードでの読み取りに対して発生する料金。
REG-WriteRequestUnits	WRU	単位	Standard テーブルクラスを使用するオンデマンドキャパシティモードでの書き込みに対して発生する料金。
REG-IA-WriteRequestUnits	WRU	単位	Standard-IA テーブルクラスを使用するオンデマンドキャパシティモードでの書き込みに対して発生する料金。

グローバルテーブルの読み取りと書き込み

DynamoDB では、各レプリカテーブルで使用されているリソースに基づいてグローバルテーブルの使用料金が請求されます。プロビジョニングされたグローバルテーブルの場合、グローバルテーブルへの書き込みリクエストは標準 WCU ではなくレプリケート WCU (rWCU) で測定され、グローバルテーブルのグローバルセカンダリインデックスへの書き込みは WCU で測定されます。オンデマンドグローバルテーブルでは、書き込みリクエストは標準 WRU ではなくレプリケート WRU (rWRU)

で測定されます。レプリケーションに使用される rWCU の数または rWRU の数は、使用しているグローバルテーブルのバージョンによって異なります。料金は、テーブルクラスによって異なります。

グローバルセカンダリインデックス (GSI) への書き込みは、標準書き込み単位 (WCU と WRU) を使用して請求されます。読み取りリクエストとデータストレージは、単一リージョンのテーブルと同様に請求されます。

テーブルレプリカを追加して新しいリージョンでグローバルテーブルを作成または拡張した場合、DynamoDB では、復元されたデータ 1 GB ごとに、追加されたリージョンのテーブル復元に対して料金が請求されます。復元されたデータは REG-RestoreDataSize-Bytes として請求されます。詳細については、「DynamoDB のオンデマンドバックアップおよび復元の使用」を参照してください。クロスリージョンレプリケーションや、データを含むテーブルへのレプリカの追加にも、データ転送の料金が発生します。

DynamoDB グローバルテーブルでオンデマンドキャパシティモードを選択すると、各レプリカテーブルでアプリケーションが使用するリソースに対してのみ請求されます。

UsageType	単位	詳細度	説明
REG-ReplWriteCapacityUnit-Hrs	rWCU 時	時	グローバルテーブル、プロビジョンド、Standard テーブルクラス。
REG-IA-ReplWriteCapacityUnit-Hrs	rWCU 時	時	グローバルテーブル、プロビジョンド、Standard-IA テーブルクラス。
REG-ReplWriteRequestUnits	rWRU	単位	グローバルテーブル、オンデマンド、Standard テーブルクラス。
REG-IA-ReplWriteRequestUnits	rWRU	単位	グローバルテーブル、オンデマンド、Standard- IA テーブルクラス。

Streams

DynamoDB には、DynamoDB Streams と Kinesis という 2 つのストリーミングテクノロジーがあります。これらは、それぞれ料金が異なります。

DynamoDB Streams では、読み取りリクエストユニットでのデータ読み取りに対して料金が発生します。各 `GetRecords` API コールは、ストリーム読み取りリクエストとして請求されます。DynamoDB トリガーの一部として `GetRecords` から、またはレプリケーションの一部として DynamoDB グローバルテーブルから呼び出される AWS Lambda API コールに対しては、料金は発生しません。

UsageType	単位	詳細度	説明
REG-Streams-RequestsCount	カウント	単位	DynamoDB Streams の読み取りリクエストユニット。

Amazon Kinesis Data Streams では、データキャプチャユニットに対して料金が発生します。DynamoDB では、書き込みごとに 1 つの変更データキャプチャユニットが請求されます (最大 1 KB)。1 KB を超えるアイテムには、追加の変更データキャプチャユニットが必要です。テーブルのスループットキャパシティの管理なしに、アプリケーションが実行した書き込みに対してのみ料金が発生します。

UsageType	単位	詳細度	説明
REG-ChangeDataCaptureUnits-Kinesis	CDC ユニット	単位	Kinesis Data Streams の変更データキャプチャユニット。

[Storage (ストレージ)]

DynamoDB は、データの RAW バイトサイズと有効化した特徴量に応じて、項目あたりのストレージオーバーヘッドを加算して、請求対象データのサイズを測定します。

Note

DescribeTable はアイテムごとのストレージオーバーヘッドを含まないため、CUR のストレージ使用量は DescribeTable 使用時のストレージ使用量と比較して高くなります。

ストレージは 1 時間単位で計算されますが、1 時間あたりの料金の平均から計算され、月単位で課金されます。

UsageType ストレージは ByteHrs をサフィックスとして使用しますが、CUR のストレージ使用量は GB 単位で測定され、料金は GB 月単位です。

UsageType	単位	詳細度	説明
REG-TimedStorage-ByteHrs	GB	月	Standard テーブルクラスのテーブルで、DynamoDB テーブルとインデックスが使用するストレージの容量。
REG-IA-TimedStorage- ByteHrs	GB	月	Standard-IA テーブルクラスのテーブルで、DynamoDB テーブルとインデックスが使用するストレージの容量。

バックアップと復元

DynamoDB には、ポイントインタイムリカバリ (PITR) バックアップとオンデマンドバックアップの 2 種類のバックアップがあります。ユーザーは、これらのバックアップから DynamoDB テーブルに復元することもできます。以下の料金は、バックアップと復元の両方を対象としています。

バックアップストレージの料金は各月の初日に発生し、バックアップが追加または削除されるたびに調整されます。詳細については、「Understanding Amazon DynamoDB On-demand Backups and Billing」のブログ記事を参照してください。

UsageType	単位	詳細度	説明
REG-TimedBackupStorage-ByteHrs	GB	月	DynamoDB テーブルとローカルセカンダリインデックスのオンデマンドバックアップで消費されるストレージ。
TimedPITRStorage-ByteHrs	GB	月	ポイントインタイムリカバリ (PITR) バックアップで使用されるストレージ。PITR が有効である間、DynamoDB では、PITR が有効化されているテーブルサイズの 1 か月間の継続的なモニタリングに基づいてバックアップ料金が決定されます。
REG-RestoreDataSize-Bytes	GB	サイズ	DynamoDB バックアップから復元されたデータ (テーブルデータ、ローカルセカンダリインデックス、グローバルセカンダリインデックスを含む) の合計サイズ (GB 単位)。

AWS Backup

AWS Backup はフルマネージド型のバックアップサービスで、クラウドおよびオンプレミスの AWS サービス間でデータのバックアップを簡単に一元化および自動化できます。AWS Backup は、

ストレージ (ウォームストレージまたはコールドストレージ)、復元作業、およびリージョン間のデータ転送に対して請求されます。以下の UsageType 料金は、「AmazonDynamoDB」ではなく「AWSBackup」ProductCode の下に表示されます。

UsageType	単位	詳細度	説明
REG-WarmStorage-ByteHrs-DynamoDB	GB	月	1 か月を通して AWS Backup で管理される DynamoDB バックアップによって使用されたストレージ (GB-月単位)。
REG-CrossRegion-WarmBytes-DynamoDB	GB	サイズ	同一アカウントの異なる AWS リージョン、または異なる AWS アカウントに転送されたデータ。クロスリージョン転送の料金は、あるリージョンから別のリージョンにバックアップをコピーする際に発生します。料金は常にデータの転送元のアカウントに請求されます。
REG-Restore-WarmBytes-DynamoDB	GB	サイズ	ウォームストレージから復元されたデータの合計サイズ。GB 単位で測定されます。
REG-ColdStorage-ByteHrs-DynamoDB	GB	月	1 か月を通して AWS Backup で管理される DynamoDB バックアップによって使用

UsageType	単位	詳細度	説明
			されたコールドストレージ (GB-月単位)。
REG-Restore-ColdBytes-DynamoDB	GB	月	コールドストレージから復元されたデータの合計サイズ。GB 単位で測定されます。

エクスポートおよびインポート

DynamoDB から Amazon S3 にデータをエクスポートしたり、Amazon S3 から新しい DynamoDB テーブルにデータをインポートしたりすることができます。

`UsageType` は `Bytes` をサフィックスとして使用しますが、CUR のエクスポートとインポートの使用量は GB 単位で測定され請求されます。

UsageType	単位	詳細度	説明
REG-ExportDataSize-Bytes	GB	サイズ	S3 へのデータエクスポートの料金。DynamoDB でのデータエクスポートは、エクスポートが作成された時点の DynamoDB ベーステーブル (テーブルデータおよびローカルセカンダリインデックス) のサイズに基づいて課金されます。
REG-ImportDataSize-Bytes	GB	サイズ	S3 からのデータインポートの料金。サイズは、Amazon S3 のデータのオブジェ

UsageType	単位	詳細度	説明
			クトの非圧縮サイズに基づいて計算されます。GSI を使用したテーブルへのインポートには追加料金はかかりません。
REG-IncrementalExportDataSize-Bytes	GB	サイズ	継続バックアップから増分エクスポートを行うために処理されたデータのサイズに基づく料金。

データ転送

データ転送アクティビティは DynamoDB サービスに関連して表示される場合があります。DynamoDB では、インバウンドデータ転送の料金は発生しません。また、DynamoDB と同じ AWS リージョン内の他の AWS サービスとの間のデータ転送の料金も発生しません (つまり、GB あたり 0.00 USD)。AWS リージョン間 (米国東部 [バージニア北部] リージョンの DynamoDB と欧州 [アイルランド] リージョンの Amazon EC2 間など) で転送されるデータは、転送元と転送先の両方で料金が発生します。

UsageType	単位	詳細度	説明
REG-DataTransfer-In-Bytes	GB	単位	インターネットから DynamoDB へのデータ転送。
REG-DataTransfer-Out-Bytes	GB	単位	DynamoDB からインターネットへのデータ転送。

CloudWatch Contributor Insights

CloudWatch Contributor Insights for DynamoDB は、DynamoDB テーブルで最も頻繁にアクセスされスロットリングされるキーを特定する診断ツールです。以下の `UsageType` 料金は、「AmazonDynamoDB」ではなく「AmazonCloudWatch」ProductCode の下に表示されます。

UsageType	単位	詳細度	説明
REG-CW:ContributorEventsManaged	処理されたイベント	単位	処理された DynamoDB イベントの量。例えば、CloudWatch Contributor Insights が有効になっているテーブルでは、アイテムが読み書きされるたびに 1 イベントとしてカウントされます。テーブルにソートキーがある場合は、2 イベントとしてカウントされます。
REG-CW:ContributorRulesManaged	ルール数	月	Cloud Watch Contributor Insights を有効にすると、DynamoDB は最もアクセス数の多いアイテムと最もスロットリングされたキーを識別するルールを作成します。この料金は、CloudWatch contributor insights のログ用に設定された各エンティティ (テー

UsageType	単位	詳細度	説明
			ブルおよび GSI) に追加されたルールに対して発生します。

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) は、サービスで選択されたインスタンスタイプに基づいて時間単位で請求されます。以下の料金は、プロビジョニングされた DynamoDB Accelerator インスタンスに関するものです。以下の UsageType 料金は、「AmazonDynamoDB」ではなく「AmazonDAX」ProductCode の下に表示されます。

UsageType	単位	詳細度	説明
REG-NodeUsage:dax-<INSTANCETYPE>	ノード時間	時	特定のインスタンスタイプの 1 時間あたりの使用量。料金は、ノードが起動されてから終了するまでの、消費されたノード時間あたりの価格です。消費されたノード時間が 1 時間に満たない場合でも、1 時間分として請求されます。DAX は DAX クラスター内のノードごとに課金されます。クラスターに複数のノードがある場合、請求レポートには複数の明細項目が表示されます。

インスタンスタイプは、次の表のような値になります。ノードタイプの詳細については、「ノード」を参照してください。

<INSTANCETYPE>		
r3.2xlarge	r4.8xlarge	r5.8xlarge
r3.4xlarge	r4.large	r5.large
r3.8xlarge	r4.xlarge	r5.xlarge
r3.2xlarge	r5.12xlarge	t2.medium
r3.4xlarge	r4.large	r5.large
r3.xlarge	r5.16xlarge	t2.small
r4.16xlarge	r5.24xlarge	t3.medium
r4.2xlarge	r5.2xlarge	t3.small
r4.4xlarge	r5.4xlarge	

キャパシティモードを切り替える際の考慮事項

DynamoDB テーブルを作成する場合、オンデマンドまたはプロビジョンドキャパシティモードのいずれかを選択する必要があります。

テーブルは、オンデマンドモードからプロビジョンドキャパシティモードにいつでも切り替えることができます。キャパシティモード間で複数の切り替えを行う場合は、次の条件が適用されます。

- オンデマンドモードで新しく作成したテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、テーブルの作成タイムスタンプから 24 時間後のみです。
- オンデマンドモードの既存のテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、オンデマンドへの切り替えを示す最後のタイムスタンプから 24 時間後のみです。

トピック

- プロビジョンドキャパシティモードからオンデマンドキャパシティモードへの切り替え
- オンデマンドキャパシティモードからプロビジョンドキャパシティモードへの切り替え

プロビジョンドキャパシティモードからオンデマンドキャパシティモードへの切り替え

プロビジョンドモードでは、予想されるアプリケーションのニーズに基づいて読み込みおよび書き込みキャパシティを設定します。テーブルをプロビジョニングモードからオンデマンドモードに更新するときは、アプリケーションで実行することが予測される読み込みおよび書き込みスループットを指定する必要はありません。DynamoDB オンデマンドは、読み込みおよび書き込みリクエストの料金が従量制であるため、使用した分だけを支払います。これにより、コストとパフォーマンスのバランスを簡単に取ることができます。オプションで、個々のオンデマンドテーブルおよび関連するグローバルセカンダリインデックスの読み込みや書き込み (または両方) の最大スループットを設定して、コストと使用量を制限し続けることができます。特定のテーブルやインデックスの最大スループットの設定の詳細については、「オンデマンドテーブルの最大スループット」を参照してください。

プロビジョンドキャパシティモードからオンデマンドキャパシティモードに切り替えると、DynamoDB は、テーブルやパーティションの構造にいくつかの変更を行います。この処理には数分かかることもあります。切り替え期間中、テーブルは以前にプロビジョニングされた書き込みキャパシティーユニットおよび読み込みキャパシティーユニットの両方と整合性のあるスループットを提供します。

オンデマンドキャパシティモードの初期スループット

最近、既存のテーブルをオンデマンドキャパシティモードに初めて切り替えた場合、テーブルがオンデマンドキャパシティモードを使用してトラフィックを以前に処理していなくても、テーブルは次に示す以前のピーク設定を持ちます。

考えられるシナリオの例を以下に示します。

- 4,000 の WCU と 12,000 の RCU 未満に設定されたプロビジョンドテーブルが、これまでにそれを超える値にプロビジョニングされたことがない場合。このテーブルを初めてオンデマンドに切り替えると、DynamoDB は、少なくとも 4,000 の書き込みユニット/秒、12,000 の読み込みユニット/秒を即座に維持できるようにスケールアウトします。
- 8,000 の WCU と 24,000 の RCU に設定されたプロビジョンドテーブルの場合。このテーブルをオンデマンドに切り替えると、少なくとも 8,000 の書き込みユニット/秒、24,000 の読み込みユニット/秒を引き続き維持できます。

- 8,000 WCU と 24,000 RCU で設定されたプロビジョンドテーブルが、一定期間にわたり 6,000 の書き込みユニット/秒、18,000 の読み込みユニット/秒を消費した場合。このテーブルをオンデマンドに切り替えると、少なくとも 8,000 の書き込みユニット/秒、24,000 の読み込みユニット/秒を引き続き維持できます。以前のトラフィックにより、このテーブルはスロットリングなしではるかに高いレベルのトラフィックを維持できる可能性があります。
- 以前は 10,000 の WCU と 10,000 の RCU でプロビジョニングされていたテーブルが、現在は 10 の RCU と 10 の WCU でプロビジョニングされている場合。このテーブルをオンデマンドに切り替えると、少なくとも 10,000 の書き込みユニット/秒、10,000 の読み込みユニット/秒を維持できます。

自動スケーリング設定

テーブルをプロビジョニングモードからオンデマンドモードに更新すると、以下のようになります。

- コンソールを使用している場合、Auto Scaling 設定 (ある場合) がすべて削除されます。
- AWS CLI または AWS SDK を使用している場合、Auto Scaling 設定はすべて保持されます。これらの設定は、テーブルをもう一度プロビジョニングされた請求モードに更新すると適用できます。

オンデマンドキャパシティモードからプロビジョンドキャパシティモードへの切り替え

オンデマンドキャパシティーモードからプロビジョンドキャパシティーモードに戻すと、テーブルは、テーブルがオンデマンドキャパシティーモードに設定されたときに到達した前のピークと整合性のあるスループットを提供します。

容量の管理

テーブルをオンデマンドモードからプロビジョニングモードに更新するときは、以下の点を考慮に入れてください。

- AWS CLI または AWS SDK を使用している場合、Amazon CloudWatch を使用して消費履歴 (`ConsumedWriteCapacityUnits` および `ConsumedReadCapacityUnits` メトリクス) を参照し、新しいスループット設定を決定することで、テーブルおよびグローバルセカンダリインデックスのプロビジョンされた適切な容量設定を選択します。

> **Note**
>
> グローバルテーブルからプロビジョニングモードに切り替える場合、すべてのリージョンレプリカにわたって基本テーブルおよびグローバルセカンダリインデックスの最大消費量を調べた上で、新しいスループット設定を決定します。

- オンデマンドモードからプロビジョンドモードに戻す場合は、移行中にテーブルまたはインデックスのキャパシティを処理できるように、最初のプロビジョンドユニットを十分高い値に設定してください。

Auto Scaling の管理

テーブルをプロビジョニングモードからオンデマンドモードに更新すると、以下のようになります。

- コンソールを使用する場合は、以下のデフォルトで自動スケーリングを有効にすることをお勧めします。
  - ターゲット使用率: 70%
  - プロビジョニングされた最小キャパシティー: ユニット 5 個
  - プロビジョニングされた最大キャパシティー: リージョンの最大値
- AWS CLI または SDK を使用している場合、前の Auto Scaling 設定 (ある場合) が保持されます。

DynamoDB を他の AWS サービスで使用する

Amazon DynamoDB を他の AWS サービスと統合することで、繰り返しのタスクを自動化したり、複数のサービスにまたがるアプリケーションを構築できます。

トピック

Amazon Cognito を使用したファイル内の AWS 認証情報の設定

ウェブおよびモバイルアプリケーション用の AWS 認証情報を取得する際には、Amazon Cognito を使用することをお勧めします。Amazon Cognito を使用すると、AWS 認証情報をファイル上でハードコーディングする必要がありません。AWS Identity and Access Management (IAM) ロールを使用して、アプリケーションの認証されたユーザーと認証されていないユーザーのために一時的な認証情報が生成されます。

たとえば、Amazon Cognito の認証されていないロールを使用して Amazon DynamoDB ウェブサービスにアクセスするように JavaScript ファイルを設定するには、次の手順を実行します。

Amazon Cognito と統合する認証情報を設定するには

1. 認証されていない ID を許可する Amazon Cognito ID プールを作成します。

```
aws cognito-identity create-identity-pool \
    --identity-pool-name DynamoPool \
    --allow-unauthenticated-identities \
    --output json
{
    "IdentityPoolId": "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
    "AllowUnauthenticatedIdentities": true,
    "IdentityPoolName": "DynamoPool"
}
```

2. 次のポリシーを `myCognitoPolicy.json` という名前のファイルにコピーします。ID プール ID (*us-west-2:12345678-1ab2-123a-1234-a12345ab12*) を、前のステップで取得した独自の `IdentityPoolId` に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-west-2:12345678-1ab2-123a-1234-
a12345ab12"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

3. 以前のポリシーを予期する IAM ロールを作成します。このようにして、Amazon Cognito は `Cognito_DynamoPoolUnauth` ロールを引き受けられる信頼済みエンティティになります。

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. 管理ポリシー (`AmazonDynamoDBFullAccess`) をアタッチして、DynamoDB へのフルアクセスを `Cognito_DynamoPoolUnauth` ロールに許可します。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> または、DynamoDB へのきめ細かなアクセスを許可することもできます。詳細については、「詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用」を参照してください。

5. IAM ロールの Amazon リソースネーム (ARN) を取得してコピーします。

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. DynamoPool ID プールに Cognito_DynamoPoolUnauth ロールを追加します。指定するフォーマットは KeyName=string です。ここで KeyName は unauthenticated で、文字列は前のステップで取得したロールの ARN です。

```
aws cognito-identity set-identity-pool-roles \
--identity-pool-id "us-west-2:12345678-1ab2-123a-1234-a12345ab12" \
--roles unauthenticated=arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth --
output json
```

7. ファイルで Amazon Cognito 認証情報を指定します。それに応じて IdentityPoolId および RoleArn を変更します。

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

Amazon Cognito 認証情報を使用して、DynamoDB ウェブサービスに対して JavaScript プログラムを実行できます。詳細については、「AWS SDK for JavaScript 入門ガイド」の「ウェブブラウザ内で認証情報を設定する」を参照してください。

DynamoDB から Amazon Redshift へのデータのロード

Amazon Redshift は、高度なビジネスインテリジェンス機能と強力な SQL ベースのインターフェイスを使用して Amazon DynamoDB を補完します。DynamoDB テーブルのデータを Amazon Redshift にコピーすると、Amazon Redshift クラスター内の他のテーブルとの結合など、複雑なデータ分析のクエリを実行できます。

プロビジョニングされたスループットの面から見ると、DynamoDB テーブルからのコピーオペレーションによって、テーブルの読み込みキャパシティーが低下します。データをコピーした後は、Amazon Redshift 内で SQL クエリを実行しても DynamoDB に全く影響はありません。これは、DynamoDB 自体に対してではなく、DynamoDB からのデータのコピーに対してクエリが実行されるためです。

DynamoDB テーブルからデータをロードする前に、まずデータのロード先として Amazon Redshift テーブルを作成する必要があります。注意していただきたいのは、NoSQL 環境から SQL 環境にデータをコピーしていることと、1 つの環境での特定のルールが他の環境には適用されないことです。考慮すべき相違点の例を以下に示します。

- DynamoDB テーブル名には「.」(ドット) や「-」(ダッシュ) を含む最大 255 文字を使用することができ、大文字と小文字が区別されます。Amazon Redshift テーブル名は 127 文字までに制限されます。ドットやダッシュは使用できず、大文字と小文字は区別されません。さらに、テーブル名は Amazon Redshift の予約語と重複してはいけません。
- DynamoDB では、SQL の NULL の概念がサポートされていません。Amazon Redshift において、DynamoDB 内の空白またはブランクの属性値をどのように解釈するか、つまり、NULL と空白のフィールドのどちらで処理するかを指定する必要があります。
- DynamoDB のデータ型は、Amazon Redshift のデータ型と直接対応していません。Amazon Redshift テーブルの各列において、データ型が正しく、DynamoDB からのデータに対応したサイズであることを確認する必要があります。

Amazon Redshift SQL からの COPY コマンドの例を以下に示します。

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
credentials 'aws_access_key_id=<Your-Access-Key-ID>;aws_secret_access_key=<Your-Secret-Access-Key>'
readratio 50;
```

この例では、DynamoDB のソーステーブルは、`my-favorite-movies-table` です。Amazon Redshift のターゲットテーブルは、`favoritemovies` です。`readratio 50` 節により、プロビジョニングされたスループットの消費される割合が制限されます。この場合、COPY コマンドでは、`my-favorite-movies-table`用にプロビジョニングされた読み込みキャパシティーの 50% 以下しか使用されません。この割合については、未使用のプロビジョニングされたスループットの平均よりも小さい値に設定することを強くお勧めします。

DynamoDB から Amazon Redshift にデータをロードする詳細な手順については、Amazon Redshift データベースデベロッパーガイドの次のセクションを参照してください。

- DynamoDB テーブルからのデータのロード
- The COPY command
- COPY の例

Amazon EMR での Apache Hive を使用した DynamoDB データの処理

Amazon DynamoDB は、Amazon EMR で実行されるデータウェアハウスアプリケーションである Apache Hive と統合されています。Hive では、DynamoDB テーブルにあるデータの読み込み/書き込みが行えます。これにより、以下のことが可能になります。

- SQL に似た言語 (HiveQL) を使用して、その時点の DynamoDB データがクエリできます。
- DynamoDB テーブルから Amazon S3 バケットに (あるいはその逆の方向で) データをコピーできます。
- DynamoDB テーブルから Hadoop Distributed File System (HDFS) に (あるいはその逆の方向で) データをコピーできます。
- DynamoDB テーブルで JOIN オペレーションを実行できます。

トピック

概要

Amazon EMR は、膨大な量のデータの迅速かつコスト効率の良い処理を容易に行えるサービスです。Amazon EMR を使用するには、Hadoop のオープンソースフレームワークを実行する Amazon EC2 インスタンスの、マネージド型クラスターを起動します。Hadoopは、タスクがクラスター内の

複数のノードにマッピングされている場合に、MapReduce アルゴリズムを実装するための分散型アプリケーションです。各ノードは、他のノードと並列的に指定された作業を処理します。それらの出力は単一のノードに集約され、最終的な結果が得られます。

Amazon EMR では、永続的または一時的のどちらかを選択して、クラスターを起動することができます。

- 永続的なクラスターは、シャットダウンされるまでその実行が継続します。永続的なクラスターは、データ分析やデータウエアハウスなどを含む、インタラクティブな使用に最適です。
- 一時的なクラスターは、ジョブフローの処理に必要な長さだけ実行された後、自動的にシャットダウンされます。一時的なクラスターは、スクリプトの実行など、定期的な処理タスクに最適です。

Amazon EMR のアーキテクチャと管理の詳細については、Amazon EMR 管理ガイドを参照してください。

ユーザーは、Amazon EMR クラスターを起動する際に、初期段階での Amazon EC2 インスタンスの数とタイプを指定します。また、(Hadoop 自体に加えて) クラスターで実行する他の分散型アプリケーションも指定する必要があります。これらのアプリケーションには Hue、Mahout、Pig、Spark などが含まれます。

Amazon EMR のアプリケーションについては、Amazon EMR リリース ガイドを参照してください。

クラスター構成に応じて、次の中の 1 つ以上のノードタイプが利用可能です。

- リーダーノード – クラスターを管理します。MapReduce の実行可能ファイル、および生データのサブセットを、コアおよびタスクインスタンスグループに適切に分散させます。また、実行された各タスクのステータスを追跡し、インスタンスグループの正常性をモニタリングします。クラスターにはリーダーノードが 1 つのみ存在します。
- コアノード – MapReduce タスクを実行しデータを保存します。その際、Hadoop Distributed File System (HDFS) を使用します。
- タスクノード – (オプション) MapReduce タスクを実行します。

チュートリアル:Amazon DynamoDB と Apache Hive の使用

このチュートリアルでは、まず、Amazon EMR クラスターを起動した上で、DynamoDB テーブルに格納されているデータの処理に Apache Hive を使用していきます。

Hiveは、Hadoop 用のデータウェアハウスアプリケーションで、複数のソースからのデータを処理および分析することを可能にします。Hive では、SQL に似た言語である HiveQL を使用しながら、ローカルの Amazon EMR クラスター、または外部のデータソース (Amazon DynamoDB など) に保存されたデータを操作できます。

詳細については、「Hive Tutorial」を参照してください。

トピック

開始する前に

このチュートリアルでは、以下が必要になります。

- AWS アカウント。アカウントをお持ちでない場合は、「AWS へのサインアップ」を参照してください。
- SSH クライアント (セキュアシェル)。SSH クライアントを使用して、Amazon EMR クラスターのリーダーノードに接続し、対話型コマンドを実行します。ほとんどの Linux、Unix、および Mac OS X の実装では、SSH クライアントをデフォルトで利用できます。Windows ユーザーの場合は、SSH がサポートする PuTTY クライアントをダウンロードしてインストールできます。

次のステップ

ステップ1:Amazon EC2 キーペアを作成する

ステップ1:Amazon EC2 キーペアを作成する

このステップでは、Amazon EMR リーダーノードに接続し、Hive コマンドを実行するために必要な Amazon EC2 のキーペアを作成します。

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. リージョン (例: `US West (Oregon)`) を選択します。これは、DynamoDB テーブルが配置されているリージョンと同じリージョンにする必要があります。
3. ナビゲーションペインで、[キーペア] を選択します。
4. [キーペアの作成] を選択します。
5. [Key pair name] (キーペア名) にキーペアの名前 (例: `mykeypair`) をタイプし、[Create] (作成) を選択します。
6. プライベートキーファイルをダウンロードします。このファイル名は、`.pem` (`mykeypair.pem` など) で終わります。ダウンロードしたプライベートキーファイルは安全な場所に保存します。ここで作成するキーペアにより起動する Amazon EMR クラスターへのアクセスには、このファイルが必要となります。

   > ⚠ Important
   >
   > キーペアが失われた場合、Amazon EMR クラスターのリーダーノードに接続することはできなくなります。

   キーペアの詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「Amazon EC2 キーペア」を参照してください。

次のステップ

ステップ 2: Amazon EMR クラスターを起動します

ステップ 2: Amazon EMR クラスターを起動します

このステップでは、Amazon EMR クラスターを設定して起動します。このクラスターには、Hive および DynamoDB 用のストレージハンドラーが、既にインストールされています。

1. Amazon EMR コンソール (https://console.aws.amazon.com/emr) を開きます。
2. [クラスターの作成] を選択します。
3. [Create Cluster - Quick Options] (クラスターの作成 – クイックオプション) ページで、以下を実行します。

a. [Cluster name] (クラスター名) にクラスターの名前を入力します (例: `My EMR cluster` など)。

b. [EC2 key pair] (EC2 キーペア) で、先に作成してあるキーペアを選択します。

その他の設定はデフォルト値のままにしておきます。

4. [クラスターを作成] を選択します。

クラスターを起動するには数分間かかります。このプロセスの進捗状況は、Amazon EMR コンソールの [Cluster Details] (クラスターの詳細) ページで確認できます。

ステータスが `Waiting` に変わると、クラスターは使用可能状態になっています。

クラスターのログファイルと Amazon S3

Amazon EMR クラスターは、クラスターのステータスやデバッグに関する情報が記載されたログファイルを生成します。[Create Cluster - Quick Options (クラスターの作成-クイックオプション)] のデフォルト設定には、Amazon EMR でのログ記録に関する設定が含まれています。

まだ Amazon S3 バケットが存在しない場合は、AWS Management Console マネジメントコンソールにより作成されます。このバケットの名前は `aws-logs-`*`account-id`*`-`*`region`* となります。ここで、 *`account-id`* は AWS アカウント番号で、*`region`*は、クラスターを起動したリージョン (例えば `aws-logs-123456789012-us-west-2`) です。

> **Note**
>
> Amazon S3 コンソールを使用して、このログファイルを表示できます。詳細については、「Amazon EMR 管理ガイド」の「ログファイルを表示する」を参照してください。

このバケットは、ログ記録以外の目的でも使用できます。例えば、バケットを Hive スクリプトを保存する場所として使用することや、Amazon DynamoDB から Amazon S3 にデータをエクスポートする際の送信先として使用することが可能です。

次のステップ

ステップ 3: リーダーノードに接続します

ステップ 3: リーダーノードに接続します

Amazon EMR クラスターのステータスが `Waiting` の状態であれば、SSH を使用してリーダーノードに接続し、コマンドラインによるオペレーションを実行できます。

1. Amazon EMR コンソールで、クラスターの名前を選択して、そのステータスを表示します。
2. [Cluster Details] (クラスターの詳細) ページで、[Leader public DNS] (リーダーのパブリック DNS) フィールドを見つけます。これは、Amazon EMR クラスターのリーダーノードのためのパブリック DNS 名です。
3. DNS 名の右側で、SSH リンクを選択します。
4. [Connect to the Leader Node Using SSH] (SSH を使用してマスターノードに接続する) にある手順を実行します。

   使用しているオペレーティングシステムに応じて、Windows タブまたは Mac/Linux タブを選択し、指示に従いリーダーノードに接続します。

SSH または PuTTY を使用してリーダーノードへの接続が完了すると、次のようなコマンドプロンプトが表示されます。

```
[hadoop@ip-192-0-2-0 ~]$
```

次のステップ

ステップ 4: HDFS にデータをロードします

ステップ 4: HDFS にデータをロードします

このステップでは、データファイルを Hadoop Distributed File System (HDFS) にコピーし、データファイルにマップする外部 Hive テーブルを作成します。

サンプルデータをダウンロードします

1. サンプルデータアーカイブ (`features.zip`) のダウンロード:

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. アーカイブからの `features.txt` ファイルの抽出:

```
unzip features.zip
```

3. features.txt ファイルの最初の数行を表示:

```
head features.txt
```

この結果は以下のようになります。

```
1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

features.txt ファイルには、米国地名委員会 (http://geonames.usgs.gov/domestic/download_data.htm) で提供されるデータのサブセットが含まれます。ここでの各行のフィールドは、次を表しています。

- フィーチャ ID (固有の識別子)
- 名前
- クラス (湖、森林、小川、その他)
- 状態
- 緯度 (度)
- 経度 (度)
- 高度 (フィート)

4. コマンドプロンプトに次のコマンドを入力します。

```
hive
```

コマンドプロンプトがこのように変わります: hive>

5. 次の HiveQL ステートメントを入力して、ネイティブ Hive テーブルを作成します。

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

6. 次の HiveQL ステートメントを入力して、データとともにテーブルをロードします。

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. これで、`features.txt` ファイルからのデータが格納された、ネイティブ Hive テーブルが用意できます。これを確認するには、次の HiveQL ステートメントを入力します。

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

   出力には、州の一覧といくつかの地理的特徴が表示されます。

次のステップ

[ステップ 5: データを DynamoDB にコピーします](#)

ステップ 5: データを DynamoDB にコピーします

このステップでは、データを Hive テーブル (`hive_features`) から DynamoDB の新しいテーブルにコピーします。

1. https://console.aws.amazon.com/dynamodb/ で DynamoDB コンソールを開きます。
2. [テーブルの作成] を選択します。

3. [Create DynamoDB table] (DynamoDB テーブルの作成) ページで、次の操作を行います。

    a. [Table] (テーブル) に **`Features`** とタイプします。

    b. [Partition key] (パーティションキー) フィールドの [Primary key] (プライマリキー) に **`Id`** とタイプします。データ型を [数値] に設定します。

        [Use default settings] (デフォルト設定の使用) を消します。[Provisioned Capacity] (プロビジョニングキャパシティ) に以下をタイプします。

        - [Read Capacity Units] (読み込みキャパシティユニット) — `10`
        - [Write Capacity Units] (書き込みキャパシティユニット) — `10`

    [Create] を選択します。

4. Hive プロンプトが表示された状態で、次の HiveQL ステートメントを入力します。

```
CREATE EXTERNAL TABLE ddb_features
    (feature_id   BIGINT,
    feature_name  STRING,
    feature_class STRING,
    state_alpha   STRING,
    prim_lat_dec  DOUBLE,
    prim_long_dec DOUBLE,
    elev_in_ft    BIGINT)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
    "dynamodb.table.name" = "Features",

 "dynamodb.column.mapping"="feature_id:Id,feature_name:Name,feature_class:Class,state_alpha
);
```

    これで、Hive と DynamoDB のフィーチャーテーブル間でのマッピングが確立されました。

5. 次の HiveQL ステートメントを入力して DynamoDB にデータをインポートします。

```
INSERT OVERWRITE TABLE ddb_features
SELECT
    feature_id,
    feature_name,
    feature_class,
    state_alpha,
    prim_lat_dec,
    prim_long_dec,
    elev_in_ft
FROM hive_features;
```

Hive が MapReduce ジョブを送信します。このジョブは Amazon EMR クラスターによって処理されます。このジョブが完了するまで数分間かかります。

6. データが DynamoDB にロードされていることを確認します。

   a. DynamoDB コンソールのナビゲーションペインで、[Tables] (テーブル) を選択します。

   b. フィーチャーテーブルを選択し、[Items] (アイテム) タブを選択して、データを表示します。

次のステップ

[ステップ 6: DynamoDB テーブル内のデータをクエリを行います]

ステップ 6: DynamoDB テーブル内のデータをクエリを行います

このステップでは、HiveQL を使用して DynamoDB の フィーチャーテーブルをクエリします。以下の Hive クエリを使用します。

1. すべてのフィーチャタイプ (`feature_class`) をアルファベット順に取得:

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. 名前が文字「M」で始まるすべての湖を取得：

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. 1 マイル (5,280 フィート) より高度が高く、少なくとも 3 つのフィーチャを持つ州を取得:

```
SELECT state_alpha, feature_class, COUNT(*)
FROM ddb_features
WHERE elev_in_ft > 5280
```

```
GROUP by state_alpha, feature_class
HAVING COUNT(*) >= 3
ORDER BY state_alpha, feature_class;
```

次のステップ

[ステップ 7: (オプション) クリーンアップする](#)

ステップ 7: (オプション) クリーンアップする

この段階で、チュートリアルは完了しています。引き続きこのセクションを読むことで、Amazon EMR での DynamoDB データの操作について、さらに詳細に学習することができます。このステップを実行する場合は、Amazon EMR クラスターを稼働させ続けることができます。

クラスターが不要である場合は、それを終了し、関連するリソースの削除を行ってください。これにより、不要なリソースに対して課金されることを回避できます。

1. Amazon EMR クラスターを終了します。
   a. Amazon EMR コンソール (https://console.aws.amazon.com/emr) を開きます。
   b. Amazon EMR クラスターを選択した上で、[Terminate] (終了) を選択し確認します。
2. DynamoDB 内のフィーチャーテーブルを削除します。
   a. https://console.aws.amazon.com/dynamodb/ で DynamoDB コンソールを開きます。
   b. ナビゲーションペインで [テーブル] を選択します。
   c. フィーチャ-テーブルを選択します。[Actions] (アクション) メニューから、[Delete Table] (テーブルの削除) を選択します。
3. Amazon EMR ログファイルが保存されている Amazon S3 バケットを削除します。
   a. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
   b. バケットのリストから [`aws-logs-` *`accountID`*`-`*`region`*] を選択します。ここで、*`accountID`* は使用している AWS アカウント番号であり、*`region`* はクラスターを起動したリージョンです。
   c. [Actions (アクション)] メニューから、[Delete (削除)] を選択します。

Hive に外部テーブルを作成します

チュートリアル:Amazon DynamoDB と Apache Hive の使用 において、DynamoDB テーブルにマッピングされた外部 Hive テーブルを作成しました。この状態で外部テーブルに対して HiveQL ステートメントを発行すると、読み込み/書き込みの操作が DynamoDB テーブルに渡されます。

外部テーブルは、他の場所で管理および格納されているデータソースへのポインタと捉えることができます。ここでは、基盤となるそのデータソースは、DynamoDB テーブルです。(このテーブルは外部に用意する必要があります。Hive 内から DynamoDB テーブルを作成、更新、または削除することはできません。) 外部テーブルの作成には、`CREATE EXTERNAL TABLE` ステートメントを使用します。その後は、DynamoDB 内のデータが Hive 内にローカルに格納されているかのように、HiveQL を使用してそのデータを操作できます。

> Note
>
> `INSERT` ステートメントを使用すると、外部テーブルにデータを挿入でき、また、`SELECT` ステートメントにより、そのテーブルのデータを選択できます。ただし、テーブル内のデータを操作するために、`UPDATE` または `DELETE` ステートメントを使用することはできません。

外部テーブルが必要でなくなった時に、`DROP TABLE` を使ってそのテーブルを削除します。この場合でも、`DROP TABLE` は Hive の外部テーブルだけを削除します。このオペレーションは、基盤となっている DynamoDB テーブルまたはそのデータには影響を与えません。

トピック

- CREATE EXTERNAL TABLE 構文
- データ型のマッピング

CREATE EXTERNAL TABLE 構文

DynamoDB テーブルにマッピングする外部 Hive テーブルを作成するための HiveQL 構文を次に示します。

```
CREATE EXTERNAL TABLE hive_table

 (hive_column1_name hive_column1_datatype, hive_column2_name hive_column2_datatype...)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
```

```
TBLPROPERTIES (
    "dynamodb.table.name" = "dynamodb_table",
    "dynamodb.column.mapping" =
 "hive_column1_name:dynamodb_attribute1_name,hive_column2_name:dynamodb_attribute2_name..."
);
```

1 行目から CREATE EXTERNAL TABLE ステートメントが始まりますが、作成する Hive テーブルの名前 (hive_table) を指定します。

2 行目では、hive_table の列とデータ型を指定します。DynamoDB テーブルの属性に対応する列とデータ型を定義する必要があります。

3 行目は STORED BY 句で、ここでは、Hive と DynamoDB テーブル間のデータ管理を処理するクラスを指定します。DynamoDB の場合、STORED BY は 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler' に設定する必要があります。

4 行目で TBLPROPERTIES 句が始まり、以下の DynamoDBStorageHandler のパラメータを定義しています。

- dynamodb.table.name – DynamoDB テーブルの名前。
- dynamodb.column.mapping – Hive テーブルの列名と DynamoDB テーブルの対応する属性のペア。各ペアの形式は、hive_column_name:dynamodb_attribute_name とし、それぞれをカンマで区切ります。

次の点に注意してください。

- Hive テーブル名の名前は、DynamoDB テーブル名と同じである必要はありません。
- Hive テーブルの列名は、DynamoDB テーブルの列名と同じである必要はありません。
- dynamodb.table.name で指定されたテーブルは、DynamoDB 内に置いてある必要があります。
- 複数 dynamodb.column.mapping:
  - DynamoDB テーブルには、キースキーマ属性をマッピングする必要があります。これには、パーティションキーと (それが存在する場合は) ソートキー が含まれます。
  - DynamoDB テーブルの非キー属性のマッピングは、必須ではありません。ただし、Hive テーブルに対しクエリを実行しても、マッピングしていないこれらの属性のデータは表示されません。
  - Hive テーブルの列のデータ型と DynamoDB 属性のデータ型に互換性がない場合、Hive テーブルをクエリすると、これらの列には NULL が表示されます。

Note

CREATE EXTERNAL TABLE ステートメントでは、TBLPROPERTIES 句に関する検証は行われません。dynamodb.table.name および dynamodb.column.mapping に指定した値は、テーブルへのアクセスを試みた時点で、 クラスによってのみ評価されます。

データ型のマッピング

次の表に、DynamoDB でのデータ型と、それに互換性のある Hive でのデータ型を示します。

DynamoDB データ型	Hive データ型
文字列	STRING
数値	BIGINT-または-DOUBLE
バイナリ	BINARY
文字列セット	ARRAY<STRING>
数値セット	ARRAY<BIGINT> 、または ARRAY<DOUBLE>
バイナリセット	ARRAY<BINARY>

Note

以下の DynamoDB データ型は、DynamoDBStorageHandler クラスではサポートされないため、dynamodb.column.mapping とともに使用することはできません。

- マップ
- リスト
- ブール値
- Null

ただし、これらのデータ型を処理する必要がある場合は、DynamoDB アイテム全体を、マップ内のキーと値の両方の文字列のマップとして表す item という名前の 1 つのエンティティ

を作成できます。詳細については、「列マッピングを使用しないデータをコピー」を参照してください。

数値型の DynamoDB 属性をマッピングする場合は、適切な Hive タイプを選択する必要があります。

- Hive `BIGINT` 型は 8 バイトの符号付き整数です。これは、Java での `long` データ型と等価です。
- Hive `DOUBLE` 型は 8 ビット倍精度浮動小数点数です。これは、Java での `double` 型と等価です。

選択した Hive データ型よりも精度が高い数値データが DynamoDB に格納されている場合、DynamoDB データにアクセスすると精度を損失する可能性があります。

DynamoDB から HDFS (または Amazon S3) にバイナリ型のデータをエクスポートすると、データは Base64 でエンコードされた文字列として格納されます。Amazon S3 または HDFS から DynamoDB に対し、バイナリ型でデータをインポートする場合には、そのデータが Base64 文字列としてエンコードされていることを確認します。

HiveQL ステートメントの処理

Hive は、MapReduceジョブを実行するためのバッチ指向フレームワークとして、Hadoop上で動作するアプリケーションです。HiveQL ステートメントを発行すると、Hive は結果をすぐに返すことができるかどうか、あるいは、MapReduce ジョブを送信する必要があるかどうかを判断します。

例えば、(チュートリアル:Amazon DynamoDB と Apache Hive の使用 より) ddb_features テーブルを考えてみます。次の Hive クエリは、州の省略した名前と、それぞれの州に存在する山頂の数を出力します。

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive は結果をすぐに返しません。代わりに、Hive からは MapReduceジョブが送信され、Hadoop フレームワークによって処理されます。Hive は、このジョブが完了するまで待機してから、クエリの結果を表示します。

```
AK  2
AL  2
```

```
AR  2
AZ  3
CA  7
CO  2
CT  2
ID  1
KS  1
ME  2
MI  1
MT  3
NC  1
NE  1
NM  1
NY  2
OR  5
PA  1
TN  1
TX  1
UT  4
VA  1
VT  2
WA  2
WY  3
Time taken: 8.753 seconds, Fetched: 25 row(s)
```

ジョブのモニタリングとキャンセル

Hive が Hadoop ジョブを起動すると、そのジョブからの出力が表示されます。表示されるジョブの完了ステータスは、ジョブの進捗に応じて更新されます。このステータスは、長時間更新されないこともります。(この現象は、読み込みキャパシティーのプロビジョニングが低く設定されている、大型の DynamoDB テーブルをクエリした場合に発生する可能性があります。)

Ctrl+C とタイプすることで、完了する前に任意のタイミングで、このジョブをキャンセルすることが可能です。

DynamoDB 内データのクエリ

次の例で、DynamoDB に保存されたデータへのクエリに、HiveQL を使用する方法をいくつか示します。

これらの例は、チュートリアル (ステップ 5: データを DynamoDB にコピーします) にある ddb_features テーブルを参照しています。

トピック

集計関数の使用

HiveQL では、データ値を集計するための組み込み関数が利用できます。例えば、選択した列で最大値を検索するためには MAX 関数を使用できます。次の例では、コロラド州で標高が最も高いという特徴を返します。

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

GROUP BY 句および HAVING 句の使用

GROUP BY 句を使用して、複数のレコードのデータを収集できます。これは、多くの場合、SUM、COUNT、MIN、または MAX のような集計関数で使用されます。また、一定の基準を満たさない結果を破棄する場合には、HAVING 句を使用することもできます。

次の例では、ddb_features テーブルの中に 5 つ以上の特徴を持つ州での、最も高い標高のリストを返します。

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

2 つの DynamoDB テーブルの結合

次の例では、DynamoDB 内のテーブルに対し、別の Hive テーブル (east_coast_states) をマッピングしています。SELECT ステートメントにより、これらの 2 つのテーブル間での結合が行われます。結合はクラスターで計算され、以下を返します。結合は DynamoDB 内では発生しません。

以下のデータが保存されている、EastCoastStates という名前の DynamoDB テーブルがあるとします。

```
StateName       StateAbbrev

Maine           ME
New Hampshire   NH
Massachusetts   MA
Rhode Island    RI
Connecticut     CT
New York        NY
New Jersey      NJ
Delaware        DE
Maryland        MD
Virginia        VA
North Carolina  NC
South Carolina  SC
Georgia         GA
Florida         FL
```

このテーブルは、east_coast_states という名前の Hive 外部テーブルとして利用可能であると仮定します。

```
CREATE EXTERNAL TABLE ddb_east_coast_states (state_name STRING, state_alpha STRING)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "EastCoastStates",
"dynamodb.column.mapping" = "state_name:StateName,state_alpha:StateAbbrev");
```

次の結合により、少なくとも 3 つの特徴を持つ米国東海岸の州が返されます。

```
SELECT ecs.state_name, f.feature_class, COUNT(*)
FROM ddb_east_coast_states ecs
JOIN ddb_features f on ecs.state_alpha = f.state_alpha
GROUP BY ecs.state_name, f.feature_class
HAVING COUNT(*) >= 3;
```

異なるソースからのテーブルの結合

次の例では、s3_east_coast_states は、Amazon S3 に格納された CSV ファイルに関連付けられた Hive テーブルです。ddb_features テーブルは DynamoDB 内のデータに関連付けられています。この例では、これら 2 つのテーブルを結合し、名前が「New」で始まる州における地理的な特徴を返します。

```
create external table s3_east_coast_states (state_name STRING, state_alpha STRING)
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';
```

```
SELECT ecs.state_name, f.feature_name, f.feature_class
FROM s3_east_coast_states ecs
JOIN ddb_features f
ON ecs.state_alpha = f.state_alpha
WHERE ecs.state_name LIKE 'New%';
```

Amazon DynamoDB との間でデータをコピーします

チュートリアル:Amazon DynamoDB と Apache Hive の使用 では、ネイティブの Hive テーブルから外部にある DynamoDB テーブルにデータをコピーし、その外部 DynamoDB テーブルにクエリを実行しました。外部テーブルは Hive の外側に存在するテーブルです。仮に、DynamoDB テーブルにマッッピングされた Hive テーブルを削除したとしても、DynamoDB テーブルは影響を受けません。

DynamoDB テーブル、Amazon S3 バケット、ネイティブ Hive テーブル、および Hadoop Distributed File System (HDFS) の間でデータをコピーする場合は、Hive が優れたソリューションとなります。このセクションでは、これらのオペレーションの例を示します。

トピック

- DynamoDB とネイティブな Hive テーブル間でのデータのコピー
- DynamoDB と Amazon S3 間でのデータのコピー
- DynamoDB と HDFS 間のデータをコピー
- データ圧縮を使用する
- 表示できない UTF-8 文字データの読み込み

DynamoDB とネイティブな Hive テーブル間でのデータのコピー

DynamoDB テーブル内に存在するデータは、ネイティブの Hive テーブルにコピーできます。これにより、コピーした時点のデータのスナップショットを作成できます。

多くの HiveQL クエリを実行する必要があるものの、DynamoDB からプロビジョニングされたスループットキャパシティーを消費したくない場合などに、この操作を行います。ネイティブ Hive テーブルのデータは DynamoDB からのデータのコピーであり、その時点でのデータではないため、クエリでは、データが最新であると想定することはできません。

Note

このセクションの例は、チュートリアル:Amazon DynamoDB と Apache Hive の使用 の手順に従い、および、ddb_features という名前の外部テーブルが DynamoDB に存在することを前提にしています。

Example DynamoDB からネイティブ Hive テーブルへ

次のように、ネイティブの Hive テーブルを作成し、ddb_features からのデータを格納することができます。

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

このデータはいつでも更新できます。

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

ここでの例では、サブクエリ SELECT * FROM ddb_features により、ddb_features からのすべてのデータを取得できます。データのサブセットのみをコピーしたい場合は、このサブクエリで WHERE 句を使用できます。

次の例では、湖と山頂の一部の属性のみを含むネイティブの Hive テーブルを作成します。

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example ネイティブの Hive テーブルから DynamoDB へ

次の HiveQL ステートメントを使用して、ネイティブな Hive テーブルから ddb_features に対してデータをコピーします。

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

DynamoDB と Amazon S3 間でのデータのコピー

DynamoDB テーブルにあるデータは、Hive を使用して Amazon S3 バケットにコピーすることができます。

この操作は、DynamoDB テーブルにデータのアーカイブを作成したい場合などに行います。例えば、DynamoDB にテストデータのベースラインセットが置かれており、テスト環境でそれを操作する必要があるとします。このベースラインデータは、Amazon S3 バケットにコピーした上で、必要なテストを実行できます。その後、Amazon S3 バケットから DynamoDB にベースラインデータを復元することで、テスト環境をリセットすることができます。

チュートリアル:Amazon DynamoDB と Apache Hive の使用 の作業を行った場合、Amazon EMR ログを保存する Amazon S3 バケットが、既に作成されています。バケットのルートパスがわかっている場合は、このセクションの例のために、このバケットを利用できます。

1. Amazon EMR コンソール (https://console.aws.amazon.com/emr) を開きます。
2. [Name] (名前) で、クラスターを選択します。
3. URI が、[Configuration Details] (設定の詳細) の [Log URI] (ログの URI) に一覧表示されています。
4. バケットのルートパスを書き留めておきます。この命名規則は以下のとおりです。

   ```
   s3://aws-logs-accountID-region
   ```

   accountID は、使用している AWS アカウント ID であり、region は、バケットが置かれた AWS リージョンです。

> Note
>
> これらの例では、ここに示すようなバケット内のサブパスを使用します。
>
> `s3://aws-logs-123456789012-us-west-2/hive-test`

以下の手順は、チュートリアルの手順が正しく行われたこと、および、ddb_features という名前の外部テーブルが DynamoDB 内に存在することを前提として書かれています。

トピック

- Hive のデフォルト形式を使用したデータのコピー
- ユーザー指定の形式でデータをコピー

- 列マッピングを使用しないデータをコピー
- Amazon S3 内のデータを表示

Hive のデフォルト形式を使用したデータのコピー

Example DynamoDB から Amazon S3 へのコピー

INSERT OVERWRITE ステートメントを使用して、Amazon S3 に直接書き込みます。

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

Amazon S3 でのデータファイルは次のようになります。

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

各フィールドは SOH 文字 (0x01、ヘッダーの開始) で区切られます。ファイルでは、SOH は **^A** と表示されています。

Example Amazon S3 から DynamoDB へのコピー

1. Amazon S3 内のフォーマットされていないデータを指す外部テーブルを作成します。

```
CREATE EXTERNAL TABLE s3_features_unformatted
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. データを DynamoDB にコピーします。

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_unformatted;
```

ユーザー指定の形式でデータをコピー

フィールド区切りに独自の文字を指定する場合は、Amazon S3 バケットにマッピングされる外部テーブルを作成します。この手法は、カンマ区切り値 (CSV) のデータファイルを作成する場合などに使用できます。

Example DynamoDB から Amazon S3 へのコピー

1. Amazon S3 にマッピングされる Hive 外部テーブルを作成します。このためには、 DynamoDB 外部テーブルのデータ型と正確に一致しているデータ型を使用します。

```
CREATE EXTERNAL TABLE s3_features_csv
    (feature_id       BIGINT,
    feature_name      STRING,
    feature_class     STRING,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE,
    prim_long_dec     DOUBLE,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. DynamoDB からデータをコピーします。

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

Amazon S3 でのデータファイルは次のようになります。

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example Amazon S3 から DynamoDB へのコピー

単一の HiveQL ステートメントにより、DynamoDB テーブルに Amazon S3 からのデータを書き込むことができます。

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

列マッピングを使用しないデータをコピー

DynamoDB からのデータは、データ型や列マッピングを指定せずにそのままの形式でコピーし、Amazon S3 に書き込むことができます。この手法は、DynamoDB データのアーカイブを作成して Amazon S3 に保存する場合などに使用します。

Example DynamoDB から Amazon S3 へのコピー

1. DynamoDB テーブルに関連付けられた外部テーブルを作成します。(この HiveQL ステートメントには `dynamodb.column.mapping` はありません)。

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. Amazon S3 バケットに関連付けられた別の外部テーブルを作成します。

   ```
   CREATE EXTERNAL TABLE s3_features_no_mapping
       (item MAP<STRING, STRING>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   LINES TERMINATED BY '\n'
   LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
   ```

3. DynamoDB から Amazon S3 にデータをコピーします。

   ```
   INSERT OVERWRITE TABLE s3_features_no_mapping
   SELECT * FROM ddb_features_no_mapping;
   ```

Amazon S3 でのデータファイルは次のようになります。

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
```

```
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

各フィールドの先頭には、STX文字 (0x02、テキストの開始) が置かれ、末尾には ETX文字 (0x03、テキストの終わり) が置かれます。ファイルでは、STX は **^B** として表示され、ETX は **^C** として表示されます。

Example Amazon S3 から DynamoDB へのコピー

単一の HiveQL ステートメントにより、DynamoDB テーブルに Amazon S3 からのデータを書き込むことができます。

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

Amazon S3 内のデータを表示

SSH を使用してリーダーノードに接続している場合には、AWS Command Line Interface (AWS CLI) を使用して、Hive が Amazon S3 に書き込んだデータにアクセスできます。

以下のステップは、このセクションで示された手順のいずれかを使用して、データが DynamoDB から Amazon S3 にコピーされていることを前提として記述されています。

1. 現在 Hive コマンドプロンプトが表示されている場合は、それを終了し、Linux コマンドプロンプトを表示します。

   ```
   hive> exit;
   ```

2. Amazon S3 バケット内の hive-test ディレクトリの内容を一覧表示します。(このディレクトリには、Hive が DynamoDB からデータをコピーしています)。

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   結果は以下のようになります。

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

このファイル名 (000000_0) は、システムにより生成されています。

3. (オプション) Amazon S3 からリーダーノードのローカルファイルシステムに、データファイルをコピーできます。この操作を完了すると、標準の Linux コマンドラインユーティリティを使用して、ファイル内のデータを操作できるようになります。

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

   結果は以下のようになります。

   ```
   download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
   to ./000000_0
   ```

   > Note
   >
   > リーダーノードのローカルファイルシステムには、容量に関する制限があります。ローカルファイルシステム内で使用可能な領域より大きいサイズのファイルに対しては、このコマンドを使用しないでください。

DynamoDB と HDFS 間のデータをコピー

DynamoDB テーブルにあるデータであれば、Hive を使用して Hadoop Distributed File System (HDFS) にコピーすることができます。

DynamoDB からのデータを必要とする MapReduce ジョブを実行している場合などに、この操作を行います。DynamoDB から HDFS にデータをコピーする際には、Amazon EMR クラスターで使用可能なすべてのノードを並行して使用しながら、Hadoop がそのデータの処理を行います。MapReduce ジョブの完了時、その結果を HDFSからDDBに書き込むことができます。

次の例では、Hive が、ここに示す HDFS ディレクトリ `/user/hadoop/hive-test` に対する読み書きを実行しています。

> Note
>
> このセクションの例は、チュートリアル:Amazon DynamoDB と Apache Hive の使用 の手順が正確に完了していること、および、ddb_features という名前の外部テーブルが DynamoDB に存在することを前提に書かれています。

トピック

- Hive のデフォルト形式を使用したデータのコピー
- ユーザー指定の形式でデータをコピー
- 列マッピングを使用しないデータをコピー
- HDFS 内のデータへのアクセス

Hive のデフォルト形式を使用したデータのコピー

Example DynamoDB から HDFS へのコピー

`INSERT OVERWRITE` ステートメントを使用することで、HDFS に直接書き込みます。

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

HDFS 内に置かれるデータファイルは次のようになります。

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

各フィールドは SOH 文字 (0x01、ヘッダーの開始) で区切られます。ファイルでは、SOH は **^A** と表示されています。

Example HDFS から DynamoDB へのコピー

1. HDFS 内のフォーマットされていないデータにマッピングする外部テーブルを作成します。

```
CREATE EXTERNAL TABLE hdfs_features_unformatted
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
```

```
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. データを DynamoDB にコピーします。

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_unformatted;
```

ユーザー指定の形式でデータをコピー

異なるフィールド区切り文字を使用する場合は、外部テーブルを作成し、そのテーブルをHDFS ディレクトリにマッピングします。この手法は、カンマ区切り値 (CSV) のデータファイルを作成する場合などに使用できます。

Example DynamoDB から HDFS へのコピー

1. HDFS にマッピングする Hive 外部テーブルを作成します。このためには、 DynamoDB 外部テーブルのデータ型と正確に一致しているデータ型を使用します。

```
CREATE EXTERNAL TABLE hdfs_features_csv
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. DynamoDB からデータをコピーします。

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

HDFS 内に置かれるデータファイルは次のようになります。

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
```

```
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example HDFS から DynamoDB へのコピー

単一の HiveQL ステートメントにより、DynamoDB テーブルに HDFS からのデータを書き込むことができます。

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

列マッピングを使用しないデータをコピー

DynamoDB からのデータは、データ型や列マッピングを指定せずにそのままの形式でコピーし、HDFS に書き込むことができます。この手法は、DynamoDB データのアーカイブを作成し、HDFS に保存する場合などに使用できます。

> **Note**
>
> DynamoDB テーブルに Map、List、Boolean、または Null 型の属性が含まれている場合、Hive を使用して DynamoDB から HDFS にデータをコピーするには、これが唯一の方法となります。

Example DynamoDB から HDFS へのコピー

1. DynamoDB テーブルに関連付けられた外部テーブルを作成します。(この HiveQL ステートメントには `dynamodb.column.mapping` はありません)。

   ```
   CREATE EXTERNAL TABLE ddb_features_no_mapping
       (item MAP<STRING, STRING>)
   STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
   TBLPROPERTIES ("dynamodb.table.name" = "Features");
   ```

2. HDFS ディレクトリに関連付けられた別の外部テーブルを作成します。

   ```
   CREATE EXTERNAL TABLE hdfs_features_no_mapping
       (item MAP<STRING, STRING>)
   ```

```
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION 'hdfs:///user/hadoop/hive-test';
```

3. DynamoDB から HDFS にデータをコピーします。

   ```
   INSERT OVERWRITE TABLE hdfs_features_no_mapping
   SELECT * FROM ddb_features_no_mapping;
   ```

HDFS 内に置かれるデータファイルは次のようになります。

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

各フィールドの先頭には、STX文字 (0x02、テキストの開始) が置かれ、末尾には ETX文字 (0x03、テキストの終わり) が置かれます。ファイルでは、STX は **^B** として表示され、ETX は **^C** として表示されます。

Example HDFS から DynamoDB へのコピー

単一の HiveQL ステートメントにより、DynamoDB テーブルに HDFS からのデータを書き込むことができます。

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

HDFS 内のデータへのアクセス

HDFS は分散ファイルシステムであり、Amazon EMR クラスター内のすべてのノードからアクセス可能です。SSH を使用してリーダーノードに接続することで、Hive が HDFS に書き込んだデータに対し、コマンドラインツールを使用してのアクセスが可能になります。

HDFSは、リーダーノード上のローカルなファイルシステムとは異なります。標準の Linux コマンド (cat、cp、mv、または rm) を使用して、HDFS 内のファイルやディレクトリはを操作することはできません。代わりに hadoop fs コマンドを使用して、これらのタスクを実行します。

以下のステップは、このセクションで示された手順のいずれかを使用して、データが DynamoDB から HDFS にコピーされていることを前提として記述されています。

1. 現在 Hive コマンドプロンプトが表示されている場合は、それを終了し、Linux コマンドプロンプトを表示します。

   ```
   hive> exit;
   ```

2. HDFS内の、/user/hadoop/hive-test ディレクトリの内容を一覧表示します。(このディレクトリには、Hive が DynamoDB からデータをコピーしています)。

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   結果は以下のようになります。

   ```
   Found 1 items
   -rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
   ```

   このファイル名 (000000_0) は、システムにより生成されています。

3. ファイルのコンテンツを表示します。

   ```
   hadoop fs -cat /user/hadoop/hive-test/000000_0
   ```

   > Note
   >
   > この例では、比較的小さい (約 29 KB) ファイルを使用しています。非常に大きなファイルや、表示不可能な文字を含むファイルに対して、ここでのコマンドを使用する際には注意が必要です。

4. (オプション) データファイルは、HDFS からリーダーノード上のローカルファイルシステムにコピーすることができます。そうすることで、標準の Linux コマンドラインユーティリティを通じて、ファイル内のデータを使用できるようになります。

   ```
   hadoop fs -get /user/hadoop/hive-test/000000_0
   ```

このコマンドでは、ファイルの上書きは行われません。

> Note
>
> リーダーノードのローカルファイルシステムには、容量に関する制限があります。ローカルファイルシステム内で使用可能な領域より大きいサイズのファイルに対しては、このコマンドを使用しないでください。

データ圧縮を使用する

Hive を使用して異なるデータソース間でデータをコピーする際には、オンザフライのデータ圧縮を要求できます。Hive では、いくつかの圧縮コーデックが利用可能です。その中の 1 つを Hive セッション中に選択します。これにより、データは指定した形式で圧縮されます。

次の例では、LZO(Lempel-Ziv-Oberhumer)アルゴリズムを使用してデータを圧縮します。

```
SET hive.exec.compress.output=true;
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
FROM hiveTableName;
```

Amazon S3 内に出力されるファイルは、その末尾には `.lzo` (例えば、`8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`) が付いたシステム生成の名前が付けられます。

以下の圧縮コーデックを利用できます。

- `org.apache.hadoop.io.compress.GzipCodec`
- `org.apache.hadoop.io.compress.DefaultCodec`
- `com.hadoop.compression.lzo.LzoCodec`
- `com.hadoop.compression.lzo.LzopCodec`

- org.apache.hadoop.io.compress.BZip2Codec
- org.apache.hadoop.io.compress.SnappyCodec

表示できない UTF-8 文字データの読み込み

表示できない UTF-8 文字データの読み込み、および書き込みを行うには、Hive テーブル作成時に STORED AS SEQUENCEFILE 句を使用します。SequenceFile は、Hadoop バイナリファイル形式で作成されています。このファイルを読み取るには Hadoop を使用する必要があります。次の例に、DynamoDB から Amazon S3 にデータをエクスポートする方法を示します。この機能を使用して印刷不可の UTF-8 でエンコードされた文字を処理できます。

```
CREATE EXTERNAL TABLE s3_export(a_col string, b_col bigint, c_col array<string>)
STORED AS SEQUENCEFILE
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3_export SELECT *
FROM hiveTableName;
```

パフォーマンスチューニング

DynamoDB テーブルにマッピングする Hive 外部テーブルを作成しても、DynamoDB からの読み込みまたは書き込み容量は消費されません。ただし、Hive テーブルでの読み込み/書き込みのアクティビティ (INSERT または SELECT など) は、基盤となっている DynamoDB テーブルでの読み込み/書き込みオペレーションとして、直接反映されます。

Amazon EMR の Apache Hive では、DynamoDB テーブルにおける I/O 負荷を分散するための独自のロジックが実装されています。また、テーブルでプロビジョニングされたスループットの超過を、最小限に抑制することが試みられます。Amazon EMR は各 Hive クエリの終了時点で、プロビジョニング済みのスループットが超過された回数を含め、実行時のメトリクスを返します。この情報を DynamoDB テーブルの CloudWatch メトリックスと共に使用することで、後続のリクエストのパフォーマンスを向上することにつなげられます。

Amazon EMR コンソールには、クラスターのモニタリングに関する基本的なツールが用意されています。詳細については、「Amazon EMR 管理ガイド」の「クラスターを表示し、モニタリングする」を参照してください。

また、クラスターと Hadoop のジョブをモニタリングするために、Hue、Ganglia、Hadoop ウェブインターフェイスなどのウェブベースのツールを使用することもできます。詳細については、

「Amazon EMR 管理ガイド」の「Amazon EMR クラスターでホストされているウェブインターフェイスの表示」を参照してください。

このセクションでは、外部 DynamoDB テーブルで実行される Hive オペレーションのパフォーマンスを、調整する際に使用できる手順について説明します。

トピック

- DynamoDB のプロビジョニングされたスループット
- マッパーの調整
- その他のトピック

DynamoDB のプロビジョニングされたスループット

外部 DynamoDB テーブルに対して HiveQL ステートメントを発行すると、`DynamoDBStorageHandler` クラスにより、適切な低レベルの DynamoDB API リクエストが生成されます。このリクエストは、プロビジョニングされたスループットを消費します。十分な読み込みまたは書き込み容量が DynamoDB テーブルにない場合、リクエストはスロットリングされ、HiveQL のパフォーマンス低下を引き起こします。このためテーブルには、十分なスループットキャパシティーを確保する必要があります。

例えば、DynamoDB テーブルのために 100 個の読み込みをプロビジョニングしているとします。毎秒読み込むことができるのは、409,600 バイト (100×4 KBの読み込みキャパシティーユニットサイズ) です。ここで、このテーブルに、20 GB (21,474,836,480 バイト) のデータが含まれており、HiveQL を使ってすべてのデータを選択する `SELECT` ステートメントを使いたいとします。この場合、次のようにクエリの実行にかかる時間を見積もることができます。

21,474,836,480 / 409,600 = 52,429 秒 = 14.56 時間

このシナリオでは、DynamoDB テーブルがボトルネックになります。Hive のスループットが 1 秒あたり 409,600 バイトに制限されているため、Amazon EMR ノードを追加することはできません。`SELECT` ステートメントの処理に必要な時間を短縮する唯一の方法は、DynamoDB テーブルのプロビジョニングされた読み込みキャパシティを増やすことです。

同様の計算により、DynamoDB テーブルにマッピングされた Hive 外部テーブルに、データを一括ロードする際に必要となる時間も推定できます。項目ごとに必要な書き込みキャパシティーユニットの合計数 (1KB = 1 未満、1～2KB = 2 など) を決定し、それにロードする項目数を掛けます。これにより、必要な書き込みキャパシティーユニットの数がわかります。その数を、1 秒あたりに割り当て

られる書き込みキャパシティーユニットの数で割ります。これにより、テーブルのロードに要する秒数が計算できます。

テーブルの CloudWatch メトリクスは、定期的にモニタリングする必要があります。DynamoDB コンソールでその概要を簡単に確認するには、対象のテーブルを選択し、[Metrics] (メトリクス) タブを開きます。消費された読み込み/書き込みキャパシティーユニット数、および、スロットリングされた読み込み/書き込みリクエスト数を表示できます。

読み込みキャパシティ

Amazon EMR は、テーブルでのプロビジョニング済みスループットの設定に応じて、DynamoDB テーブルに対するリクエスト負荷を管理します。しかし、ジョブ出力に多数の ProvisionedThroughputExceeded メッセージが見られる場合には、デフォルトの読み込みレートを調整することも可能です。このためには、dynamodb.throughput.read.percent 設定変数を変更できます。Hive コマンドプロンプトで SET コマンドを使用して、この変数を設定できます。

```
SET dynamodb.throughput.read.percent=1.0;
```

この変数は、その時点の Hive セッションの間のみ有効です。一度 Hive を終了し、再び再開した場合には、dynamodb.throughput.read.percent はデフォルト値に戻ります。

dynamodb.throughput.read.percent の値は 0.1 と 1.5 の間で設定できます。0.5 は、デフォルトの読み込みレートを表し、Hive がテーブルの読み込みキャパシティの半分を消費しようとすることを意味します。上記の値を 0.5 より上に設定すると、Hive は読み込みリクエストレートを増加させ、0.5 より低くした場合はリクエストレートを減少させます。(実際の読み込みレートは、DynamoDB テーブルに、統一ディストリビューションのキーがあるかどうかなどの要因によって変わります。)

テーブルのプロビジョニングされた読み込みキャパシティを、Hive が頻繁に枯渇させている場合、または読み込みリクエストのスロットリング頻度が高すぎる場合は、dynamodb.throughput.read.percent を 0.5 より低く設定してみてください。テーブルに十分な読み込みキャパシティがある場合で、HiveQL オペレーションにさらに高い応答性を求める場合には、この値を 0.5 より高く設定できます。

書き込みキャパシティ

Amazon EMR は、テーブルでのプロビジョニング済みスループットの設定に応じて、DynamoDB テーブルに対するリクエスト負荷を管理します。しかし、ジョブ出力に多数の ProvisionedThroughputExceeded メッセージが見られる場合には、デフォルトの書き込みレートを調整することも可能です。このためには、dynamodb.throughput.write.percent 設定変

数を変更できます。Hive コマンドプロンプトで SET コマンドを使用して、この変数を設定できます。

```
SET dynamodb.throughput.write.percent=1.0;
```

この変数は、その時点の Hive セッションの間のみ有効です。一度 Hive を終了し、再び再開した場合には、`dynamodb.throughput.write.percent` はデフォルト値に戻ります。

`dynamodb.throughput.write.percent` の値は `0.1` と `1.5` の間で設定できます。`0.5` は、デフォルトの書き込みレートを表し、Hive がテーブルの書き込みキャパシティの半分を消費しようとすることを意味します。上記の値を `0.5` より上に設定すると、Hive は書き込みリクエストレートを増加させ、`0.5` より低くした場合はリクエストレートを減少させます。(実際の書き込みレートは、DynamoDB テーブルに、統一ディストリビューションのキーがあるかどうかなどの要因によって変わります。)

テーブルのプロビジョニングされた書き込みキャパシティを、Hive が頻繁に枯渇させている場合、または書き込みリクエストのスロットリング頻度が高すぎる場合は、`dynamodb.throughput.write.percent` を `0.5` より低く設定してみてください。テーブルに十分なキャパシティがある場合で、HiveQL オペレーションにさらに高い応答性を求める場合には、この値を `0.5` より高く設定できます。

Hive を使用してデータを DynamoDB に書き込む場合は、書き込みキャパシティーユニットの数をクラスター内のマッパーの数より大きくする必要があります。例えば、Amazon EMR クラスターが 10 の m1.xlarge ノードで構成されているとします。m1.xlarge ノードタイプでは、8 のマッパータスクが提供されるため、クラスターには合計 80 のマッパー (10 × 8) が存在することになります。DynamoDB テーブルにある書き込みキャパシティーユニットが 80 個未満の場合、Hive による書き込みオペレーションが、そのテーブルの書き込みスループットをすべて消費してしまう可能性があります。

Amazon EMR での各ノードタイプのマッパーの数については、「Amazon EMR デベロッパーガイド」の「タスクの設定」を参照してください。

マッパーの詳細については、「マッパーの調整」を参照してください。

マッパーの調整

Hive が Hadoop ジョブを起動すると、このジョブは 1 つ以上のマッパータスクによって処理されます。DynamoDB テーブルに十分なスループットキャパシティが用意されているのであれば、クラスター内のマッパーの数が変更可能なので、これによりパフォーマンスを向上させられる可能性があります。

> Note
>
> Hadoop ジョブで使用されるマッパータスクの数は、Hadoop がデータを論理ブロックに分割する入力分割による影響を受けます。Hadoop により十分な入力分割が行われない場合、書き込みオペレーションが、DynamoDB テーブルで使用可能なすべての書き込みスループットを消費できないことがあります。

マッパーの数を増やします

Amazon EMR の各マッパーの最大読み込みレートは、1 秒あたり 1 MiB です。クラスター内のマッパーの数は、そのクラスター内のノードのサイズによって異なります。(ノードのサイズとノードあたりのマッパー数については、「Amazon EMR デベロッパーガイド」の「タスクの設定」を参照してください。)

DynamoDB テーブルに十分な読み込み用スループット容量がある場合は、次のいずれかの方法でマッパーの数を増やすことができます。

- クラスター内のノードのサイズを大きくする。たとえば、クラスターで (ノードごとに 3 つのマッパー) m1.large ノードを使用しているのであれば、これを (ノードごとに 8 つのマッパー) m1.xlarge ノードにアップグレードしてみることができます。
- クラスター内のノード数を増やす。たとえば、3 つの m1.xlarge ノードを持つクラスターであれば、合計 24 個のマッパーを使用できます。同じタイプのノードでクラスターのサイズを 2 倍にすれば、48 個のマッパーが使用可能になります。

AWS Management Console マネジメントコンソールを使用して、クラスター内のノードのサイズまたは数を管理することができます。(これらの変更を有効にするには、クラスターの再起動が必要な場合があります。)

マッパーの数を増やすもう 1 つの方法は、`mapred.tasktracker.map.tasks.maximum` Hadoop の設定パラメータを変更することです。(これは Hadoop パラメータであり、Hive パラメータではないため、コマンドプロンプトからインタラクティブに変更することはできません。) `mapred.tasktracker.map.tasks.maximum` の値を増加することで、ノードのサイズや数を増やすことなく、マッパーの数を増やすことができます。ただし、この値を高く設定しすぎると、クラスターノードのメモリが不足する可能性があります。

`mapred.tasktracker.map.tasks.maximum` の値は、Amazon EMR クラスターを初めて起動する際のブートストラップアクションの中で設定します。詳細については、「Amazon EMR 管理ガイ

ド」の「(オプション) 追加のソフトウェアをインストールするためのブートストラップアクションの作成」を参照してください。

マッパーの数を減らします

SELECT ステートメントを使用して DynamoDB にマッピングする外部 Hive テーブルからデータを選択する場合、Hadoop ジョブはクラスター内のマッパーの最大数まで、必要な数のタスクを使用できます。このシナリオでは、長時間実行される Hive クエリが DynamoDB テーブルのプロビジョニングされた読み込みキャパシティーをすべて消費することで、他のユーザーに悪影響を与える可能性があります。

`dynamodb.max.map.tasks` パラメータを使用して、マップタスクの上限を設定することができます。

```
SET dynamodb.max.map.tasks=1
```

この値は、1 以上にする必要があります。Hive がクエリを処理する際に実行される Hadoop ジョブは、DynamoDB テーブルからの読み込み時、`dynamodb.max.map.tasks` を超えるタスクを使用することはありません。

その他のトピック

Hive を使用して DynamoDB にアクセスするアプリケーションを調整するための、他の方法を以下に示します。

再試行の期間

デフォルトでは、2 分以内に DynamoDB からの結果が返されない場合、Hive は Hadoop ジョブを再実行します。この間隔は、以下のように `dynamodb.retry.duration` パラメータを変更することで調整できます。

```
SET dynamodb.retry.duration=2;
```

この値は、再試行間隔 (分) を表す 0 以外の整数である必要があります。`dynamodb.retry.duration` のデフォルトは 2 (分) です。

並列データリクエスト

複数のユーザーまたは複数のアプリケーションから、単一のテーブルに複数のデータリクエストが行われると、プロビジョニング済み読み込みスループットの枯渇につながり、パフォーマンス速度を低下させることがあります。

処理間隔

DynamoDB におけるデータの整合性は、各ノードで実行される読み込み/書き込みオペレーションの順序に影響を受けます。Hive クエリの進行中に、別のアプリケーションが DynamoDB テーブルに新しいデータをロードしたり、既存のデータの変更や削除を行ったりする場合があります。この場合、クエリの実行中にデータに対して行われた変更は Hive クエリの結果に反映されないことがあります。

[Request time] (リクエストタイム)

DynamoDB テーブルの需要が低いタイミングでテーブルにアクセスするように、Hive クエリをスケジューリングすると、パフォーマンスを向上させられます。例えば、アプリケーションのほとんどのユーザーがサンフランシスコに在住の場合、大部分のユーザーが就寝して DynamoDB データベースのレコードを更新することがない午前 4 時 (PST) に、毎日のデータエクスポートを行います。

Amazon S3 との統合

Amazon DynamoDB のインポートおよびエクスポート機能は、コードを記述せずに Amazon S3 と DynamoDB のテーブル間でデータを移動するシンプルで効率的な方法を提供します。

DynamoDB のインポートおよびエクスポート機能は、DynamoDB テーブルアカウントの移動、変換、コピーに役立ちます。S3 ソースからのインポートや、DynamoDB テーブルデータの Amazon S3 へのエクスポートにより、Athena、Amazon SageMaker、AWS Lake Formation などの AWS のサービスを使用してデータを分析し、実用的なインサイトを抽出できます。また、データを新しい DynamoDB テーブルに直接インポートして、大規模な 1 桁ミリ秒のパフォーマンスで新しいアプリケーションを構築し、テーブルとアカウント間のデータ共有を容易にして、ディザスタリカバリとビジネス継続性計画を簡素化することもできます。

トピック

- Amazon S3 からの DynamoDB データのインポート: 仕組み
- Amazon S3 への DynamoDB データのエクスポート: 仕組み

Amazon S3 からの DynamoDB データのインポート: 仕組み

データを DynamoDB にインポートするには、データが CSV、DynamoDB JSON、または Amazon Ion 形式で Amazon S3 バケット内にある必要があります。データは ZSTD または GZIP 形式で圧縮

することも、非圧縮形式で直接インポートすることもできます。ソースデータは、単一の Amazon S3 オブジェクトでも、同じプレフィックスを使用する複数の Amazon S3 オブジェクトでもかまいません。

データは新しい DynamoDB テーブルにインポートされ、インポートリクエストを開始すると作成されます。セカンダリインデックスを使用してこのテーブルを作成し、インポートが完了したらすぐに、すべてのプライマリインデックスとセカンダリインデックスでデータのクエリと更新を行うことができます。インポートした後にグローバルテーブルレプリカを追加することもできます。

> Note
>
> Amazon S3 のインポートプロセス中に、DynamoDB はインポートされる新しいターゲットテーブルを作成します。既存のテーブルへのインポートは、現在この機能ではサポートされていません。

Amazon S3 からのインポートでは、新しいテーブルの書き込み容量が消費されないため、データを DynamoDB にインポートするために追加の容量をプロビジョニングする必要はありません。データインポートの料金は、インポートの結果として処理される Amazon S3 のソースデータの非圧縮サイズに基づいています。処理されたが、ソースデータの形式やその他の不整合のためにテーブルに読み込めなかった項目も、インポートプロセスの一部として請求されます。詳細については、「Amazon DynamoDB の料金表」を参照してください。

別のアカウントによって所有されている Amazon S3 バケットからデータをインポートするには、そのバケットから読み取るための正しい許可が必要です。新しいテーブルは、ソース Amazon S3 バケットとは異なるリージョンにある場合もあります。手順については、「Amazon Simple Storage Service の設定とアクセス許可」を参照してください。

インポート時間は、Amazon S3 におけるデータの特性に直接関係します。これには、データサイズ、データ形式、圧縮スキーム、データ分散の均一性、Amazon S3 オブジェクトの数、およびその他の関連変数が含まれます。特に、均一に分散されたキーを持つデータセットは、歪んだデータセットよりもインポートが高速になります。例えば、セカンダリインデックスのキーが月をパーティショニングに使用し、すべてのデータが 12 月のものだった場合、このデータのインポートにはかなり時間がかかることがあります。

キーに関連付けられた属性は、ベーステーブル上で一意であることが想定されます。一意でないキーがある場合、インポートによって、最後の上書きだけが残るまで、関連する項目が上書きされます。例えば、プライマリキーが月で、複数の項目が 9 月に設定されている場合、新しい項目はそれぞれ以前に書き込まれた項目を上書きし、「月」のプライマリキーが 9 月に設定された 1 つの項目だけ

が残ります。このような場合、インポートテーブルの説明で処理される項目の数は、ターゲットテーブルの項目数と一致しません。

テーブルインポートのコンソールと API アクションは、すべて AWS CloudTrail に記録されます。詳細については、「AWS CloudTrail を使用して DynamoDB オペレーションをログに記録する」を参照してください。

次の動画では、Amazon S3 から DynamoDB に直接インポートする方法について紹介します。

Amazon S3 からのインポート

トピック

DynamoDB でテーブルのインポートをリクエストする

DynamoDB インポートでは、Amazon S3 バケットから新しい DynamoDB テーブルにデータをインポートできます。テーブルのインポートをリクエストするには、DynamoDB コンソール、CLI、CloudFormation、または DynamoDB を使用できます。

AWS CLI を使用する場合は、最初に設定する必要があります。詳細については、「DynamoDB にアクセスする」を参照してください。

> Note
>
> - テーブルのインポート機能は、複数の異なる AWS のサービス (Amazon S3 や CloudWatch など) と連携して動作します。インポートを開始する前に、インポート API を呼び出すユーザーまたはロールに、機能が依存するすべてのサービスとリソースに対するアクセス許可があることを確認します。
> - インポートの進行中に Amazon S3 オブジェクトを変更しないでください。オペレーションが失敗したり、キャンセルされたりする可能性があります。
>
> エラーとトラブルシューティングの詳細については、「インポート形式のクォータと検証」を参照してください。

トピック

IAM アクセス許可のセットアップ

データは、読み取りアクセス許可を持っている Amazon S3 バケットであれば、どのバケットからでもインポートを実行できます。インポート元のバケットは、ソーステーブルと同じリージョンに存在する必要はなく、所有者が同じである必要もありません。AWS Identity and Access Management (IAM) には、ソース Amazon S3 バケットの関連アクションと、デバッグ情報を提供するために必要な CloudWatch アクセス許可を含める必要があります。次にポリシーの例を示します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBImportAction",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ImportTable",
                "dynamodb:DescribeImport",
                "dynamodb:ListImports"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table*"
        },
        {
            "Sid": "AllowS3Access",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
               "arn:aws:s3:::your-bucket/*",
               "arn:aws:s3:::your-bucket"
            ]
        },
        {
            "Sid": "AllowCloudwatchAccess",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:/aws-dynamodb/*"
        }
    ]
}
```

Amazon S3 のアクセス許可

別のアカウントが所有する Amazon S3 バケットソースでインポートを開始するときは、ロールまたはユーザーが Amazon S3 オブジェクトにアクセスできることを確認します。Amazon S3 の `GetObject` コマンドを実行し、認証情報を使用することで、これを確認できます。API を使用する場合、Amazon S3 バケット所有者パラメータは、デフォルトで現在のユーザーのアカウント ID に設定されます。クロスアカウントインポートの場合は、このパラメータにバケット所有者のアカウント ID が正しく入力されていることを確認します。次のコードは、ソースアカウントの Amazon S3 バケットポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {"Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::awsexamplebucket1/*"
        }
    ]
}
```

AWS Key Management Service

インポート用の新しいテーブルを作成するときに、DynamoDB が所有していない保管時の暗号化キーを選択した場合は、カスタマーマネージドキーで暗号化された DynamoDB テーブルを操作するために必要なアクセス許可を AWS KMS に付与する必要があります。詳細については、「AWS KMS キーの使用を認可する」を参照してください。Amazon S3 オブジェクトがサーバー側の暗号化で暗号化されている場合は、インポートを開始するロールまたはユーザーが、AWS KMS キーを使用して復号するアクセス権を持っていることを確認してください。この機能は、お客様が指定したキーによるサーバー側の暗号化 (SSE-C) で暗号化された Amazon S3 オブジェクトはサポートしません。

CloudWatch のアクセス許可

インポートを開始するロールまたはユーザーには、インポートに関連付けられたロググループとログストリームに対する作成と管理のアクセス許可が必要です。

AWS Management Console を使用してインポートをリクエストする

DynamoDB コンソールを使用して、`MusicCollection` という新しいテーブルに既存のデータをインポートする方法を次の例に示します。

テーブルのインポートをリクエストするには

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[Import from S3] (S3 からインポート) を選択します。
3. 表示されたページで、[Import from S3] (S3 からインポート) を選択します。
4. [Import from S3] (S3 からインポート) を選択します。
5. [ソース S3 の URL] に Amazon S3 のソース URL を入力します。

   ソースバケットを所有している場合は、[S3 を参照] を選択してバケットを検索します。または、バケットの URL を `s3://bucket/prefix` の形式で入力します。`prefix` は Amazon S3 キープレフィックスです。これは、インポートする Amazon S3 オブジェクトの名前であるか、インポートするすべての Amazon S3 オブジェクトが共有するキープレフィックスのいずれかです。

> **Note**
>
> DynamoDB エクスポートリクエストと同じプレフィックスを使用することはできません。エクスポート機能は、すべてのエクスポートのフォルダ構造とマニフェストファイルを作成します。同じ Amazon S3 パスを使用すると、エラーが発生します。
> 代わりに、特定のエクスポートからのデータを含むフォルダにインポートを指定します。この場合、正しいパスの形式は `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/Data/` になります。ここで、`XXXXXXXX-XXXXXX` はエクスポート ID です。エクスポート ID はエクスポート ARN にあり、形式は `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>` です。例えば、`arn:aws:dynamodb:`*`us-east-1`*`:`*`123456789012`*`:table/`*`ProductCatalog`*`/export/`*`01234567890123-a1b2c3d4`* と指定します。

6. 自分が [S3 バケット所有] であることを確認します。ソースバケットが別のアカウントによって所有されている場合は、[別の AWS アカウント] を選択します。次に、バケット所有者のアカウント ID を入力します。

7. [ファイル圧縮をインポート] で、[圧縮なし]、[GZIP]、または [ZSTD] を適宜選択します。

8. 適切なインポートファイルフォーマットを選択します。オプションは、[DynamoDB JSON]、[Amazon Ion]、または [CSV] です。[CSV] を選択する場合は、[CSV ヘッダー] および [CSV 区切り文字] の 2 つの追加オプションがあります。

   [CSV ヘッダー] を使用する場合は、ヘッダーをファイルの最初の行から取得するか、カスタマイズするかを選択します。[Customize your headers] (ヘッダーをカスタマイズ) を選択した場合は、インポートに使用するヘッダーの値を指定できます。この方法で指定される CSV ヘッダーは大文字と小文字が区別され、ターゲットテーブルのキーを含むことが想定されます。

   [CSV 区切り文字] では、項目を区切る文字を設定します。デフォルトではカンマが選択されます。[カスタム区切り文字] を選択した場合、区切り文字は正規表現パターン `[,;:|\t ]` と一致する必要があります。

9. [次へ] ボタンを選択し、データを保存するために作成される新しいテーブルのオプションを選択します。

> Note
>
> プライマリキーとソートキーはファイル内の属性と一致する必要があります。一致しない場合、インポートは失敗します。属性では、大文字と小文字が区別されます。

10. [次へ] をもう一度選択してインポートオプションを確認し、[インポート] を選択してインポートタスクを開始します。まず、新しいテーブルが「作成中」のステータスで「テーブル」にリストされます。この時点では、テーブルにアクセスできません。
11. インポートが完了すると、ステータスが「アクティブ」と表示され、テーブルの使用を開始できます。

AWS Management Consoleで過去のインポートの詳細を取得する

過去に実行したインポートタスクに関する情報は、ナビゲーションサイドバーの [Import from S3] (S3 からインポート) を選択し、[インポート] タブを選択して表示できます。インポートパネルには、過去 90 日間に作成されたすべてのインポートのリストが表示されます。[インポート] タブにリストされているタスクの ARN を選択すると、選択した詳細設定など、そのインポートに関する情報が取得されます。

AWS CLI を使用してインポートをリクエストする

次の例では、プレフィックスのプレフィックスが bucket という名前の S3 バケットから、target-table という新しいテーブルに CSV 形式のデータをインポートします。

```
aws dynamodb import-table --s3-bucket-source S3Bucket=bucket,S3KeyPrefix=prefix \
            --input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"hk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"hk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}' \
            --input-format-options '{"Csv": {"HeaderList": ["hk", "title", "artist",
 "year_of_release"], "Delimiter": ";"}}'
```

> Note
>
> AWS Key Management Service (AWS KMS) によって保護されたキーを使用してインポートを暗号化することを選択した場合、そのキーはインポート先 Amazon S3 バケットと同じリージョンにある必要があります。

AWS CLIで過去のインポートの詳細を取得する

過去に実行したインポートタスクに関する情報は、`list-imports` コマンドを実行して見つけることができます。このコマンドは、過去 90 日間に作成されたすべてのインポートのリストを返します。インポートタスクのメタデータは 90 日後に期限切れになり、それより古いジョブはこのリストに表示されなくなりますが、Amazon S3 バケット内のオブジェクトまたはインポート中に作成されたテーブルは削除されません。

```
aws dynamodb list-imports
```

詳細な設定など、特定のインポートタスクに関する詳細情報を取得するには、`describe-import` コマンドを使用します。

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

DynamoDB 用の Amazon S3 のインポート形式

DynamoDB は、CSV、DynamoDB JSON、および Amazon Ion の 3 つの形式でデータをインポートできます。

トピック

- CSV
- DynamoDB Json
- Amazon Ion

CSV

CSV 形式のファイルは、改行で区切られた複数の項目で構成されます。デフォルトでは、DynamoDB はインポートファイルの最初の行をヘッダーとして解釈し、列がカンマで区切られることを想定しています。ファイル内の列数と一致する限り、適用されるヘッダーを定義することもできます。ヘッダーを明示的に定義すると、ファイルの最初の行が値としてインポートされます。

> Note
>
> CSV ファイルからインポートする場合、ベーステーブルとセカンダリインデックスのハッシュ範囲とキー以外のすべての列が DynamoDB 文字列としてインポートされます。

二重引用符のエスケープ

CSV ファイルに存在する二重引用符文字はすべてエスケープする必要があります。エスケープしないと、次の例に示すように、インポートは失敗します。

```
id,value
"123",Women's Full Lenth Dress
```

引用符を 2 組の二重引用符でエスケープすると、この同じインポートが成功します。

```
id,value
"""123""",Women's Full Lenth Dress
```

テキストを適切にエスケープしてインポートすると、元の CSV ファイルと同じように表示されます。

```
id,value
"123",Women's Full Lenth Dress
```

DynamoDB Json

DynamoDB JSON 形式のファイルは、複数の Item オブジェクトで構成できます。個々のオブジェクトは DynamoDB のスタンダードマーシャリングされた JSON 形式で、改行が項目区切り文字として使用されます。追加機能として、特定の時点からのエクスポートは、デフォルトでインポートソースとしてサポートされています。

> Note
>
> 新しい行は DynamoDB JSON 形式のファイルの項目区切り文字として使用されるため、項目オブジェクト内では使用しないでください。

```
{
    "Item": {
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "333-3333333333"
        },
        "Id": {
            "N": "103"
        },
        "InPublication": {
            "BOOL": false
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 103 Title"
        }
    }
}
```

> **Note**
>
> 新しい行は DynamoDB JSON 形式のファイルの項目区切り文字として使用されるため、項目オブジェクト内では使用しないでください。

```
{
    "Item": {
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "333-3333333333"
        },
        "Id": {
            "N": "103"
        },
        "InPublication": {
            "BOOL": false
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 103 Title"
        }
    }
} {
    "Item": {
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "444-444444444"
        },
        "Id": {
            "N": "104"
        },
        "InPublication": {
            "BOOL": false
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 104 Title"
        }
    }
} {
    "Item": {
        "Authors": {
            "SS": ["Author1", "Author2"]
        },
        "Dimensions": {
            "S": "8.5 x 11.0 x 1.5"
        },
        "ISBN": {
            "S": "555-5555555555"
        },
        "Id": {
            "N": "105"
        },
        "InPublication": {
            "BOOL": false
        },
        "PageCount": {
            "N": "600"
        },
        "Price": {
            "N": "2000"
        },
        "ProductCategory": {
            "S": "Book"
        },
        "Title": {
            "S": "Book 105 Title"
        }
    }
}
```

Amazon Ion

Amazon Ion は、大規模なサービス指向アーキテクチャをエンジニアリングしながら、急速な開発、デカップリング、効率性といった日々の課題に対処するために構築されたリッチタイプの自己記述型階層型データシリアル化フォーマットです。

データを Ion 形式でインポートすると、Ion データ型は新しい DynamoDB テーブルの DynamoDB データ型にマップされます。

	Ion から DynamoDB へのデータ型変換	B
1	`Ion Data Type`	`DynamoDB Representation`
2	`string`	`String (s)`
3	`bool`	`Boolean (BOOL)`
4	`decimal`	`Number (N)`
5	`blob`	`Binary (B)`
6	`list (with type annotation $dynamodb_SS, $dynamodb_NS, or $dynamodb_BS)`	`Set (SS, NS, BS)`
7	`list`	`List`
8	`struct`	`Map`

Ion ファイル内の項目は、改行で区切られます。各行は Ion バージョンマーカーで始まり、Ion 形式の項目が続きます。

> **Note**
>
> 次の例では、読みやすくするために 1 つの Ion 形式ファイルの項目が複数の行にフォーマットされています。

```
$ion_1_0 {
    Item:{
        Authors:$dynamodb_SS::["Author1","Author2"],
        Dimensions:"8.5 x 11.0 x 1.5",
        ISBN:"333-3333333333",
        Id:103.,
        InPublication:false,
        PageCount:6d2,
        Price:2d3,
        ProductCategory:"Book",
        Title:"Book 103 Title"
    }
}
```

インポート形式のクォータと検証

インポートクォータ

us-east-1、us-west-2、および eu-west-1 の各リージョンでは、Amazon S3 からの DynamoDB のインポートは、一度に 15 TB の合計インポートソースオブジェクトサイズで 50 個の同時インポートジョブに対応できます。他のすべてのリージョンでは、合計サイズが 1 TB で最大 50 個の同時インポートタスクに対応できます。各インポートジョブでは、すべてのリージョンで最大 50,000 個の Amazon S3 オブジェクトを使用できます。これらのデフォルトクォータはすべてのアカウントに適用されます。これらのクォータを改訂する必要があると考える場合は、アカウントチームに連絡してください。ケースバイケースで検討されます。DynamoDB の制限の詳細については、「Service Quotas」を参照してください。

検証エラー

インポートプロセス中に、DynamoDB でデータの解析中にエラーが発生することがあります。エラーごとに、DynamoDB は CloudWatch ログを出力し、発生したエラーの総数を記録します。Amazon S3 オブジェクト自体の形式が正しくない場合や、その内容が DynamoDB 項目を形成できない場合は、オブジェクトの残りの部分の処理がスキップされることがあります。

> **Note**
>
> Amazon S3 データソースに同じキーを共有する項目が複数ある場合、項目は 1 つが残るまで上書きされます。この場合、1 つの項目がインポートされ、他の項目が無視されたかのように見えることがあります。重複した項目はランダムな順序で上書きされ、エラーとしてカウントされず、CloudWatch ログにも出力されません。
> インポートが完了すると、インポートされた項目の総数、エラーの総数、および処理された項目の合計数が表示されます。トラブルシューティングをさらに進めるには、インポートされた項目の合計サイズと処理されたデータの合計サイズを確認することもできます。

インポートエラーには、API 検証エラー、データ検証エラー、および設定エラーの 3 つのカテゴリがあります。

API 検証エラー

API 検証エラーは、同期 API からの項目レベルのエラーです。一般的な原因として、アクセス許可の問題、必須パラメータの欠落、およびパラメータ検証の失敗が挙げられます。API コールが失敗した理由の詳細は、`ImportTable` リクエストによってスローされる例外に含まれています。

データ検証エラー

データ検証エラーは、項目レベルまたはファイルレベルで発生する可能性があります。インポート中、ターゲットテーブルにインポートする前に、DynamoDB ルールに基づいて項目が検証されます。項目が検証に失敗し、インポートされない場合、インポートジョブはその項目をスキップし、次の項目に進みます。ジョブの終了時に、インポートのジョブは FAILED に設定され、FailureCode、ItemValidationError、および FailureMessage「Some of the items failed validation checks and were not imported。Please check CloudWatch error logs for more details.」(一部の項目は検証チェックに失敗したため、インポートされませんでした。詳細については、CloudWatch エラーログを確認してください) が表示されます。

データ検証エラーの一般的な原因として、オブジェクトが解析できない、オブジェクトの形式が正しくない (入力で DYNAMODB_JSON が指定されているが、オブジェクトが DYNAMODB_JSON でない)、および指定されたソーステーブルキーとスキーマの不一致が挙げられます。

設定エラー

設定エラーは、通常、アクセス許可の検証によるワークフローエラーです。インポートワークフローでは、リクエストを受け入れた後、一部のアクセス許可がチェックされます。Amazon S3 や CloudWatch など、必要な依存関係の呼び出しで問題が発生した場合、プロセスでインポートステータスが FAILED とマークされます。`failureCode` および `failureMessage` は、失敗の理由を指します。該当する場合、失敗メッセージには、CloudTrail での失敗の理由を調査するために使用できるリクエスト ID も含まれます。

よくある設定エラーとして、Amazon S3 バケットの URL が間違っていることや、Amazon S3 バケット、CloudWatch Logs、および Amazon S3 オブジェクトの復号に使用される AWS KMS キーにアクセスする許可がないことが挙げられます。詳細については、「Using and data keys」(データキーの使用) を参照してください。

ソース Amazon S3 オブジェクトの検証

ソース S3 オブジェクトの検証を行うには、次のステップを実行します。

1. データ形式と圧縮タイプの検証

   - 指定したプレフィックスで一致するすべての Amazon S3 オブジェクトが、同じ形式 (DYNAMODB_JSON、DYNAMODB_ION、CSV) であることを確認します。
   - 指定したプレフィックスで一致するすべての Amazon S3 オブジェクトが、同じ方法 (GZIP、ZSTD、NONE) で圧縮されていることを確認します。

   > Note
   >
   > ImportTable 呼び出しで指定された入力形式が優先されるため、Amazon S3 オブジェクトには、対応する拡張子 (.csv、.json、.ion、.gz、.zstd など) は必要はありません。

2. インポートデータが目的のテーブルスキーマに準拠していることを検証します

   - ソースデータの各項目にプライマリキーがあることを確認します。ソートキーはインポートのオプションです。
   - プライマリキーとソートキーに関連付けられた属性タイプが、テーブル作成パラメータで指定されているように、テーブルおよび GSI スキーマの属性タイプと一致することを確認します。

トラブルシューティング

CloudWatch ログ

インポートジョブが失敗した場合、詳細なエラーメッセージが CloudWatch Logs に投稿されます。これらのログにアクセスするには、まず出力から ImportArn を取得し、次のコマンドを使用して describe-import を実行します。

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

出力例:

```
aws dynamodb describe-import --import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311"
{
    "ImportTableDescription": {
        "ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658528578619-c4d4e311",
        "ImportStatus": "FAILED",
        "TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
        "TableId": "7b7ecc22-302f-4039-8ea9-8e7c3eb2bcb8",
        "ClientToken": "30f8891c-e478-47f4-af4a-67a5c3b595e3",
        "S3BucketSource": {
            "S3BucketOwner": "ACCOUNT",
            "S3Bucket": "my-import-source",
            "S3KeyPrefix": "import-test"
        },
        "ErrorCount": 1,
        "CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-
dynamodb/imports:*",
        "InputFormat": "CSV",
        "InputCompressionType": "NONE",
        "TableCreationParameters": {
            "TableName": "target-table",
            "AttributeDefinitions": [
                {
                    "AttributeName": "pk",
                    "AttributeType": "S"
                }
            ],
            "KeySchema": [
                {
                    "AttributeName": "pk",
                    "KeyType": "HASH"
                }
            ],
            "BillingMode": "PAY_PER_REQUEST"
        },
        "StartTime": 1658528578.619,
        "EndTime": 1658528750.628,
        "ProcessedSizeBytes": 70,
        "ProcessedItemCount": 1,
        "ImportedItemCount": 0,
        "FailureCode": "ItemValidationError",
        "FailureMessage": "Some of the items failed validation checks and were not
 imported. Please check CloudWatch error logs for more details."
    }
}
```

上記のレスポンスからロググループとインポート ID を取得し、それを使用してエラーログを取得します。インポート ID は、ImportArn フィールドの最後のパス要素です。ロググループ名は /

`aws-dynamodb/imports` です。エラーログストリーム名は `import-id/error` です。この例では、`01658528578619-c4d4e311/error` のようになります。

項目にキー pk がない

ソース S3 オブジェクトにパラメータとして指定されたプライマリキーが含まれていない場合、インポートは失敗します。例えば、インポートのプライマリキーを列名「pk」として定義する場合などです。

```
aws dynamodb import-table —s3-bucket-source S3Bucket=my-import-
source,S3KeyPrefix=import-test.csv \
            —input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"pk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"pk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}'
```

列「pk」が、次のコンテンツを含むソースオブジェクト `import-test.csv` にありません。

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

データソースにプライマリキーがないため、項目の検証エラーによりこのインポートは失敗します。

CloudWatch エラーログの例:

```
aws logs get-log-events —log-group-name /aws-dynamodb/imports —log-stream-name
 01658528578619-c4d4e311/error
{
"events": [
{
"timestamp": 1658528745319,
"message": "{\"itemS3Pointer\":{\"bucket\":\"my-import-source\",\"key\":
\"import-test.csv\",\"itemIndex\":0},\"importArn\":\"arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311\",\"errorMessages
\":[\"One or more parameter values were invalid: Missing the key pk in the item\"]}",
"ingestionTime": 1658528745414
}
],
"nextForwardToken": "f/36986426953797707963335499204463414460239026137054642176/s",
"nextBackwardToken": "b/36986426953797707963335499204463414460239026137054642176/s"
}
```

このエラーログには、「One or more parameter values were invalid: Missing the key pk in the item」(1 つ以上のパラメータ値が無効でした: 項目にキー pk がありません) と表示されます。このインポートジョブが失敗したため、「target-table」テーブルが存在し、項目がインポートされなかったことから空になっています。最初の項目が処理され、オブジェクトは項目の検証に失敗しました。

問題を解決するには、「target-table」が不要になった場合は削除します。次に、ソースオブジェクトに存在するプライマリキー列名を使用するか、ソースデータを次のように更新します。

```
pk,title,artist,year_of_release
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

ターゲットテーブルが存在する

インポートジョブを開始して、次のようなレスポンスを受け取った場合:

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

このエラーを修正するには、まだ存在しないテーブル名を選択し、インポートを再試行する必要があります。

指定されたバケットは存在しません

ソースバケットが存在しない場合、インポートは失敗し、エラーメッセージの詳細が CloudWatch に記録されます。

インポートの説明の例:

```
aws dynamodb —endpoint-url $ENDPOINT describe-import —import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658530687105-e6035287"
{
"ImportTableDescription": {
"ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658530687105-e6035287",
"ImportStatus": "FAILED",
"TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
"TableId": "e1215a82-b8d1-45a8-b2e2-14b9dd8eb99c",
"ClientToken": "3048e16a-069b-47a6-9dfb-9c259fd2fb6f",
"S3BucketSource": {
"S3BucketOwner": "531234567890",
"S3Bucket": "BUCKET_DOES_NOT_EXIST",
"S3KeyPrefix": "import-test"
},
"ErrorCount": 0,
"CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-dynamodb/
imports:*",
"InputFormat": "CSV",
"InputCompressionType": "NONE",
"TableCreationParameters": {
"TableName": "target-table",
"AttributeDefinitions": [
{
"AttributeName": "pk",
"AttributeType": "S"
}
],
"KeySchema": [
{
"AttributeName": "pk",
"KeyType": "HASH"
}
],
"BillingMode": "PAY_PER_REQUEST"
},
"StartTime": 1658530687.105,
"EndTime": 1658530701.873,
"ProcessedSizeBytes": 0,
"ProcessedItemCount": 0,
"ImportedItemCount": 0,
"FailureCode": "S3NoSuchBucket",
"FailureMessage": "The specified bucket does not exist (Service: Amazon S3; Status
 Code: 404; Error Code: NoSuchBucket; Request ID: Q4W6QYYFDWY6WAKH; S3 Extended Request
 ID: ObqSlLeIMJpQqHLRX2C5Sy7n+8g6iGPwy7ixg7eEeTuEkg/+chU/JF+RbliWytMlkUlUcuCLTrI=;
 Proxy: null)"
}
}
```

FailureCode は S3NoSuchBucket となり、FailureMessag には、リクエスト ID や、エラーをスローしたサービスなどの詳細が含まれます。データがテーブルにインポートされる前にエラーがキャッチされたため、新しい DynamoDB テーブルは作成されません。場合によっては、データインポートの開始後にこれらのエラーが発生すると、部分的にインポートされたデータを含むテーブルが保持されることがあります。

このエラーを修正するには、ソース Amazon S3 バケットが存在することを確認してから、インポートプロセスを再開します。

Amazon S3 から DynamoDB にインポートするためのベストプラクティス

以下は、Amazon S3 から DynamoDB にデータをインポートするためのベストプラクティスです。

S3 オブジェクトの数を 50,000 個までに制限する

インポートジョブごとに最大 50,000 個の S3 オブジェクトがサポートされています。データセットに 50,000 個を超えるオブジェクトが含まれている場合は、それらをより大きなオブジェクトに統合することを検討してください。

過度に大きな S3 オブジェクトは避ける

S3 オブジェクトは並列してインポートされます。中規模の S3 オブジェクトが多数ある場合は、過剰なオーバーヘッドを発生せずに並列実行が可能です。1 KB 未満の項目については、各 S3 オブジェクトに 4,000,000 個の項目を配置することを検討してください。項目の平均サイズが大きい場合は、それに比例して各 S3 オブジェクトに配置する項目数を減らします。

ソート順のデータをランダム化する

S3 オブジェクトがデータをソート順に保持している場合、ローリングホットパーティションを作成する場合があります。これは、あるパーティションがすべてのアクティビティを受け取り、その後に次のパーティションがアクティビティを受け取る、というような状況です。ソート順のデータは、インポート中に同じターゲットパーティションに書き込まれる S3 オブジェクト内の連続した項目として定義されます。データがソート順序になる一般的な状況の 1 つは、項目がパーティションキーでソートされ、繰り返される項目が同じパーティションキーを共有する CSV ファイルです。

ホットパーティションのローリングを回避するため、こうしたケースでは順序をランダム化することをお勧めします。これにより、書き込みオペレーションを分散してパフォーマンスを向上させることができます。詳細については、「データアップロード中の書き込みアクティビティの効率的な分散」を参照してください。

データを圧縮して、S3 オブジェクトの合計サイズをリージョン内の制限以下に保つ

S3 からのインポートプロセスでは、インポートする S3 オブジェクトデータの合計サイズに制限があります。us-east-1、us-west-2、eu-west-1 リージョンでは 15 TB、その他のすべてのリージョンでは 1 TB までです。制限は raw S3 オブジェクトサイズに基づいています。

圧縮すると、より多くの raw データを制限内に収めることができます。圧縮だけではインポートが制限内に収まらない場合は、AWS Premium Support に連絡してクォータを増やすこともできます。

項目のサイズがパフォーマンスにどのように影響するか注意する

項目の平均サイズが非常に小さい (200 バイト未満) 場合、インポート処理は項目サイズが大きい場合よりも多少時間を要する場合があります。

グローバルセカンダリインデックスなしでのインポートを検討する

インポート作業の所要時間は、1 つまたは複数のグローバルセカンダリインデックス (GSI) の有無によって異なる場合があります。カーディナリティの低いパーティションキーを使用してインデックスを作成する予定の場合、インポート作業が終了するまでインデックスの作成を延期すると (それらをインポートジョブに含めるよりも) インポート速度が向上する可能性があります。

> Note
>
> インポート中に GSI を作成しても書き込み料金は発生しません (インポート後に GSI を作成すると発生します)。

Amazon S3 への DynamoDB データのエクスポート: 仕組み

DynamoDB の S3 へのエクスポートは、DynamoDB データを Amazon S3 バケットに大規模にエクスポートするためのフルマネージドソリューションです。S3 への DynamoDB エクスポートを使用すると、Amazon DynamoDB テーブルのデータをポイントインタイムリカバリ（PITR) ウィンドウ内の任意の時点から Amazon S3 バケットにエクスポートできます。エクスポート機能を使用するには、テーブルで PITR を有効にする必要があります。この機能を使用すると、Athena、AWS Glue、Amazon SageMaker、Amazon EMR や AWS Lake Formation などの他の AWS サービスを使用して、データの分析や複雑なクエリを有効にできます。

S3 への DynamoDB エクスポートでは、DynamoDB テーブルからフルデータと増分データの両方をエクスポートできます。エクスポートは読み込みキャパシティーユニット (RCU) を一切消費せず、テーブルのパフォーマンスや可用性にも影響しません。サポートされているエクスポートファイル形式は、DynamoDB JSON 形式と Amazon Ion 形式です。また、データを別の AWS アカウントが所有する S3 バケットや、異なる AWS リージョンにエクスポートすることもできます。データは常にエンドツーエンドで暗号化されます。

DynamoDB のフルエクスポートは、エクスポートが行われた時点の DynamoDB テーブル (テーブルデータとローカルセカンダリインデックス) のサイズ (テーブルデータとローカルセカンダリインデックス) に基づいて課金されます。DynamoDB の増分エクスポートは、エクスポートされている期間に継続的バックアップから処理されたデータのサイズに基づいて課金されます。エクスポートされたデータを Amazon S3 に保存する場合と、Amazon S3 バケットに対して行われた PUT リクエスト

には追加料金が適用されます。これらの料金の詳細については、「Amazon DynamoDB 料金表」と「Amazon S3 料金表」を参照してください。

Service Quotas の詳細については、「Amazon S3 へのテーブルのエクスポート」を参照してください。

トピック

- DynamoDB でテーブルのエクスポートをリクエストする
- DynamoDB テーブルのエクスポート出力形式

DynamoDB でテーブルのエクスポートをリクエストする

DynamoDB テーブルのエクスポートを使用すると、テーブルデータを Amazon S3 バケットにエクスポートできます。これにより、Athena、AWS Glue、Amazon SageMaker、Amazon EMR、および AWS Lake Formation などの他の AWS サービスを使用して、データに対して分析や複雑なクエリの実行が許可されます。テーブルのエクスポートをリクエストするには、AWS Management Console、AWS CLI、または DynamoDB API のいずれかを使用します。

> ⓘ Note
>
> Amazon S3 のリクエスタ支払いバケットはサポートされていません。

DynamoDB はフルエクスポートと増分エクスポートの両方をサポートしています。

- フルエクスポートを使用すると、テーブルの完全なスナップショットをポイントインタイムリカバリ (PITR) ウィンドウ内の任意の時点から Amazon S3 バケットにエクスポートできます。
- 増分エクスポートでは、指定した期間に変更、更新、または削除された DynamoDB テーブルのデータを、PITR ウィンドウ内で Amazon S3 バケットにエクスポートできます。

トピック

- 前提条件
- AWS Management Console を使用してエクスポートをリクエストする
- AWS Management Console で過去のエクスポートの詳細を取得する
- AWS CLI を使用してエクスポートをリクエストする
- AWS CLI で過去のエクスポートの詳細を取得する

- AWS SDK を使用してエクスポートをリクエストする
- AWS SDK を使用して過去のエクスポートの詳細を取得する

前提条件

PITR を有効にする

S3 へのエクスポート機能を使用するには、テーブルで PITR を有効にする必要があります。PITR を有効にする方法の詳細については、「ポイントインタイムリカバリ」を参照してください。PITR が有効になっていないテーブルのエクスポートをリクエストすると、「ExportTableToPointInTime オペレーションの呼び出し時にエラーが発生しました (PointInTimeRecoveryUnavailableException): テーブル 'my-dynamodb-table' のポイントインタイムリカバリが有効になっていません」という例外メッセージが表示されてリクエストが失敗します。

S3 アクセス権限の設定

テーブルデータは、書き込みアクセス許可のある任意の Amazon S3 バケットにエクスポートできます。エクスポート先のバケットは、ソーステーブル所有者と同じ AWS リージョンに存在する必要はなく、所有者が同じである必要もありません。AWS Identity and Access Management (IAM) ポリシーでは、S3 アクション (s3:AbortMultipartUpload、s3:PutObject、およびs3:PutObjectAcl) と DynamoDB エクスポートアクション (dynamodb:ExportTableToPointInTime) を実行できるようにする必要があります。S3 バケットへのエクスポートを実行するアクセス許可をユーザーに付与するポリシーの例を次に示します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "AllowWriteToDestinationBucket",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::your-bucket/*"
        }
    ]
}
```

別のアカウントにある S3 バケットに書き込む必要がある場合や書き込みアクセス許可がない場合、S3 バケット所有者は、DynamoDB からバケットへのエクスポートをユーザーに許可するバケットポリシーを追加する必要があります。エクスポート先の S3 バケットのポリシー例を次に示します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::awsexamplebucket1/*"
        }
    ]
}
```

エクスポートの進行中にこれらの権限を取り消すと、ファイルの一部だけがエクスポートされます。

> **Note**
>
> エクスポート先のテーブルまたはバケットがカスタマーマネージドキーで暗号化されている場合、その KMS キーのポリシーは、これを使用する許可を DynamoDB に与える必要があります。この権限は、エクスポートジョブをトリガーする IAM ユーザー/ロールを通じて付与されます。ベストプラクティスを含む暗号化の詳細については、「Amazon DynamoDB が AWS KMS を使用する方法」および「カスタム KMS キーの使用」を参照してください。

AWS Management Console を使用してエクスポートをリクエストする

DynamoDB コンソールを使用して、`MusicCollection` という既存のテーブルをエクスポートする方法を次の例に示します。

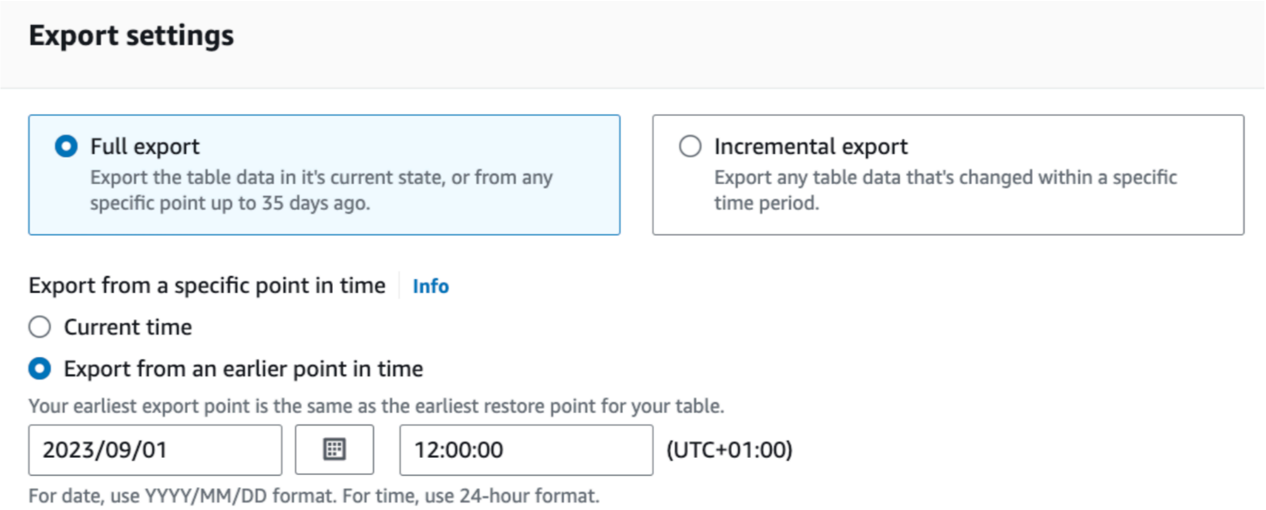
> **Note**
>
> この手順は、ポイントインタイムリカバリを有効にしていることを前提としています。これを `MusicCollection` テーブルに対して有効にするには、[Overview] (概要) タブにある [Table details] (テーブルの詳細) セクションの [Point-in-time recovery] (ポイントインタイムリカバリ) で [Enable] (有効化) を選択します。

テーブルのエクスポートをリクエストするには

1. AWS Management Console にサインインして DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/) を開きます。
2. コンソールの左側のナビゲーションペインで、[S3 へのエクスポート] を選択します。
3. [S3 にエクスポート] ボタンを選択します。
4. コピー元テーブルとエクスポート先の S3 バケットを選択します。エクスポート先のバケットがお客様のアカウントによって所有されている場合は、[Browse S3] (S3 を参照) ボタンをクリックしてバケットを検索します。そうでない場合は、`s3://`**`bucketname`**`/`**`prefix`** `format.` の形式でバケットの URL を入力します。ここで、**`prefix`** は、エクスポート先のバケットを整理するのに役立つオプションのフォルダです。
5. [フルエクスポート] または [増分エクスポート] を選択します。フルエクスポートでは、指定した時点のテーブルのフルテーブルスナップショットが出力されます。増分エクスポートでは、指定したエクスポート期間中にテーブルに加えられた変更が出力されます。出力は、エクスポート期間における項目の最終状態のみを含むように圧縮されます。同じエクスポート期間内に複数の更新があったとしても、項目はエクスポートに 1 回しか表示されません。

   Full export

   1. テーブルスナップショット全体をエクスポートする時点を選択します。これは PITR ウィンドウ内のどの時点でもかまいません。または、[現在の時刻] を選択して最新のスナップショットをエクスポートすることもできます。



Export settings

● Full export
Export the table data in it's current state, or from any specific point up to 35 days ago.

○ Incremental export
Export any table data that's changed within a specific time period.

Export from a specific point in time | Info

○ Current time

● Export from an earlier point in time

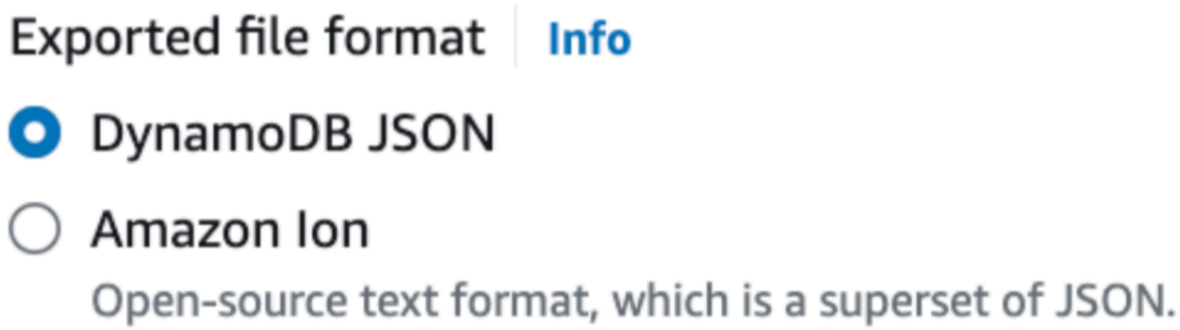
Your earliest export point is the same as the earliest restore point for your table.

2023/09/01 | 12:00:00 (UTC+01:00)

For date, use YYYY/MM/DD format. For time, use 24-hour format.

2. [エクスポートされたファイル形式] で、[DynamoDB JSON] と [Amazon Ion] のどちらかを選択します。デフォルトで、テーブルは、ポイントインタイムリカバリウィンドウの復元可能な最新の時点から DynamoDB JSON 形式でエクスポートされ、Amazon S3 キー (SSE S3) を使用して暗号化されます。これらのエクスポート設定は、必要に応じて変更できます。

> Note
>
> AWS Key Management Service (AWS KMS) によって保護されたキーを使用してエクスポートを暗号化することを選択した場合、そのキーはエクスポート先 S3 バケットと同じリージョンにある必要があります。



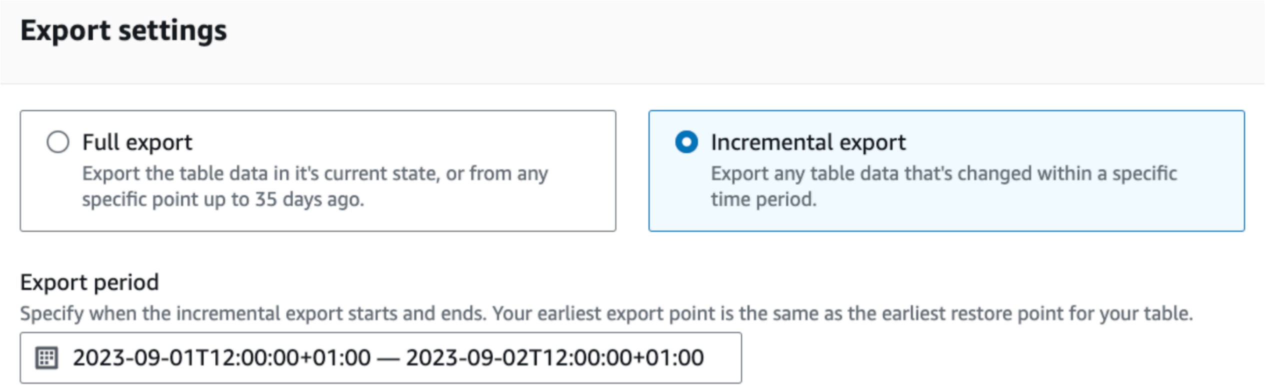
Exported file format | Info

● DynamoDB JSON

○ Amazon Ion
Open-source text format, which is a superset of JSON.

Incremental export

1. 増分データをエクスポートする [エクスポート期間] を選択します。PITR ウィンドウ内で開始時刻を選択します。エクスポート期間は 15 分～24 時間に設定する必要があります。エクスポート期間の開始時刻は期間に含まれ、終了時刻は期間に含まれません。



Export settings

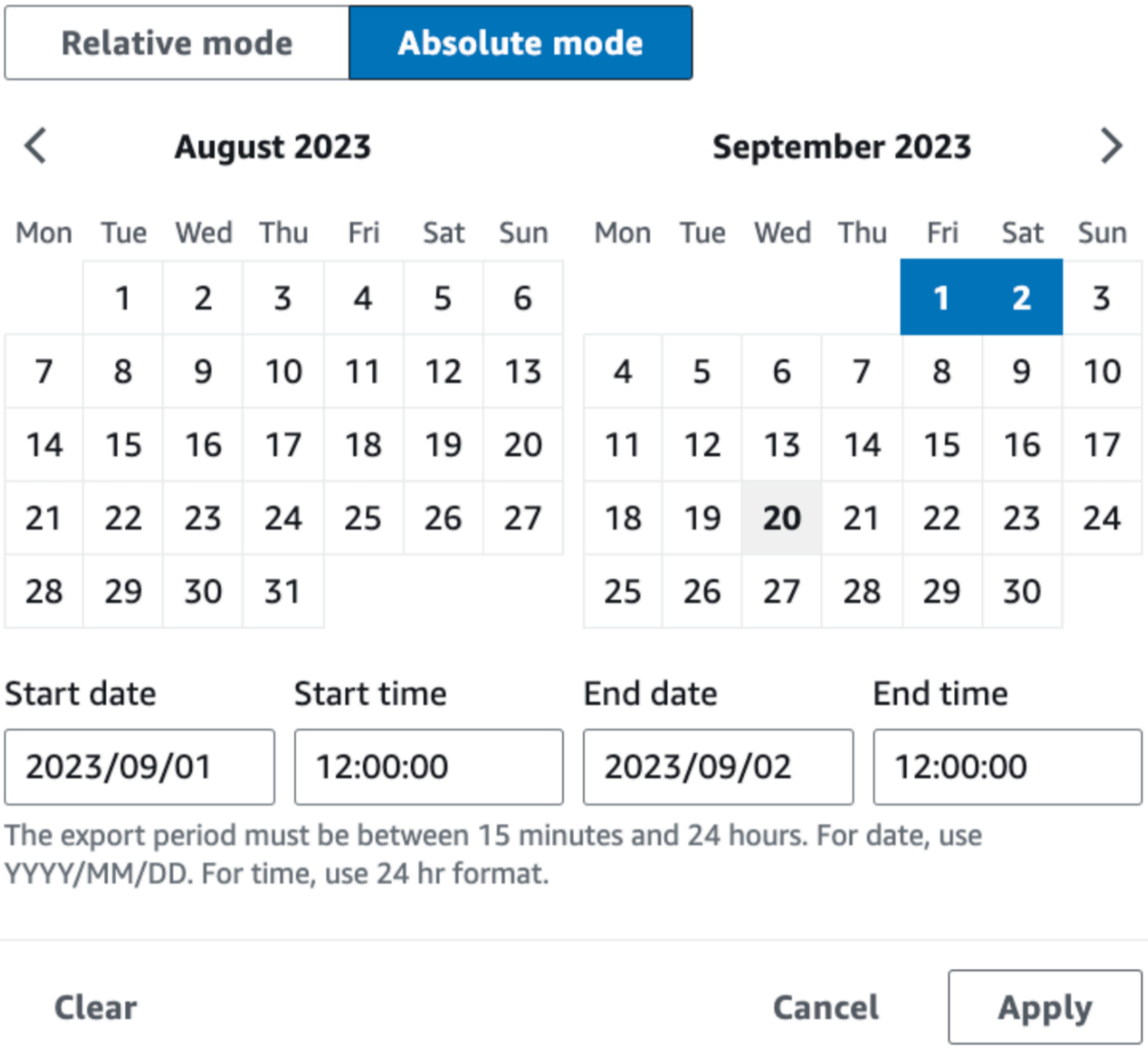
○ Full export
Export the table data in it's current state, or from any specific point up to 35 days ago.

● Incremental export
Export any table data that's changed within a specific time period.

Export period
Specify when the incremental export starts and ends. Your earliest export point is the same as the earliest restore point for your table.
2023-09-01T12:00:00+01:00 — 2023-09-02T12:00:00+01:00

2. [絶対モード] または [相対モード] を選択します。

   a. [絶対モード] では、指定した期間の増分データがエクスポートされます。



August 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

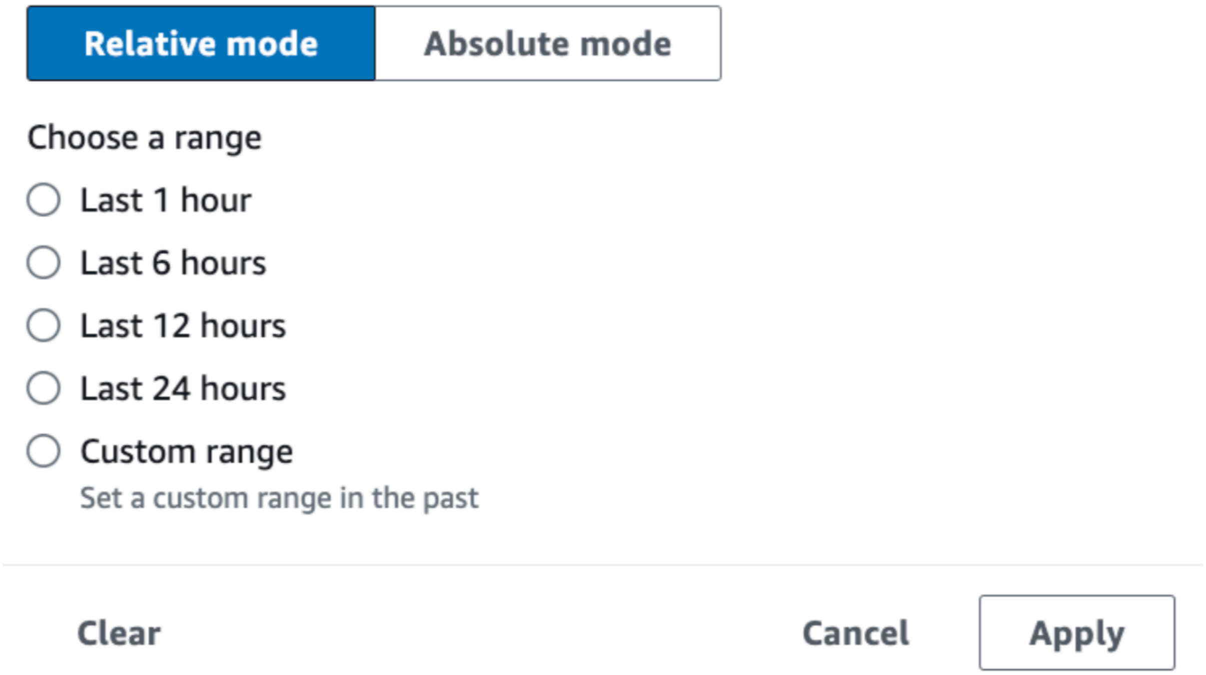
September 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Start date	Start time	End date	End time
2023/09/01	12:00:00	2023/09/02	12:00:00

The export period must be between 15 minutes and 24 hours. For date, use YYYY/MM/DD. For time, use 24 hr format.

b. [相対モード] では、エクスポートジョブの送信時間を基準にしたエクスポート期間の差分データがエクスポートされます。



Choose a range

○ Last 1 hour

○ Last 6 hours

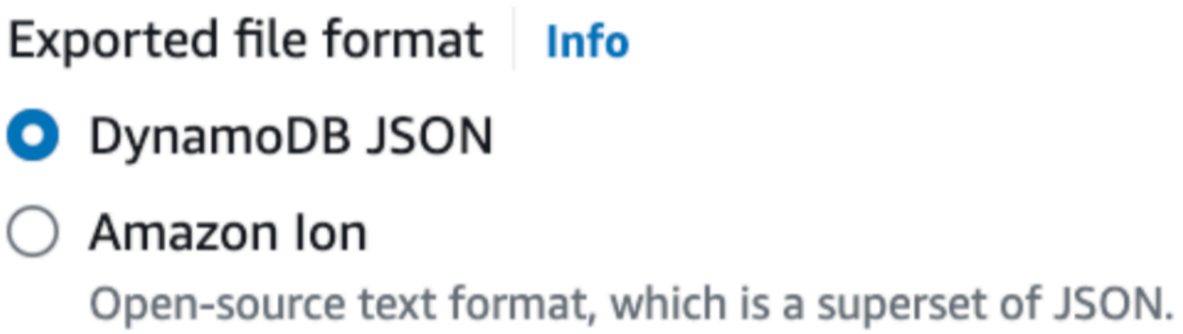
○ Last 12 hours

○ Last 24 hours

○ Custom range
Set a custom range in the past

3. [エクスポートされたファイル形式] で、[DynamoDB JSON] と [Amazon Ion] のどちらかを選択します。デフォルトで、テーブルは、ポイントインタイムリカバリウィンドウの復元可能な最新の時点から DynamoDB JSON 形式でエクスポートされ、Amazon S3 キー (SSE S3) を使用して暗号化されます。これらのエクスポート設定は、必要に応じて変更できます。

> Note
>
> AWS Key Management Service (AWS KMS) によって保護されたキーを使用してエクスポートを暗号化することを選択した場合、そのキーはエクスポート先 S3 バケットと同じリージョンにある必要があります。

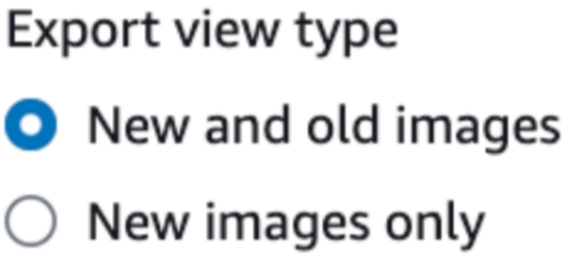


Exported file format

● DynamoDB JSON

○ Amazon Ion
Open-source text format, which is a superset of JSON.

4. [エクスポートビュータイプ] では、[新旧イメージ] または [新しいイメージのみ] を選択します。新しいイメージには、項目の最新の状態が表示されます。古いイメージには、指定した「開始日時」の直前の項目の状態が表示されます。デフォルト設定は [新旧イメージ] です。新旧イメージの詳細については、「増分エクスポート出力」を参照してください。



Export view type

● New and old images

○ New images only

6. [エクスポート] を選択して開始します。

エクスポートされたデータは、トランザクションとして一貫しません。トランザクションオペレーションは、2 つのエクスポート出力間で分断される可能性があります。トランザクションオペレーションで変更された項目のサブセットはエクスポートに反映される一方で、同じトランザクションで変更された別のサブセットは同じエクスポートリクエストに反映されない場合があります。ただし、最終的にエクスポートの一貫性は保たれます。エクスポート中にトランザクションが分断された場合、残りのトランザクションは直後のエクスポートに (重複することなく) 含まれます。エクスポートに使用される時間は内部システムクロックに基づいており、変動する可能性はアプリケーションのローカルクロックの 1 分のみです。

AWS Management Console で過去のエクスポートの詳細を取得する

過去に実行したエクスポートタスクに関する情報は、ナビゲーションサイドバーで [S3 へのエクスポート] セクションを選択して確認できます。このセクションには、過去 90 日間に作成したエクスポートの一覧が表示されます。[エクスポート] タブにリストされているタスクの ARN を選択すると、エクスポートに関する情報 (選択した詳細設定など) を取得できます。エクスポートタスクのメタデータは 90 日後に期限切れになり、それより古いジョブはこのリストに表示されなくなりますが、S3 バケット内のオブジェクトは、バケットポリシーで許可されている限り維持されます。エクスポート中に S3 バケットに作成されたオブジェクトが DynamoDB によって削除されることはありません。

AWS CLI を使用してエクスポートをリクエストする

以下の例は、AWS CLI を使用して、MusicCollection という名前の既存のテーブルを ddb-export-musiccollection という S3 バケットにエクスポートする方法を示しています。

> Note
>
> この手順は、ポイントインタイムリカバリを有効にしていることを前提としています。この機能を MusicCollection テーブルに対して有効にするには、次のコマンドを実行します。
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Full export

次のコマンドは、2020-Nov というプレフィックスが付いた ddb-export-musiccollection-9012345678 という S3 バケットに MusicCollection をエクスポートします。テーブルデータは、ポイントインタイムリカバリウィンドウの特定の時点から DynamoDB JSON 形式でエクスポートされ、Amazon S3 キー (SSE-S3) を使用して暗号化されます。

> Note
>
> クロスアカウントテーブルのエクスポートをリクエストする場合は、必ず --s3-bucket-owner オプションを含めます。

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
  --s3-bucket ddb-export-musiccollection-9012345678 \
  --s3-prefix 2020-Nov \
  --export-format DYNAMODB_JSON \
  --export-time 1604632434 \
  --s3-bucket-owner 9012345678 \
  --s3-sse-algorithm AES256
```

Incremental export

次のコマンドは、新しい `--export-type` と `--incremental-export-specification` を指定して増分エクスポートを実行します。イタリックの値はすべて独自の値に置き換えてください。時間はエポックからの秒数で指定します。

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
  --s3-bucket BUCKET --s3-prefix PREFIX \
  --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
  --export-type INCREMENTAL_EXPORT
```

> **Note**
>
> AWS Key Management Service (AWS KMS) によって保護されたキーを使用してエクスポートを暗号化することを選択した場合、そのキーはエクスポート先 S3 バケットと同じリージョンにある必要があります。

AWS CLI で過去のエクスポートの詳細を取得する

過去に実行したエクスポートリクエストに関する情報は、`list-exports` コマンドを実行して見つけることができます。このコマンドは、過去 90 日間に作成されたすべてのエクスポートのリストを返します。エクスポートタスクのメタデータは 90 日後に期限切れになり、それより古いジョブは `list-exports` コマンドによって返されなくなりますが、S3 バケット内のオブジェクトは、バケットポリシーで許可されている限り維持されます。エクスポート中に S3 バケットに作成されたオブジェクトが DynamoDB によって削除されることはありません。

エクスポートのステータスは、成功または失敗するまで `PENDING` となります。成功すると、ステータスは `COMPLETED` に変わります。失敗すると、ステータスは `FAILED` に変わり、`failure_message` と `failure_reason` が追加されます。

以下の例では、オプションの `table-arn` パラメータを使用して、特定のテーブルのエクスポートのみを一覧表示します。

```
aws dynamodb list-exports \
    --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

詳細な構成設定など、特定のエクスポートタスクに関する詳細情報を取得するには、describe-export コマンドを使用します。

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

AWS SDK を使用してエクスポートをリクエストする

これらのコードスニペットを使用して、好きな AWS SDK を使ったテーブルのエクスポートをリクエストします。

Python

フルエクスポート

```
import boto3
from datetime import datetime

# remove endpoint_url for real use
client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/export_table_to_point_in_time.html
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

増分エクスポート

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')

client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    IncrementalExportSpecification={
      'ExportFromTime': datetime(2023, 9, 20, 12, 0, 0),
      'ExportToTime': datetime(2023, 9, 20, 13, 0, 0),
      'ExportViewType': 'NEW_AND_OLD_IMAGES'
    },
    ExportType='INCREMENTAL_EXPORT',
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

AWS SDK を使用して過去のエクスポートの詳細を取得する

これらのコードスニペットを使用して、好きな AWS SDK を使った過去のテーブルのエクスポートの詳細を取得します。

Python

フルエクスポート

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/list_exports.html

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
  )
)
```

増分エクスポート

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/describe_export.html
```

```
print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

DynamoDB テーブルのエクスポート出力形式

DynamoDB テーブルのエクスポートには、テーブルデータを含むファイルに加えて、マニフェストファイルが含まれます。これらのファイルはすべて、エクスポート要求で指定した Amazon S3 バケットに保存されます。以下のセクションでは、各出力オブジェクトの形式と内容について説明します。

フルエクスポート出力

マニフェストファイル

DynamoDB では、エクスポートリクエストごとにマニフェストファイルがチェックサムファイルとともに指定した S3 バケットに作成されます。

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

テーブルのエクスポートをリクエストする際は、**`export-prefix`** を選択します。これにより、エクスポート先 S3 バケットのファイルを整理できます。**`ExportId`** は、同じ S3 バケットへの複数のエクスポートを保証する、サービスによって生成される一意のトークンです。`export-prefix` は相互に上書きしません。

エクスポートでは、パーティションごとに少なくとも 1 つのファイルが作成されます。空のパーティションの場合、エクスポートリクエストにより空のファイルが作成されます。各ファイル内のすべての項目は、その特定のパーティションのハッシュされたキースペースからのものです。

Note

また、DynamoDB では、マニフェストファイルと同じディレクトリに `_started` という名前の空のファイルが作成されます。このファイルは、エクスポート先のバケットが書き込み

可能であること、およびエクスポートが開始されたことを検証します。これは安全に削除できます。

要約マニフェスト

manifest-summary.json ファイルには、エクスポートジョブに関する概要情報が含まれています。これにより、共有データフォルダー内のどのデータファイルがこのエクスポートに関連付けられているかがわかります。この形式は次のとおりです。

```
{
   "version": "2020-06-30",
   "exportArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4",
   "startTime": "2020-11-04T07:28:34.028Z",
   "endTime": "2020-11-04T07:33:43.897Z",
   "tableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog",
   "tableId": "12345a12-abcd-123a-ab12-1234abc12345",
   "exportTime": "2020-11-04T07:28:34.028Z",
   "s3Bucket": "ddb-productcatalog-export",
   "s3Prefix": "2020-Nov",
   "s3SseAlgorithm": "AES256",
   "s3SseKmsKeyId": null,
   "manifestFilesS3Key": "AWSDynamoDB/01693685827463-2d8752fd/manifest-files.json",
   "billedSizeBytes": 0,
   "itemCount": 8,
   "outputFormat": "DYNAMODB_JSON",
   "exportType": "FULL_EXPORT"
}
```

ファイルマニフェスト

manifest-files.json ファイルには、エクスポートされたテーブルデータを含むファイルに関する情報が含まれています。ファイルは JSON Lines 形式であるため、改行が項目区切り文字として使用されます。次の例では、読みやすくするためにファイルマニフェストの 1 つのデータファイルの詳細が複数行にフォーマットされています。

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
    "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

データファイル

DynamoDB では、DynamoDB JSON と Amazon Ion の 2 つの形式でテーブルデータをエクスポートできます。選択した形式にかかわらず、データは、 キーによって名前が付けられた複数の圧縮ファイルに書き込まれます。これらのファイルも `manifest-files.json` ファイル内に一覧表示されます。

フルエクスポート後の S3 バケットのディレクトリ構造では、export Id フォルダーにマニフェストファイルとデータファイルがすべて含まれます。

```
DestinationBucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685827463-2d8752fd     // the single full export
    #   ### manifest-files.json     // manifest points to files under 'data' subfolder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### data                    // The data exported by full export
    #   #   ### asdl123dasas.json.gz
    #   #   ...
    #   ### _started                // empty file for permission check
```

DynamoDB JSON

DynamoDB JSON 形式のテーブルエクスポートは、複数の `Item` オブジェクトで構成されます。個々のオブジェクトは DynamoDB のスタンダードマーシャリングされた JSON 形式です。

DynamoDB JSON エクスポートデータ用のカスタムパーサーを作成する場合、形式は [JSON Lines](https://jsonlines.org/) です。これは、改行が項目の区切り文字として使用されていることを意味します。Athena や AWS Glue など、多くの AWS サービスで、この形式は自動的に解析されます。

次の例では、読みやすくするために DynamoDB JSON エクスポートの 1 つの項目が複数行にフォーマットされています。

```
{
    "Item":{
        "Authors":{
            "SS":[
                "Author1",
                "Author2"
            ]
        },
        "Dimensions":{
            "S":"8.5 x 11.0 x 1.5"
        },
        "ISBN":{
            "S":"333-3333333333"
        },
        "Id":{
            "N":"103"
        },
        "InPublication":{
            "BOOL":false
        },
        "PageCount":{
            "N":"600"
        },
        "Price":{
            "N":"2000"
        },
        "ProductCategory":{
            "S":"Book"
        },
        "Title":{
            "S":"Book 103 Title"
        }
    }
}
```

Amazon Ion

Amazon Ion は、大規模なサービス指向アーキテクチャをエンジニアリングしながら、急速な開発、デカップリング、効率性といった日々の課題に対処するために構築されたリッチタイプの自己記述型階層型データシリアル化フォーマットです。DynamoDB は、Ion のテキストのフォーマットでのテーブルのエクスポートをサポートします。これは JSON のスーパーセットです。

テーブルを Ion 形式にエクスポートすると、テーブルで使用されている DynamoDB データ型が Ion データ型にマップされます。DynamoDB セットは、Ion 型注釈を使用して、ソーステーブルで使用されるデータ型を明確にします。

DynamoDB から Ion のデータ型への変換

DynamoDB データ型	Ion 表現
文字列 (S)	文字列
ブール型 (BOOL)	ブール
数値 (N)	小数点
バイナリ (B)	blob
セット (SS、NS、BS)	リスト (型注釈 $DynamoDB_SS、$DynamoDB_NS、または $DynamoDB_BS)
リスト	リスト
マップ	構造体

Ion エクスポート内の項目は、改行で区切られます。各行は Ion バージョンマーカーで始まり、Ion 形式の項目が続きます。次の例では、読みやすくするために 1 つの Ion エクスポートの項目が複数の行にフォーマットされています。

```
$ion_1_0 {
    Item:{
        Authors:$dynamodb_SS::["Author1","Author2"],
        Dimensions:"8.5 x 11.0 x 1.5",
        ISBN:"333-3333333333",
        Id:103.,
        InPublication:false,
        PageCount:6d2,
        Price:2d3,
        ProductCategory:"Book",
        Title:"Book 103 Title"
    }
}
```

増分エクスポート出力

マニフェストファイル

DynamoDB では、エクスポートリクエストごとにマニフェストファイルがチェックサムファイルとともに指定した S3 バケットに作成されます。

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

テーブルのエクスポートをリクエストする際は、**export-prefix** を選択します。これにより、エクスポート先 S3 バケットのファイルを整理できます。**ExportId** は、同じ S3 バケットへの複数のエクスポートを保証する、サービスによって生成される一意のトークンです。export-prefix は相互に上書きしません。

エクスポートでは、パーティションごとに少なくとも 1 つのファイルが作成されます。空のパーティションの場合、エクスポートリクエストにより空のファイルが作成されます。各ファイル内のすべての項目は、その特定のパーティションのハッシュされたキースペースからのものです。

> Note
>
> また、DynamoDB では、マニフェストファイルと同じディレクトリに _started という名前の空のファイルが作成されます。このファイルは、エクスポート先のバケットが書き込み可能であること、およびエクスポートが開始されたことを検証します。これは安全に削除できます。

要約マニフェスト

manifest-summary.json ファイルには、エクスポートジョブに関する概要情報が含まれています。これにより、共有データフォルダー内のどのデータファイルがこのエクスポートに関連付けられているかがわかります。この形式は次のとおりです。

```
{
 "version": "2023-08-01",
 "exportArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test/
export/01695097218000-d6299cbd",
 "startTime": "2023-09-19T04:20:18.000Z",
 "endTime": "2023-09-19T04:40:24.780Z",
 "tableArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test",
 "tableId": "b116b490-6460-4d4a-9a6b-5d360abf4fb3",
 "exportFromTime": "2023-09-18T17:00:00.000Z",
 "exportToTime": "2023-09-19T04:00:00.000Z",
 "s3Bucket": "jason-exports",
 "s3Prefix": "20230919-prefix",
 "s3SseAlgorithm": "AES256",
 "s3SseKmsKeyId": null,
 "manifestFilesS3Key": "20230919-prefix/AWSDynamoDB/01693685934212-ac809da5/manifest-
files.json",
 "billedSizeBytes": 20901239349,
 "itemCount": 169928274,
 "outputFormat": "DYNAMODB_JSON",
 "outputView": "NEW_AND_OLD_IMAGES",
 "exportType": "INCREMENTAL_EXPORT"
}
```

ファイルマニフェスト

manifest-files.json ファイルには、エクスポートされたテーブルデータを含むファイルに関する情報が含まれています。ファイルは JSON Lines 形式であるため、改行が項目区切り文字として使用されます。次の例では、読みやすくするためにファイルマニフェストの 1 つのデータファイルの詳細が複数行にフォーマットされています。

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

データファイル

DynamoDB では、DynamoDB JSON と Amazon Ion の 2 つの形式でテーブルデータをエクスポートできます。選択した形式にかかわらず、データは、 キーによって名前が付けられた複数の圧縮ファイルに書き込まれます。これらのファイルも manifest-files.json ファイル内に一覧表示されます。

増分エクスポートのデータファイルはすべて S3 バケットの共通データフォルダーに格納されています。マニフェストファイルはエクスポート ID フォルダーにあります。

```
DestinationBucket/DestinationPrefix
```

```
.
### AWSDynamoDB
    ### 01693685934212-ac809da5     // an incremental export ID
    #   ### manifest-files.json     // manifest points to files under 'data' folder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### _started                // empty file for permission check
    ### 01693686034521-ac809da5
    #   ### manifest-files.json
    #   ### manifest-files.checksum
    #   ### manifest-summary.json
    #   ### manifest-summary.md5
    #   ### _started
    ### data                        // stores all the data files for incremental
 exports
    #   ### sgad6417s6vss4p7owp0471bcq.json.gz
    #   ...
```

ファイルをエクスポートする場合、各項目の出力には、テーブル内でその項目が更新された日時を表すタイムスタンプと insert、update、または delete オペレーションであったかどうかを示すデータ構造が含まれます。タイムスタンプは内部システムクロックに基づいており、アプリケーションクロックとは異なる場合があります。増分エクスポートでは、出力構造の 2 つのエクスポートビュータイプ (新旧イメージ、または新しいイメージのみ) を選択できます。

- 新しいイメージには、項目の最新の状態が表示されます。
- 古いイメージには、指定した開始日時の直前の項目の状態が表示されます。

ビュータイプは、エクスポート期間中に項目がどのように変更されたかを確認する場合に役立ちます。また、特にダウンストリームシステムに DynamoDB パーティションキーと異なるパーティションキーがある場合に、ダウンストリームシステムを効率的に更新するのにも役立ちます。

増分エクスポート出力の項目が insert、update、または delete のどれであったかは、出力の構造を見れば推測できます。両方のエクスポートビュータイプについて、増分エクスポートの構造とそれに対応するオペレーションを以下の表にまとめています。

操作	新しいイメージのみ	新旧イメージ
Insert	キー + 新しいイメージ	キー + 新しいイメージ

操作	新しいイメージのみ	新旧イメージ
更新	キー + 新しいイメージ	キー + 新しいイメージ + 古いイメージ
削除	キー	キー + 古いイメージ
delete + insert	出力なし	出力なし

DynamoDB JSON

DynamoDB JSON 形式のテーブルエクスポートは、項目の書き込み時間を示すメタデータタイムスタンプと、それに続く項目のキーと値で構成されています。以下は、エクスポートビュータイプを新イメージおよび旧イメージとして使用した DynamoDB JSON 出力の例を示しています。

```
// Ex 1: Insert
//   An insert means the item did not exist before the incremental export window
//   and was added during the incremental export window

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Key": {
     "PK": {
       "S": "CUST#100"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#100"
     },
     "FirstName": {
       "S": "John"
     },
     "LastName": {
       "S": "Don"
     }
   }
}
```

```
// Ex 2: Update
//   An update means the item existed before the incremental export window
//   and was updated during the incremental export window.
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Key": {
     "PK": {
       "S": "CUST#200"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Grace"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#200"
     },
     "FirstName": {
       "S": "Mary"
     },
     "LastName": {
       "S": "Smith"
     }
   }
}

// Ex 3: Delete
//   A delete means the item existed before the incremental export window
//   and was deleted during the incremental export window
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Key": {
     "PK": {
       "S": "CUST#300"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#300"
     },
     "FirstName": {
       "S": "Jose"
     },
     "LastName": {
       "S": "Hernandez"
     }
   }
}

// Ex 4: Insert + Delete
//   Nothing is exported if an item is inserted and deleted within the
//   incremental export window.
```

Amazon Ion

Amazon Ion は、大規模なサービス指向アーキテクチャをエンジニアリングしながら、急速な開発、デカップリング、効率性といった日々の課題に対処するために構築されたリッチタイプの自己記述型階層型データシリアル化フォーマットです。DynamoDB は、Ion のテキストのフォーマットでのテーブルのエクスポートをサポートします。これは JSON のスーパーセットです。

テーブルを Ion 形式にエクスポートすると、テーブルで使用されている DynamoDB データ型が Ion データ型にマップされます。DynamoDB セットは、Ion 型注釈を使用して、ソーステーブルで使用されるデータ型を明確にします。

DynamoDB から Ion のデータ型への変換

DynamoDB データ型	Ion 表現
文字列 (S)	文字列
ブール型 (BOOL)	ブール

DynamoDB データ型	Ion 表現
数値 (N)	小数点
バイナリ (B)	blob
セット (SS、NS、BS)	リスト (型注釈 $DynamoDB_SS、$DynamoDB_NS、または $DynamoDB_BS)
リスト	リスト
マップ	構造体

Ion エクスポート内の項目は、改行で区切られます。各行は Ion バージョンマーカーで始まり、Ion 形式の項目が続きます。次の例では、読みやすくするために 1 つの Ion エクスポートの項目が複数の行にフォーマットされています。

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:false,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        },
        NewImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            Dimensions:"8.5 x 11.0 x 1.5",
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:true,
            PageCount:6d2,
            Price:2d3,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        }
    }
}
```

DynamoDB と Amazon OpenSearch Service のゼロ ETL 統合

Amazon DynamoDB は、OpenSearch Ingestion 用 DynamoDB プラグインを通じて Amazon OpenSearch Service とのゼロ ETL 統合が可能です。Amazon OpenSearch Ingestion では、コードを必要としないフルマネージド型のエクスペリエンスで Amazon OpenSearch Service にデータを取り込むことができます。

OpenSearch Ingestion 用 DynamoDB プラグインを使用すると、1 つ以上の DynamoDB テーブルをソースとして使用して 1 つ以上の OpenSearch Service インデックスに取り込むことができます。AWS Management Consoleの OpenSearch Ingestion または DynamoDB インテグレーションから、DynamoDB をソースとして使用する OpenSearch Ingestion パイプラインを参照して設定できます。

- OpenSearch Ingestion を初めて使う場合は、「OpenSearch Ingestion の開始方法」に記載されている手順に従ってください。
- DynamoDB プラグインの前提条件とすべての設定オプションについては、OpenSearch Ingestion 用 DynamoDB プラグインのドキュメントを参照してください。

仕組み

このプラグインは DynamoDB の Amazon S3 へのエクスポート機能を使用して、OpenSearch にロードする初期スナップショットを作成します。スナップショットがロードされると、プラグインは DynamoDB Streams を使用してそれ以降の変更をほぼリアルタイムでレプリケートします。すべての項目は OpenSearch Ingestion でイベントとして処理され、プロセッサプラグインで変更できます。属性を削除したり、複合属性を作成してルート経由でさまざまなインデックスに送信したりできます。

Amazon S3 へのエクスポート機能を使用するには、ポイントインタイムリカバリ (PITR) を有効にする必要があります。また、DynamoDB Streams を使用するには、([新旧のイメージ] オプションが選択されている状態で) DynamoDB Streams が有効になっている必要があります。エクスポート設定を除外すると、スナップショットを作ることなくパイプラインを作成できます。

また、ストリーム設定を除外すると、スナップショットのみで更新のないパイプラインを作成することもできます。このプラグインはテーブルの読み取りスループットや書き込みスループットを使用しないため、本番環境のトラフィックに影響を与えることなく安全に使用できます。ストリーム上の並列コンシューマーの数には制限があり、こうした統合を作成する前に考慮する必要があります。その他の考慮事項については、「the section called “統合に関するベストプラクティス”」を参照してください。

単純なパイプラインの場合、1 つの OpenSearch Compute Unit (OCU) は、1 秒あたり約 1 MB の書き込みを処理できます。これは約 1,000 の書き込みリクエスト単位 (WCU) に相当しますが、パイプラインの複雑さやその他の要因によって、これより多くなることも少なくなることもあります。

OpenSearch Ingestion は、回復不能なエラーの原因となるイベントのデッドレターキュー (DLQ) をサポートします。さらに、DynamoDB、パイプライン、または Amazon OpenSearch Service のいずれかでサービスが中断された場合でも、パイプラインは中断したところからユーザーの介入なしに再開します。

中断が 24 時間以上続くと、更新が失われる場合があります。ただし、中断から回復すると、パイプラインはまだ利用できる更新を引き続き処理します。デッドレターキューに入っていない限り、中断したイベントに起因する異常を修正するには、新たにインデックスを構築する必要があります。

プラグインのすべての設定と詳細については、OpenSearch Ingestion 用 DynamoDB プラグインのドキュメントを参照してください。

コンソールによる作成エクスペリエンスの統合

DynamoDB と OpenSearch Service は AWS Management Consoleでエクスペリエンスが統合されており、開始プロセスが効率化されています。以下の手順を実行すると、サービスで自動的に DynamoDB ブループリントが選択され、適切な DynamoDB 情報が自動的に追加されます。

統合を作成するには、「OpenSearch Ingestion の開始方法」に記載されている手順に従ってください。「ステップ 3: パイプラインを作成する」まで進んだら、手順の 1 と 2 を以下の手順に置き換えます。

1. DynamoDB コンソールに移動します。
2. 左側のナビゲーションペインで、[統合] を選択します。
3. OpenSearch にレプリケートする DynamoDB テーブルを選択します。
4. [Create] (作成) を選択します。

ここから、チュートリアルの以降の部分を続行します。

次のステップ

DynamoDB と OpenSearch Service の統合についての理解を深めるには、以下を参照してください。

- Amazon OpenSearch Ingestion の開始方法
- DynamoDB プラグインの設定と要件

インデックスの重大な変更の処理

OpenSearch ではインデックスに新しい属性を動的に追加できます。ただし、マッピングテンプレートが特定のキーに設定されている場合は、そのキーを変更する際に追加のアクションが必要になります。さらに、変更によって DynamoDB テーブルの全データの再処理が必要になる場合は、新たなエクスポートを開始するための手順を実行する必要があります。

> Note
>
> これらのどのオプションでも、DynamoDB テーブルと指定したマッピングテンプレートで型が競合していると、問題が発生する可能性があります。デッドレターキュー (DLQ) を必ず有効にしてください (開発中であっても) 。これにより、OpenSearch のインデックスでのインデックス作成時に競合を引き起こすレコードについて、何が間違っているのかを把握しやすくなります。

トピック

- 仕組み
- インデックスを削除してパイプラインをリセットする (パイプライン中心のオプション)
- インデックスを再作成してパイプラインをリセットする (インデックス中心のオプション)
- 新しいインデックスとシンクを作成する (オンラインオプション)
- 型の競合の回避とデバッグのベストプラクティス

仕組み

以下は、インデックスの重大な変更を処理するときに実行されるアクションの概要です。詳細な手順については、以降のセクションで説明します。

- パイプラインを停止して開始する: このオプションでは、パイプラインの状態をリセットし、パイプラインを新たにフルエクスポートで再開します。非破壊的なオプションであるため、DynamoDB 内のインデックスやデータは削除されません。これを行う前に新しいインデックスを作成しないと、インデックス内の現在の _version よりも古いドキュメントの挿入がエクスポートによって試行されるため、バージョンの競合によるエラーが多数発生する可能性があります。これらのエラーは無視して問題ありません。パイプラインの停止中は、パイプラインの料金が課金されることはありません。
- パイプラインを更新する: このオプションでは、状態を変えることなく、パイプラインの設定をブルー/グリーン方式で更新します。パイプラインに大幅な変更 (既存のインデックスに新しいルート、インデックス、またはキーを追加するなど) を行った場合は、パイプラインの完全なリセットとインデックスの再作成が必要になる場合があります。このオプションではフルエクスポートは実行されません。
- インデックスを削除して再作成する: このオプションでは、インデックスのデータ設定とマッピング設定が削除されます。マッピングに重大な変更を加える前にはこれを行う必要があります。インデックスに依存しているアプリケーションは、インデックスが再作成されて同期されるまで機能しません。インデックスを削除しても、新たなエクスポートは開始されません。インデックスの削除は、パイプラインを更新した後にのみ行ってください。そうしないと、設定を更新する前にインデックスが再作成される可能性があります。

インデックスを削除してパイプラインをリセットする (パイプライン中心のオプション)

開発途中の場合は、通常、この方法が最速のオプションになります。OpenSearch Service でインデックスを削除し、パイプラインの停止と開始を行って、全データの新たなエクスポートを開始します。これにより、マッピングテンプレートが既存のインデックスと競合したり、処理が不完全なテーブルからデータが失われたりすることがなくなります。

1. AWS Management Consoleから、あるいは AWS CLI か SDK で StopPipeline API オペレーションを使用して、パイプラインを停止します。
2. 新しい変更でパイプライン設定を更新します。
3. REST API コールまたは OpenSearch ダッシュボードを使用して OpenSearch Service のインデックスを削除します。
4. コンソールから、あるいは AWS CLI か SDK で `StartPipeline` API オペレーションを使用して、パイプラインを開始します。

> Note
>
> これにより新たなフルエクスポートが開始され、追加費用が発生します。

5. 新しいインデックスを作成するための新たなエクスポートが生成されるため、予期しない問題がないか監視します。
6. OpenSearch Service でインデックスが想定どおりであることを確認します。

エクスポートが完了し、ストリームからの読み取りが再開されると、DynamoDB テーブルデータがインデックスで使用できるようになります。

インデックスを再作成してパイプラインをリセットする (インデックス中心のオプション)

この方法は、パイプラインを DynamoDB から再開する前に OpenSearch Service でインデックスの設計を何度も繰り返す必要がある場合に適しています。これは、開発中に検索パターンのイテレーションを迅速に進めたい場合や、イテレーションのたびに新たなエクスポートの完了を待機したくない場合に役立ちます。

1. AWS Management Consoleから、あるいは AWS CLI か SDK で StopPipeline API オペレーションを呼び出して、パイプラインを停止します。
2. OpenSearch でインデックスを削除し、目的のマッピングテンプレートを使って再作成します。サンプルデータを手動で挿入して、検索が意図したとおりに機能しているかを確認できます。サンプルデータが DynamoDB のデータと競合する可能性がある場合は、次のステップに進む前に必ず削除してください。
3. パイプラインにインデックステンプレートがある場合は、削除するか、OpenSearch Service で作成済みのテンプレートに置き換えます。インデックスの名前がパイプラインでの名前と一致していることを確認してください。
4. コンソールから、あるいは AWS CLI か SDK で `StartPipeline` API オペレーションを呼び出して、パイプラインを開始します。

> Note
>
> これにより新たなフルエクスポートが開始され、追加費用が発生します。

5. 新しいインデックスを作成するための新たなエクスポートが生成されるため、予期しない問題がないか監視します。

エクスポートが完了し、ストリームからの読み取りが再開されると、DynamoDB テーブルデータがインデックスで使用できるようになります。

新しいインデックスとシンクを作成する (オンラインオプション)

この方法は、マッピングテンプレートを更新する必要があるものの、現在インデックスを本番環境で使用している場合に適しています。このオプションでは、新しいインデックスを作成し、同期と検証が完了した後にアプリケーションをそのインデックスに移動する必要があります。

> Note
>
> これにより、ストリーム上に別のコンシューマーが作成されます。AWS Lambda やグローバルテーブルのようなコンシューマーが他にもあると、問題になる可能性があります。新しいインデックスをロードするキャパシティを確保するために、既存のパイプラインの更新を一時停止しなければならなくなる場合があります。

1. 新しい設定と別のインデックス名を使用して新しいパイプラインを作成します。
2. 新しいインデックスに予期しない問題がないか監視します。
3. アプリケーションを新しいインデックスに切り替えます。
4. すべてが正しく動作していることを確認したら、古いパイプラインを停止して削除します。

型の競合の回避とデバッグのベストプラクティス

- 型の競合が生じた場合のデバッグを容易にするためには、デッドレターキュー (DLQ) を常に使用します。
- マッピングが含まれるインデックステンプレートを必ず使用し、`include_keys` を設定します。OpenSearch Service は新しいキーを動的にマッピングしますが、これによって予期しない動作 (`GeoPoint` を想定していたところ、`string` または `object` として作成された場合など) やエラー (`long` 値と `float` 値が混在している `number` など) の問題が発生する可能性があります。
- 既存のインデックスを本番環境で引き続き使用する必要がある場合は、前述のインデックスの削除手順を、パイプライン設定ファイル内のインデックスの名前を変更する手順に置き換えることもで

きます。これにより、新しいインデックスが作成されます。完了したら、新しいインデックスをポイントするようにアプリケーションを更新する必要があります。

- 型変換の問題をプロセッサで修正する場合は、`UpdatePipeline` を使ってテストできます。そのためには、停止と開始の手順を行うか、デッドレターキューの処理を実行して、以前にスキップされたエラーのあるドキュメントを修正する必要があります。

DynamoDB との統合に関するベストプラクティス

DynamoDB を他のサービスと統合する際は、各サービスを使用する際のベストプラクティスに必ず従います。ただし、統合に特有のベストプラクティスもいくつか考慮する必要があります。

トピック

- DynamoDB でスナップショットを作成する
- DynamoDB でデータ変更をキャプチャする
- OpenSearch Service と DynamoDB のゼロ ETL 統合

DynamoDB でスナップショットを作成する

- 一般的には、Amazon S3 へのエクスポートを使用して、初期レプリケーションのスナップショットを作成することをお勧めします。この方法は費用対効果が高く、スループットのためにアプリケーションのトラフィックと競合することもありません。また、新しいテーブルへのバックアップと復元の後に、スキャンオペレーションを実行することも検討してください。これにより、アプリケーションとのスループットの競合を避けることができますが、一般的にはエクスポートよりもコスト効率が大きく低下します。
- エクスポートを行う際は、必ず `StartTime` を設定します。これにより、変更データキャプチャ (CDC) をいつ開始するかを簡単に決定できます。
- S3 へのエクスポートを使用する際は、S3 バケットにライフサイクルアクションを設定します。通常、Expiration アクションは 7 日に設定しても問題ありませんが、自社で定められているガイドラインに従ってください。取り込み後にアイテムを明示的に削除した場合でも、このアクションは問題を特定するのに役立ち、不要なコストを削減し、ポリシー違反を防ぐことができます。

DynamoDB でデータ変更をキャプチャする

- ほぼリアルタイムの CDC が必要な場合は、DynamoDB Streams または Amazon Kinesis Data Streams (KDS) を使用します。どちらを使用するかを決める際、通常は、どちらがダウンストリー

ムサービスで使いやすいかを検討します。パーティションキーレベルで順番どおりにイベント処理を行う必要がある場合、または非常に大きいアイテムがある場合は、DynamoDB Streams を使用します。

- ほぼリアルタイムの CDC が必要ない場合は、Amazon S3 への増分エクスポートを使用して、2 つの時点の間に発生した変更のみをエクスポートできます。

  S3 へのエクスポートを使用してスナップショットの生成を行った場合は、同様のコードを使用して増分エクスポートを処理できるため、特に便利です。通常、S3 へのエクスポートは以前のストリーミングオプションよりも若干安価ですが、どのオプションを使用するかを決める主な要素はコストではありません。
- 通常、DynamoDB ストリームの同時コンシューマーは、最大 2 人です。統合戦略を計画する際には、この点を考慮してください。
- スキャンを使用して変更を検出しないでください。これは小規模ではうまくいくかもしれませんが、すぐに実用的ではなくなります。

OpenSearch Service と DynamoDB のゼロ ETL 統合

DynamoDB は、Amazon OpenSearch Service との DynamoDB のゼロ ETL 統合を行うことができます。詳細については、「DynamoDB plugin for OpenSearch Ingestion」および「specific best practices for Amazon OpenSearch Service」を参照してください。

構成

- 検索を実行する必要があるデータのみをインデックス化します。実装する際は、必ずマッピングテンプレート (`template_type: index_template`、`template_content`) および `include_keys` を使用します。
- ログを監視して、タイプの競合に関連するエラーがないか確認します。OpenSearch Service は、キーのすべての値が同じタイプであることを前提としています。同じでない場合、例外が生成されます。このようなエラーが発生した場合は、プロセッサを追加して、特定のキーが常に同じ値であるかどうかを確認できます。
- 通常、`document_id` 値には `primary_key` メタデータ値を使用します。OpenSearch Service では、ドキュメント ID は DynamoDB のプライマリキーと同等です。プライマリキーを使用するとドキュメントを見つけやすくなり、アップデートが矛盾なく常にそのドキュメントにレプリケートされます。

プライマリキー (例: `document_id: "${getMetadata('primary_key')}"`) は、`getMetadata` 補助関数を使用して取得できます。複合プライマリキーを使用している場合は、補助関数によって連結されます。

- 通常、`action` 設定には `opensearch_action` メタデータ値を使用します。これにより、OpenSearch Service のデータが DynamoDB の最新の状態と一致するようにアップデートがレプリケートされるようになります。

  プライマリキー (例: `action: "${getMetadata('opensearch_action')}"`) は、`getMetadata` 補助関数を使用して取得できます。フィルタリングなどの用途では、`dynamodb_event_name` を使用してストリームイベントタイプを取得することもできます。ただし、通常は設定に `action` を使用しないでください。

オブザーバビリティ

- ドロップされたイベントを処理するには、必ず OpenSearch シンクでデッドレターキュー (DLQ) を使用します。DynamoDB は一般的に OpenSearch Service ほど構造化されておらず、予期しない問題が発生する可能性は常にあります。デッドレターキューを使用すると、個々のイベントを回復できるだけでなく、回復プロセスを自動化することもできます。これにより、インデックス全体を再構築する必要がなくなります。
- レプリケーションの遅延が想定を超えたことを知らせるアラートを必ず設定します。通常は、アラートノイズがあまり大きくならないように、設定を 1 分にしておくと安全です。これは、書き込みトラフィックの急増度や、パイプラインの OpenSearch Compute Unit (OCU) 設定によって異なる場合があります。

  レプリケーションの遅延が 24 時間を超えると、ストリームはイベントをドロップし始め、インデックスを最初から完全に再構築しない限り、精度の問題が発生します。

スケーリング

- パイプラインに自動スケーリングを使用すると、ワークロードに合わせて OCU をスケールアップまたはスケールダウンできます。
- 自動スケーリングを使用しないプロビジョニングされたスループットテーブルでは、書き込みキャパシティユニット (WCU) を 1000 で割った値に基づいて OCU を設定することをお勧めします。最小値をその値より 1 OCU 低く (ただし 1 以上)、最大値をその値を 1 OCU 上回るように設定します。

  - 計算式:

    ```
    OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
    OCU_maximum = (table_WCU / 1000) + 1
    ```

  - 例: あるテーブルでは 25000 の WCU がプロビジョニングされています。パイプラインの OCU は、最小値を 24 (25000/1000 - 1) に、最大値を 26 (25000/1000 + 1) 以上に設定する必要があります。

- 自動スケーリングを使用するプロビジョニングされたスループットテーブルでは、最小および最大 WCU を 1000 で割った値に基づいて OCU を設定することをお勧めします。最小値を DynamoDB の最小値より 1 OCU 低く設定し、最大値を DynamoDB の最大値より少なくとも 1 OCU 高く設定します。

  - 計算式:

    ```
    OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
    OCU_maximum = (table_maximum_WCU / 1000) + 1
    ```

  - 例: あるテーブルには、最小 8000、最大 14000 に設定された自動スケーリングポリシーがあります。パイプラインの OCU は、最小値を 7 (8000/1000 - 1) に、最大値を 15 (14000/1000 + 1) に設定する必要があります。

- オンデマンドスループットテーブルでは、1 秒あたりの書き込みリクエストユニット数の一般的な最大値と最小値に基づいて OCU を設定することをお勧めします。利用可能な集計によっては、より長い期間にわたる平均値の算出が必要な場合があります。最小値を DynamoDB の最小値より 1 OCU 低く設定し、最大値を DynamoDB の最大値より少なくとも 1 OCU 高く設定します。

  - 計算式:

    ```
    # Assuming we have writes aggregated at the minute level
    OCU_minimum = GREATEST((min(table_writes_1min) / (60 * 1000)) - 1, 1)
    OCU_maximum = (max(table_writes_1min) / (60 * 1000)) + 1
    ```

  - 例: あるテーブルの書き込みリクエストユニット数の平均最小値は 1 秒あたり 300 で、平均最大値は 4300 です。パイプラインの OCU は、最小値を 1 (300/1000 - 1、ただし 1 以上) に、最大値を 5 (4300/1000 + 1) に設定する必要があります。

- 送信先の OpenSearch Service インデックスのスケーリングに関するベストプラクティスに従ってください。インデックスのスケーリングが十分でない場合、DynamoDB からの取り込みが遅くなり、遅延が発生する可能性があります。

Note

`GREATEST` は、引数のセットを指定すると最大値の引数を返す SQL 関数です。

Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ

このセクションでは、Amazon DynamoDB 内の現在のクォータ (以前は制限と呼ばれていました) について説明します。各クォータは、指定がない限り、リージョン単位で適用されます。

トピック

読み込み/書き込みモードとスループット

テーブルは、オンデマンドモードからプロビジョンドキャパシティモードにいつでも切り替えることができます。キャパシティモード間で複数の切り替えを行う場合は、次の条件が適用されます。

- オンデマンドモードで新しく作成したテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、テーブルの作成タイムスタンプから 24 時間後のみです。
- オンデマンドモードの既存のテーブルは、いつでもプロビジョンドキャパシティモードに切り替えることができます。ただし、オンデマンドモードに戻すことができるのは、オンデマンドへの切り替えを示す最後のタイムスタンプから 24 時間後のみです。

読み込みおよび書き込みキャパシティモード間の切り替えの詳細については、「キャパシティモードを切り替える際の考慮事項」を参照してください。

キャパシティユニットサイズ (プロビジョニングされるテーブルの場合)

最大サイズが 4 KB の項目について、1 つの読み込みキャパシティーユニット = 1 秒あたり 1 回の強力な整合性のある読み込み、あるいは 1 秒あたり 2 回の結果整合性のある読み込み。

最大サイズが 1 KB の項目について、1 つの書き込みキャパシティーユニット = 1 秒あたり 1 回の書き込み

トランザクション読み込みリクエストでは、最大 4 KB の項目を 1 秒あたりに 1 回読み込むために読み込みキャパシティーユニットが 2 個必要です。

トランザクション書き込みリクエストでは、最大 1 KB の項目を 1 秒あたり 1 回書き込むのに書き込みキャパシティーユニットが 2 個必要です。

リクエストユニットサイズ (オンデマンドテーブルの場合)

最大サイズが 4 KB の項目について、1 つの読み込みリクエストユニット = 1 秒あたり 1 回の強力な整合性のある読み込み、あるいは 1 秒あたり 2 回の結果整合性のある読み込み。

最大サイズが 1 KB の項目について、1 つの書き込みリクエストユニット = 1 秒単位の書き込み。

トランザクション読み込みリクエストでは、最大 4 KB の項目を 1 秒に 1 回読み込むのに読み込みリクエストユニットが 2 個必要です。

トランザクション書き込みリクエストでは、1 KB の項目を 1 秒に 1 回書き込むのに書き込みリクエストユニットが 2 個必要です。

スループットのデフォルトクォータ

AWS では、リージョン内でアカウントがプロビジョニングおよび利用できるスループットについて、いくつかのデフォルトのクォータがあります。

アカウントレベルの読み取りスループットとアカウントレベルの書き込みスループットクォータは、アカウントレベルで適用されます。これらのアカウントレベルのクォータは、特定のリージョン内のすべてのアカウントテーブルとグローバルセカンダリインデックスのプロビジョニングされたスループットキャパシティーの合計に適用されます。アカウントで使用可能なスループットをすべて 1 つのテーブルにプロビジョニングすることも、複数のテーブルに分けてプロビジョニングすることもできます。一これらのクォータは、プロビジョンドキャパシティーモードを使用するテーブルにのみ適用されます。

プロビジョンドキャパシティーモードを使用するテーブルと、オンデマンドキャパシティーモードを使用するテーブルに適用されるテーブルレベルの読み込みスループットのクォータとテーブルレベルの書き込みスループットクォータは、それぞれ異なります。

プロビジョンドキャパシティーモードのテーブルと GSI の場合、クォータは、リージョン内の任意のテーブルまたはその GSI にプロビジョニングできる読み取りおよび書き込みキャパシティユニットの最大数です。個々のテーブルとそのすべての GSI の合計も、アカウントレベルの読み取り/書き込みスループットクォータを下回っている必要があります。これに加えて、プロビジョニングされたすべてのテーブルとその GSI の合計がアカウントレベルの読み取り/書き込みスループットクォータを下回っている必要があります。

オンデマンドキャパシティモードテーブルと GSI の場合、テーブルレベルのクォータは、任意のテーブル、またはそのテーブル内の個々の GSI で使用可能な最大読み取り/書き込みキャパシティユニットです。アカウントレベルの読み取り/書き込みスループットクォータは、オンデマンドモードのテーブルには適用されません。

アカウントにデフォルトで適用されるスループットのクォータは以下のとおりです。

	オンデマンド	プロビジョンド	調整可能
`Per table`	`40,000 read request units`	`40,000 read capacity units`	はい

	オンデマンド	プロビジョンド	調整可能
	and 40,000 write request units	and 40,000 write capacity units	
Per account	Not applicable	80,000 read capacity units and 80,000 write capacity units	はい
Minimum throughput for any table or global secondary index	Not applicable	1 read capacity unit and 1 write capacity unit	はい

Service Quotas コンソール、AWS API および AWS CLI を使用して、調整可能なクォータのクォータ増加のリクエストを行うことができます。

アカウントレベルのスループットクォータについては、Service Quotas コンソール、AWSCloudWatch コンソール、AWS API および AWS CLI を使用して、CloudWatch アラームを作成し、現在の使用量が適用されたクォータ値の指定した割合に達すると、自動的に通知されるようにすることができます。CloudWatch を使用すると、AccountProvisionedReadCapacityUnits および AccountProvisionedWriteCapacityUnits AWS 使用状況メトリクスを確認して、使用状況をモニタリングすることもできます。使用状況に関するメトリクスの詳細については、「AWS 使用状況メトリクス」を参照してください。

スループットの増加または減少 (プロビジョニングされたテーブルの場合)

プロビジョニングされたスループットの増加

ReadCapacityUnits または WriteCapacityUnits オペレーションを使用して、必要な回数だけ AWS Management Console または UpdateTable を増やすことができます。1 回の呼び出しで、テーブル、そのテーブルの任意のグローバルセカンダリインデックス、またはこれらの任意の組み合わせに対して、プロビジョニングされるスループットを増やすことができます。新しい設定は、UpdateTable オペレーションが完了するまでは有効になりません。

プロビジョニングされたキャパシティーを追加する場合、アカウントごとのクォータを超えることはできません。また、DynamoDB では、プロビジョニングされたキャパシティーを急速に増やすことはできません。これらの制限に達しない限り、テーブルのプロビジョニング容量を必要なだけ増やすことができます アカウントごとのクォータの詳細については、前述の「スループットのデフォルトクォータ」セクションを参照してください。

プロビジョニングされたスループットの減少

UpdateTable オペレーションのすべてのテーブルとグローバルセカンダリインデックスでは、ReadCapacityUnits か WriteCapacityUnits (またはその両方) を減らすことができます。新しい設定は、UpdateTable オペレーションが完了するまでは有効になりません。

1 日あたりの DynamoDB テーブルで実行できるプロビジョンドキャパシティーの減少数には、デフォルトのクォータがあります。日付は、協定世界時 (UTC) に従って定義されます。特定の日に、その日に他の減少をまだ実行していない限り、1 時間以内に最大 4 回の減少を実行することから始めることができます。その後、1 時間あたり 1 回追加で減少を実行できます (60 分に 1 回)。これにより、1 日の最大減少回数は 27 回になります。

必要に応じて、Service Quotas コンソール、AWS API および AWS CLI を使用して、クォータ増加のリクエストを行うことができます。

> ⚠ Important
>
> テーブルとグローバルセカンダリインデックスの減少制限は別々に設定されているため、特定のテーブルのグローバルセカンダリインデックスにはいずれも、独自の減少制限が設定されています。ただし、1 つのリクエストでテーブルとグローバルセカンダリインデックスのスループットを縮小し、いずれかが現在の制限を超えた場合は拒否されます。リクエストが部分的に処理されることはありません。

Example

1 日の最初の 4 時間で、グローバルセカンダリインデックスが設定されているテーブルは次のように変更できます。

- テーブルの WriteCapacityUnits か ReadCapacityUnits (または両方) を 4 時間減らします。
- グローバルセカンダリインデックスの WriteCapacityUnits か ReadCapacityUnits (または両方) を 4 時間減らします。

同じ日の終わりに、テーブルとグローバルセカンダリインデックスのスループットは、合計 27 回ずつ縮小することができる可能性があります。

リザーブドキャパシティ

AWS アカウントで購入できるアクティブなリザーブドキャパシティの量に、デフォルトのクォータを設定します。クォータ制限は、書き込みキャパシティユニット (WCU) と読み取りキャパシティユニット (RCU) のリザーブドキャパシティの組み合わせです。

	アクティブなリザーブドキャパシティ	調整可能
アカウントごと	1,000,000 のプロビジョンドキャパシティユニット (WCU _ RCU)	はい

1 回の購入で 1,000,000 を超えるプロビジョニング済みキャパシティユニットを購入しようとすると、このサービスクォータ制限に関するエラーが表示されます。アクティブなリザーブドキャパシティがあり、追加のリザーブドキャパシティを購入しようとして、アクティブなプロビジョンドキャパシティユニットが 1,000,000 ユニットを超える場合、このサービスクォータ制限に関するエラーが表示されます。

1,000,000 を超えるリザーブドキャパシティの引き上げが必要な場合は、サポートチームにリクエストを送信してクォータの引き上げをリクエストできます。

インポートクォータ

us-east-1、us-west-2、および eu-west-1 の各リージョンでは、Amazon S3 からの DynamoDB のインポートは、一度に 15 TB の合計インポートソースオブジェクトサイズで 50 個の同時インポートジョブに対応できます。他のすべてのリージョンでは、合計サイズが 1 TB で最大 50 個の同時インポートタスクに対応できます。各インポートジョブでは、すべてのリージョンで最大 50,000 個の Amazon S3 オブジェクトを使用できます。インポートと検証の詳細については、「インポート形式の割り当てと検証」を参照してください。

Contributor Insights

DynamoDB テーブルで Customer Insights を有効にする場合でも、Contributor Insights のルール制限が適用されます。詳細については、「CloudWatch サービスのクォータ」を参照してください。

テーブル

テーブルのサイズ

テーブルのサイズには実用的な制限はありません。テーブルは項目数やバイト数について制限がありません。

1 アカウント (1 リージョン) あたりのドメインの最大数

AWS アカウントについては、AWS リージョンごとに 2,500 個のテーブルという初期クォータがあります。

1 つのアカウントで2,500を超えるテーブルが必要な場合は、AWS アカウントチームに連絡して、最大 10,000 テーブルまで増やすことを検討してください。10,000 を超える場合のベストプラクティスは、複数のアカウントを設定することです。各アカウントは最大 10,000 個のテーブルに対応できます。

Service Quotas コンソール、AWS API および AWS CLI を使用して、アカウントの最大テーブル数に対するデフォルトおよび適用されたクォータ値を確認し、必要に応じてクォータ増加のリクエストを行うことができます。AWS サポートにチケットを申請してクォータの増加をリクエストすることもできます

Service Quotas コンソール、AWS API および AWS CLI を使用すると、CloudWatch アラームを作成し、現在の使用量が適用されたクォータ値の指定した割合に達すると、自動的に通知されるようにすることができます。CloudWatch を使用すると、`TableCount` AWS 使用状況メトリクスを確認して、使用状況をモニタリングすることもできます。使用状況に関するメトリクスの詳細については、「AWS 使用状況メトリクス」を参照してください。

グローバルテーブル

AWSでは、グローバルテーブルを使用するときにプロビジョンまたは利用できるスループットについて、いくつかのデフォルトのクォータがあります。

	オンデマンド	プロビジョンド
Per table	40,000 read request units and 40,000 write request units	40,000 read capacity units and 40,000 write capacity units
Per table, per destination Region, per day	10 TB for all source tables to which a replica was added for this destination Region	10 TB for all source tables to which a replica was added for this destination Region

トランザクションオペレーションは、書き込みが最初に行われた AWS リージョン内でのみ、不可分性、一貫性、分離性、および耐久性 (ACID) を保証します。グローバルテーブルのリージョン間では、トランザクションはサポートされていません。たとえば、米国東部 (オハイオ) リージョンと米国西部 (オレゴン) リージョンにレプリカを含むグローバルテーブルがあり、米国東部 (バージニア北部) リージョンで TransactWriteItems オペレーションを実行するとします。この場合、変更がレプリケートされると、米国西部 (オレゴン) リージョンで部分的に完了したトランザクションを確認できます。変更は、ソースリージョンでコミットされた後でのみ、他のリージョンにレプリケートされます。

Note

場合によっては、AWS Support を通じてクォータ制限の引き上げをリクエストする必要が生じることがあります。以下のいずれかに該当する場合は、https://aws.amazon.com/support を参照してください。

- 40,000 を超える書き込みキャパシティユニット (WCU) を使用するように設定されているテーブルのレプリカを追加する場合は、レプリカの追加 WCU クォータのサービスクォータの引き上げをリクエストする必要があります。
- 24 時間以内に 1 つまたは複数のレプリカを 1 つの送信先リージョンに追加し、合計が 10 TB を超える場合は、レプリカの追加データのバックフィルクォータについてサービスクォータの引き上げをリクエストする必要があります。
- 次のようなエラーが発生した場合は、サービスクォータの引き上げをリクエストする必要があります。

- リージョン「example_region_B」の現在のアカウント制限を超えるため、リージョン「example_region_A」でテーブル「example_table」のレプリカを作成できません。

セカンダリインデックス

テーブルごとのセカンダリインデックス

最大 5 つのローカルセカンダリインデックスを定義することができます。

デフォルトクォータとして、テーブルごとに 20 個のグローバルセカンダリインデックスがあります。Service Quotas コンソール、AWS API および AWS CLI を使用して、アカウントに適用されるテーブルごとのグローバルセカンダリインデックスのデフォルトと現在のクォータを確認し、必要に応じてクォータ増加のリクエストを行うことができます。https://aws.amazon.com/support にチケットを申請してクォータの増加をリクエストすることもできます。

UpdateTable オペレーションごとに、グローバルセカンダリインデックスを1 つだけ作成または削除できます。

テーブルごとの射影されたセカンダリインデックスの属性

合計最大 100 の属性を、1 つのテーブルのすべてのグローバルセカンダリインデックスに射影することができます。これは、ユーザー指定の射影された属性だけに適用されます。

CreateTable オペレーションでは、INCLUDE として ProjectionType を指定した場合には、NonKeyAttributes で指定した、すべてのローカルセカンダリインデックスを含む属性の合計数が 100 を超えてはいけません。同じ属性名を 2 つの異なるインデックスに射影した場合には、合計を計算する際に 2 つの異なる属性として計算されます。

この制限は、ProjectionType が KEYS_ONLY または ALL であるセカンダリインデックスには適用されません。

パーティションキーおよびソートキー

パーティションキーの長さ

パーティションキーと値の最小長は 1 バイトです。最大長は 2048 バイトです

パーティションキーの値

テーブルまたはセカンダリインデックスについて、パーティションキー値の明確な数に関する実質的な制限はありません。

ソートキーの長さ

ソートキーと値の最小長は 1 バイトです。最大長は 1024 バイトです

ソートキー値

一般的に、パーティションキーの値ごとのソートキーの値の数について、実質的に制限はありません。

セカンダリインデックスを持つテーブルは例外です。項目コレクションは、パーティションキー属性の値が同じ項目のセットです。グローバルセカンダリインデックスでは、項目コレクションはベーステーブルから独立しています (そのため、異なるパーティションキー属性を持つこともできます)。ただし、ローカルセカンダリインデックスでは、インデックス付きビューはテーブル内の項目と同じパーティションに配置され、同じパーティションキー属性を共有します。このローカリティのため、テーブルに 1 つ以上の LSI がある場合、項目コレクションを複数のパーティションに分散することはできません。

1 つ以上の LSI を含むテーブルの場合、項目コレクションのサイズは 10 GB を超えることはできません。これには、パーティションキー属性の値が同じであるすべてのベーステーブル項目およびすべての射影された LSI ビューが含まれます。パーティションの最大サイズは 10 GB です。詳細については、「項目コレクションのサイズ制限」を参照してください。

名前付けルール

テーブル名とセカンダリインデックス名

テーブルとセカンダリインデックスの名前は、3 文字以上、255 文字以下である必要があります。使用可能な文字は次のとおりです。

- `A-Z`
- `a-z`
- `0-9`

- _ (下線)
- - (ハイフン)
- . (ドット)

属性名

一般的に、属性名は 1 文字以上、64 KB 以下である必要があります。

以下の例外があります。これらの属性名は 255 文字以下である必要があります。

- セカンダリインデックスのパーティションキー名。
- セカンダリインデックスのソートキー名。
- ユーザー指定の射影された属性の名前 (ローカルセカンダリインデックスのみに適用)。`CreateTable` オペレーションでは、`ProjectionType` を `INCLUDE` に指定した場合には、`NonKeyAttributes` パラメータの属性名の長さが制限されます。射影タイプ `KEYS_ONLY` および `ALL` には影響しません。

これらの属性名は UTF-8 を使用してエンコードする必要があり、それぞれの名前の合計サイズが (エンコード後に) 255 バイトを超えることはできません。

データ型

文字列

文字列の長さは、項目の最大サイズである 400 KB に制約されます。

文字列は、UTF-8 バイナリエンコードの Unicode です。UTF-8 は可変幅のエンコードであるため、DynamoDB により、UTF-8 バイトを使用して文字列の長さが決定されます。

数

数値は、最大 38 桁の精度であり、正、負、または 0 のいずれかです。

- 正の範囲: `1E-130`～`9.9999999999999999999999999999999999999E+125`
- 負の範囲: `-9.9999999999999999999999999999999999999E+125`～`-1E-130`

DynamoDB では、リクエストと返信の数値データが JSON 文字列で表示されます。詳細については、「DynamoDB 低レベル API」を参照してください。

数値の精度が重要な場合は、数値型から変換する文字列を使用して、DynamoDB に数値を渡します。

バイナリ

バイナリの長さは、項目の最大サイズである 400 KB に制約されます。

バイナリ属性を操作するアプリケーションは、データを DynamoDB に送信する前に、base64 形式でエンコードする必要があります。その後 DynamoDB は、受信したデータを署名なしバイト配列にデコードし、それを属性の長さとして使用します。

項目

項目のサイズ

DynamoDB の項目の最大サイズは 400 KB で、属性名のバイナリの長さ (UTF-8 の長さ) と属性値の長さ (こちらもバイナリの長さ) を含みます。属性名はサイズ制限に反映されます。

たとえば、2 つの属性を持つ項目があり、1 つの属性は名前が "shirt-color" で値が "R"、別の属性は名前が "shirt-size" で値が "M" であるとします。この項目の合計サイズは 23 バイトです。

ローカルセカンダリインデックスを持つテーブルの項目のサイズ

テーブルのローカルセカンダリインデックスごとに、次を合計したサイズに関して 400 KB の制限があります。

- テーブルの項目データのサイズ。
- すべてのローカルセカンダリインデックスエントリ (キーの値と射影された属性を含む) の対応するエントリサイズ。

属性

項目あたりの属性名と値のペア

項目あたりの属性の累積サイズは、DynamoDB の項目の最大サイズ (400 KB) 以内である必要があります。

リスト、マップ、またはセットの値の数

値を含む項目が 400 KB の制限内である限り、リスト、マップ、またはセットにおける値の最大数の制限はありません。

属性値

属性がテーブルまたはインデックスのキー属性として使用されていない場合は、空の文字列属性とバイナリ属性値を使用できます。空の文字列とバイナリ値は、セット、リスト、およびマップ型内で許可されます。属性値は空のセット (文字列セット、数値セット、またはバイナリセット) にすることはできません。ただし、空のリストおよびマップは許可されます。

入れ子の属性の深さ

DynamoDB では、深さが最大 32 のレベルの入れ子の属性をサポートします。

式パラメータ

式パラメータには、`ProjectionExpression`、`ConditionExpression`、`UpdateExpression`、および `FilterExpression` があります。

長さ

任意の式の最大長は 4 KB です。たとえば、`ConditionExpression` a=b のサイズは 3 バイトです。

1 つの式属性名または式属性値の最大長は 255 バイトです。たとえば、`#name` は 5 バイト、`:val` は 4 バイトです。

式のすべての置換変数の最大長は 2 MB です。これはすべての `ExpressionAttributeNames` および `ExpressionAttributeValues` の長さの合計です。

演算子およびオペランド

`UpdateExpression` で許容される演算子または関数の最大数は 300 です。たとえば、UpdateExpression `SET a = :val1 + :val2 + :val3` は 2 つの「+」演算子を含みます。

`IN` コンパレータのオペランドの最大数は 100 です。

予約語

DynamoDB では、予約語と競合する名前の使用を防ぐことはできません。(詳細な一覧については、「DynamoDB の予約語」を参照してください)。

ただし、式パラメータで予約語を使用する場合は、`ExpressionAttributeNames` も指定する必要があります。詳細については、「DynamoDB の式の属性名」を参照してください。

DynamoDB のトランザクション

DynamoDB トランザクションの API オペレーションには次の制約があります。

- トランザクションには、100 個を超える一意のアクションを含めることはできません。
- トランザクションには、4 MB を超えるデータを含めることはできません。
- トランザクション内の 2 つのアクションを、同じテーブルの同じ項目に対して実行することはできません。たとえば、`ConditionCheck` と `Update` の両方を 1 つのトランザクションで同じ項目に対して実行することはできません。
- トランザクションは、複数の AWS アカウントまたはリージョンのテーブルで動作できません。
- トランザクションオペレーションは、書き込みが最初に行われた AWS リージョン内でのみ、不可分性、一貫性、分離性、および耐久性 (ACID) を保証します。グローバルテーブルのリージョン間では、トランザクションはサポートされていません。たとえば、米国東部 (オハイオ) リージョンと米国西部 (オレゴン) リージョンにレプリカを含むグローバルテーブルがあり、米国東部 (バージニア北部) リージョンで `TransactWriteItems` オペレーションを実行するとします。この場合、変更がレプリケートされると、米国西部 (オレゴン) リージョンで部分的に完了したトランザクションを確認できます。変更は、ソースリージョンでコミットされると、他のリージョンにのみレプリケートされます。

DynamoDB Streams

DynamoDB Streams でのシャードの同時読み込み

単一リージョンのテーブルがグローバルテーブルでない場合、同じ DynamoDB Streams のシャードから、同時に 2 つまでのプロセスを読み込むように設計できます。この制限を超えると、リクエストのスロットリングが発生する場合があります。グローバルテーブルでは、リクエストのスロットリングを回避するために、同時リーダーの数を 1 に制限することをお勧めします。

DynamoDB Streams が有効なテーブルの最大書き込みキャパシティ

AWS では、DynamoDB Streams が有効な DynamoDB テーブルの書き込み容量について、いくつかのデフォルトのクォータを用意しています。これらのデフォルトクォータは、プロビジョニングされた読み取り/書き込みキャパシティーモードのテーブルにのみ適用されます。アカウントにデフォルトで適用されるスループットのクォータは以下のとおりです。

- 米国東部 (バージニア北部)、米国東部 (オハイオ)、米国西部 (オレゴン)、南米 (サンパウロ)、欧州 (フランクフルト)、欧州 (アイルランド)、アジアパシフィック (東京)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、中国 (北京) リージョン:
  - テーブル単位 – 40,000 個の書き込みキャパシティーユニット
- その他すべてのリージョン:
  - テーブル単位 – 10,000 個の書き込みキャパシティーユニット

Service Quotas コンソール、AWS API および AWS CLI を使用して、アカウントに適用される DynamoDB Streams が有効なテーブルの最大書き込みキャパシティーと、アカウントに適用される現在のクォータを確認し、必要に応じてクォータの増加をリクエストできます。AWS サポートにチケットを申請してクォータの増加をリクエストすることもできます

> Note
>
> プロビジョニングされたスループットクォータは、DynamoDB Streams が有効な DynamoDB テーブルにも適用されます。Streams が有効になっているテーブルの書き込みキャパシティのクォータの増加をリクエストする場合は、このテーブルのプロビジョニングされたスループットキャパシティの増加もリクエストしてください。詳細については、「スループットのデフォルトクォータ」を参照してください。他のクォータは、より高いスループットの DynamoDB Streams を処理する場合にも適用されます。詳細については、「Amazon DynamoDB Streams API リファレンスガイド」を参照してください。

DynamoDB Accelerator (DAX)

AWS を利用可能なリージョン

DAX を使用できる AWS リージョンのリストについては、「AWS 全般のリファレンス」の「DynamoDB アクセラレーター (DAX)」を参照してください。

ノード

DAX クラスターは、1 つのみのプライマリノードと、0～10 個のリードレプリカノードで構成されます。

ノードの総数 (AWS アカウントごと) は、1 つの AWS リージョン内で 50 を超えることはできません。

パラメータグループ

リージョンごとに最大 20 の DAX パラメータグループを作成できます。

[サブネットグループ]

リージョンごとに最大 50 の DAX サブネットグループを作成できます。

サブネットグループ内では、最大 20 のサブネットを定義できます。

API 固有の制限

`CreateTable`/`UpdateTable`/`DeleteTable`/`PutResourcePolicy`/`DeleteResourcePolicy`

一般に、CreateTable、UpdateTable、DeleteTable、PutResourcePolicy、DeleteResourcePolicy のリクエストは、どのような組み合わせでも最大 500 件まで同時に実行できます。その結果、`CREATING`、`UPDATING`、または `DELETING` の状態のテーブルの合計数が 500 を超えることはできません。

テーブルグループ全体で任意の組み合わせで、ミュータブル (`CreateTable`、`DeleteTable`、`UpdateTable`、`PutResourcePolicy`、`DeleteResourcePolicy`) コントロールプレーン API リクエストを 1 秒あたり最大 2,500 回送信できます。ただし、`PutResourcePolicy` リクエストおよび `DeleteResourcePolicy` リクエストにはそれぞれ下限があります。詳細については、`PutResourcePolicy` と `DeleteResourcePolicy` のクォータに関する以下の詳細を参照してください。

また、リソースベースのポリシーを含む `CreateTable` リクエストおよび `PutResourcePolicy` リクエストは、ポリシーの KB ごとに 2 つの追加リクエストとしてカウントされます。例えば、サイズが 5 KB のポリシーの `CreateTable` リクエストまたは `PutResourcePolicy` リクエストは 11 リクエストとしてカウントされます。1 つは

CreateTable リクエスト、10 はリソースベースのポリシー (2 x 5 KB) です。同様に、サイズが 20 KB のポリシーは 41 件のリクエストとしてカウントされます。1 つはリクエストで、40 はリソースベースのポリシー (2 x 20 KB) です。

PutResourcePolicy

1 つのテーブルで 1 秒あたり最大 25 個の PutResourcePolicy API リクエストを送信できます。個々のテーブルのリクエストが成功すると、その後の 15 秒間、新しい PutResourcePolicy リクエストはサポートされません。

リソースベースのポリシードキュメントでサポートされる最大サイズは 20 KB です。DynamoDB では、この上限に照らしてポリシーのサイズを計算する際に空白はカウントされません。

DeleteResourcePolicy

1 つのテーブルで 1 秒あたり最大 50 個の DeleteResourcePolicy API リクエストを送信できます。個々のテーブルの PutResourcePolicy リクエストが成功すると、その後の 15 秒間、DeleteResourcePolicy リクエストはサポートされません。

BatchGetItem

1 回の BatchGetItem オペレーションで、最大 100 項目を取得できます。取得するすべての項目の合計サイズが 16 MB を超えてはいけません。

BatchWriteItem

1 回の BatchWriteItem オペレーションでは、最大 25 の PutItem、または DeleteItem リクエストを含むことができます。書き込むすべての項目の合計サイズが 16 MB を超えてはいけません。

DescribeStream

DescribeStream は最大で毎秒 10 回呼び出すことができます。

DescribeTableReplicaAutoScaling

DescribeTableReplicaAutoScaling メソッドでは、1 秒あたり 10 リクエストのみサポートされます。

DescribeLimits

DescribeLimits は定期的に呼び出すのみにします。1 分以内に複数回呼び出すと、スロットリングエラーが発生する可能性があります。

DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights

DescribeContributorInsights、ListContributorInsights、およびUpdateContributorInsights は定期的に呼び出すのみにします。DynamoDB では、これらの API のそれぞれで 1 秒あたり最大 5 つのリクエストがサポートされます。

DescribeTable/ListTables/GetResourcePolicy

読み取り専用 (DescribeTable、ListTables、GetResourcePolicy) コントロールプレーン API リクエストを組み合わせて、1 秒あたり最大 2,500 回送信できます。GetResourcePolicy API には、1 秒あたり 100 リクエストという個別の下限があります。

Query

Query の結果セットは、1 回の呼び出しあたり 1 MB に制限されます。クエリ応答から LastEvaluatedKey を使用して、結果をさらに取り出すこともできます。

Scan

Scan の結果セットは、1 回の呼び出しあたり 1 MB に制限されます。スキャン応答から LastEvaluatedKey を使用して、結果をさらに取り出すこともできます。

UpdateKinesisStreamingDestination

UpdateKinesisStreamingDestination オペレーションを実行する場合、24 時間に最大 3 回、ApproximateCreationDateTimePrecision を新しい値に設定できます。

UpdateTableReplicaAutoScaling

UpdateTableReplicaAutoScaling メソッドでは、1 秒あたり 10 リクエストのみサポートされます。

`UpdateTableTimeToLive`

> `UpdateTableTimeToLive` メソッドでは、指定されたテーブルごとに、`Time to Live (TTL)` の有効化または無効化のリクエストが 1 時間あたり 1 つだけサポートされます。この変更が完全に処理されるまでに最大で 1 時間かかる場合があります。この 1 時間の間に同じテーブルに対して追加の `UpdateTimeToLive` 呼び出しが行われると、ValidationException が発生します。

保管時の DynamoDB 暗号化

AWS 所有のキー、AWS マネージドキー、カスタマーマネージドキーは 24 時間いつでも切り替えることができます。テーブルの作成時から、テーブル単位で最大 4 回まで切り替え可能です。また、過去 6 時間以内に変更がなかった場合は、追加で変更することができます。これにより、1 日で変更できる最大の回数は 8 回になります (1 日の中で最初の 6 時間は 4 回、その後は 6 時間ごとに 1 回)。

AWS 所有のキー を使用する暗号化キーの切り替えは、上述のクォータを使い尽くしても、必要な回数だけ行うことができます。

クォータの拡大をリクエストしない限り、以下のクォータが適用されます。サービスクォータの増加をリクエストするには、https://aws.amazon.com/support を参照してください。

Amazon S3 へのテーブルのエクスポート

フルエクスポート: 最大 300 個の同時エクスポートタスク、またはすべての処理中のテーブルエクスポートから合計 100 TB をエクスポートできます。これらの制限は両方とも、エクスポートがキューに入る前に確認されます。

増分エクスポート: 15 分 ～ 24 時間のエクスポート期間で、最大 300 の同時ジョブ、つまり 100 TB のテーブルサイズを同時にエクスポートできます。

バックアップと復元

DynamoDB オンデマンドバックアップまたは継続的バックアップを使用してテーブルデータを復元する場合、合計 50 TB の同時復元を最大 50 回実行できます。AWS Backup では、合計 25 TB の同時復元を最大 50 回実行できます。バックアップの詳細については、「DynamoDB のオンデマンドバックアップおよび復元の使用」を参照してください。

低レベル API リファレンス

Amazon DynamoDB API リファレンスには、以下によりサポートされているオペレーションの詳しいリストが記載されています。

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)

Amazon DynamoDB のトラブルシューティング

以下のトピックでは、Amazon DynamoDB の使用時に発生する可能性のあるエラーや問題のトラブルシューティングに関するアドバイスを提供します。ここに記載されていない問題が見つかった場合は、このページの [Feedback] ボタンを使用して報告することができます。

トラブルシューティングに関するアドバイス、およびサポートへの一般的な質問に対する回答については、AWS ナレッジセンターにアクセスしてください。

トピック

- Amazon DynamoDB でのレイテンシーの問題のトラブルシューティング
- プロビジョンドキャパシティモードを使用した DynamoDB テーブルのスロットリング問題

Amazon DynamoDB でのレイテンシーの問題のトラブルシューティング

ワークロードのレイテンシーが高いと思われる場合は、CloudWatch SuccessfulRequestLatency メトリクスを分析し、平均レイテンシーをチェックして、それが DynamoDB に関連しているかどうかを確認できます。報告された SuccessfulRequestLatency にある程度のばらつきがあるのは正常であり、時折 (特に Maximum 統計) 急上昇が起きても心配する必要はありません。ただし、Average 統計が急激に増加し続けている場合は、AWS Service Health Dashboard と Personal Health Dashboard で詳細を確認する必要があります。考えられる原因としては、テーブル内の項目のサイズ (1 KB の項目と 400 KB の項目ではレーテンシーが異なります) やクエリのサイズ (10 項目と 100 項目) などがあります。

必要に応じて、AWS Support でサポートケースを作成することを検討し、作成したランブックに従って、アプリケーションで使用可能なフォールバックオプション (マルチリージョンアーキテクチャの場合はリージョンの退避など) を引き続き評価してください。リクエストが遅い場合は、サポートケースを開くときに AWS Supportに提示できるようにリクエスト ID を記録しておく必要があります。

SuccessfulRequestLatency メトリックスは DynamoDB サービス内部のレイテンシーのみを測定し、クライアント側のアクティビティとネットワークトリップ時間は含まれません。クライアントから DynamoDB サービスへの呼び出しの全体的なレイテンシーについて詳しく知るには、AWS SDK でレイテンシーメトリックスロギングを有効にできます。

> **Note**
>
> ほとんどのシングルトンオペレーション (プライマリキーの値を完全に指定することで 1 つの項目に適用されるオペレーション) では、DynamoDB は 1 桁のミリ秒単位の `Average SuccessfulRequestLatency` を返します。この値には、DynamoDB エンドポイントにアクセスする呼び出し側コードのトランスポートオーバーヘッドは含まれていません。複数項目のデータオペレーションの場合、レーテンシーは結果セットのサイズ、返されるデータ構造の複雑さ、適用される条件式やフィルター式などの要因によって異なります。同じパラメータで同じデータセットに対して複数項目のオペレーションを繰り返す場合、DynamoDB は一貫した高い `Average SuccessfulRequestLatency` を提供します。

レイテンシーを減らすには、次の戦略を 1 つ以上検討してください。

- リクエストのタイムアウトと再試行動作の調整:クライアントから DynamoDB へのパスは多くのコンポーネントを通過しますが、それぞれが冗長性を念頭に置いて設計されています。ネットワークの回復力の範囲、TCP パケットタイムアウト、DynamoDB 自体の分散アーキテクチャについて考えてみてください。デフォルトの SDK の動作は、ほとんどのアプリケーションに対して適切なバランスを取るように設計されています。最善のレイテンシーを最優先する場合は、SDK のデフォルトのリクエストタイムアウトと再試行設定を調整し、クライアントで測定されたリクエストの成功までの一般的なレイテンシーに合わせることを検討してください。通常よりも大幅に時間がかかるリクエストは、最終的に成功する可能性が低くなります。フェイルファストしてから新しいリクエストを行うと、別のパスを使用してすぐに成功する可能性があります。これらの設定を積極的に使用し過ぎると、マイナス面がある場合があることにも注意してください。このトピックに関する役立つ説明は、「レイテンシーを考慮した Amazon DynamoDB アプリケーションのための AWS Java SDK HTTP リクエスト設定のチューニング」に記載されています。
- クライアントと DynamoDB エンドポイント間の距離を縮める:ユーザーが世界中に分散している場合は、グローバルテーブル – DynamoDB の複数リージョンレプリケーション を使用することを検討してください。グローバルテーブルを使用では、そのテーブルを利用できるようにする AWS リージョンを指定できます。ローカルのグローバルテーブルレプリカからデータを読み込むと、ユーザーのレーテンシーを大幅に減らすことができます。また、DynamoDB ゲートウェイエンドポイントを使用して、クライアントトラフィックを VPC 内に留ることも検討してください。
- キャッシュを使用する:トラフィックの読み取り量が多い場合は、DynamoDB Accelerator (DAX) とインメモリアクセラレーション などのキャッシュサービスの使用を検討してください。DAX は、フルマネージド型で可用性の高い、DynamoDB 用のインメモリキャッシュです。1 秒あたりのリクエスト数が数百万におよぶ場合であっても、最大 10 倍のパフォーマンスの (ミリセカンドからマイクロセカンドの範囲への) 向上を実現します。

- 接続の再利用:DynamoDB リクエストは、デフォルトで HTTPS に設定されている認証済みセッションを介して行われます。接続の開始には時間がかかるため、最初のリクエストのレーテンシーは通常よりも長くなります。すでに初期化されている接続でリクエストを行うと、DynamoDB は整合性のある低いレイテンシーを実現できます。このため、新しい接続を確立するまでのレーテンシーを避けるために、他にリクエストが行われない場合は 30 秒ごとに「キープアライブ」`GetItem` リクエストを行うことをお勧めします。
- 結果整合性のある読み込みを使用する:アプリケーションで強い整合性のある読み取りが必要ない場合は、デフォルトの結果整合性のある読み込みを使用することを検討してください。結果整合性のある読み込みを行うと、コストが低くなり、レイテンシーが一時的に増加する可能性も低くなります。詳細については、「読み込み整合性」を参照してください。

プロビジョンドキャパシティモードを使用した DynamoDB テーブルのスロットリング問題

テーブルまたはインデックスでプロビジョニングされたスループットキャパシティを超過したアプリケーションは、リクエストスロットリングの対象になります。スロットリングは、アプリケーションで大量のキャパシティーユニットが消費されるのを防ぎます。DynamoDB は、読み込みや書き込みのオペレーションをスロットリングすると、発信者に `ProvisionedThroughputExceededException` を返します。その後、アプリケーションは、リクエストの再試行前に短時間待機するなど、適切なアクションを実行できます。

このトピックでは、一般的なスロットリング問題のトラブルシューティング方法と CloudWatch を使用して問題の発生元を調査する方法について説明します。

トピック

スロットリング問題のトラブルシューティング

スロットリングに関連していると思われる問題のトラブルシューティングでは、まずスロットリングの発生元が DynamoDB とアプリケーションのいずれであるかを確認することが重要です。

一般的なシナリオと、その解決に役立つ手順を以下に示します。

DynamoDB テーブルには十分なプロビジョンドキャパシティがあるようでも、リクエストがスロットリングされる

この問題は、スループットが 1 分あたりでは平均未満でも、1 秒あたりでは使用可能な量を超えている場合に発生することがあります。DynamoDB は分単位のメトリクスのみを CloudWatch に報告し、メトリクスは 1 分間の合計として計算されて平均化されます。ただし、DynamoDB 自体のレート制限は 1 秒単位で適用されます。このため、その 1 分間の範囲でわずかな時間 (たとえば数秒未満) であっても、発生したスループットが多すぎると、その範囲の残りのリクエストが抑制される可能性があります。

例えば、テーブルに 60 WCU がプロビジョニングされている場合、1 分間に 3,600 回の書き込みオペレーションを実行できます。ただし、1 秒で 3600 個の WCU リクエストがすべて実行された場合、残りの時間にスロットリングが発生します。

このシナリオを解決する 1 つの方法は、API 呼び出しに多少のジッターとエクスポネンシャルバックオフを追加することです。詳細については、バックオフとジッターに関するこの投稿を参照してください。

自動スケーリングが有効になっていても、テーブルがスロットリングされる

これは、トラフィックでの突然のスパイク時に発生する可能性があります。自動スケーリングは、2 つのデータポイントが 1 分の間にターゲット使用率の設定値を超えた場合にトリガーされます。したがって、消費された容量が目標使用率よりも高い状態が 2 分間継続すると、自動スケーリングが実行されます。ただし、スパイクの間隔が 1 分を超えると、自動スケーリングはトリガーされない場合があります。

同様に、15 個の連続するデータポイントが目標使用率を下回ると、スケールダウンイベントがトリガーされます。いずれの場合でも、自動スケーリングのトリガー後に `UpdateTable` API オペレーションが呼び出されます。この呼び出しは、テーブルまたはインデックスのプロビジョンドキャパシティを更新するのに数分かかる場合があります。この期間中、テーブルの前のプロビジョンドキャパシティを超えるリクエストはスロットルされます。

つまり、自動スケーリングで DynamoDB テーブルをスケールアップするには、連続したデータポイントでターゲット使用率の値を超える必要があります。このため、自動スケーリングは、スパイキーなワークロードに対処するためのソリューションとしては推奨されません。詳細については、自動スケーリングによるコスト最適化に関するドキュメントを参照してください。

ホットキーが原因でスロットリングの問題が発生している可能性がある

DynamoDB では、高いカーディナリティを持たないパーティションキーによって、少数のパーティションのみをターゲットとする多くのリクエストが発生する可能性があります。結果のホットパーティションが、1 秒あたり 3,000 RCU または 1,000 WCU のパーティション制限を超えると、スロットリングが発生する場合があります。CloudWatch Contributor Insights (CCI) は、各テーブル

の項目アクセスパターンの CCI グラフを提供して、この問題のデバッグを支援する診断ツールです。このツールを使用すると、DynamoDB テーブルの最も頻繁にアクセスされるキーやその他のトラフィックの傾向を継続的に監視できます。CloudWatch Contributor Insights の詳細については、「CloudWatch Contributor Insights for DynamoDB」を参照してください。詳細については、「パーティションキーを設計してワークロードを分散する」および「適切な DynamoDB パーティションキーの選択」を参照してください。

テーブルへのトラフィックがテーブルレベルのスループットクォータを超えている

テーブルレベルの読み取りスループットとテーブルレベルの書き込みスループットクォータは、テーブルレベルで適用されます。これらのクォータは、プロビジョンドキャパシティモードとオンデマンドキャパシティモードの両方のテーブルに適用されます。デフォルトでは、テーブルに設定されるスループットクォータは 40,000 読み込みリクエストユニットと 40,000 書き込みリクエストユニットです。テーブルへのトラフィックがこのクォータを超えると、テーブルがスロットリングする可能性があります。この発生を防止する方法の詳細については、「Amazon DynamoDB をモニタリングして運用状況を把握する」を参照してください。

この問題を解決するには、Service Quotas コンソールを使用して、アカウントのテーブルレベルの読み取りまたは書き込みスループットクォータを増やします。

CloudWatch メトリクスを使用したスロットリング問題の調査

以下は、スロットリングイベント中にモニタリングすべき DynamoDB メトリクスの一部です。これらを使用して、リクエストのスロットリングを起こしているオペレーションを見つけて、根本的な問題を特定します。

- `ThrottledRequests`
  - 1 つのスロットリングされているリクエストに複数のスロットリングイベントが含まれることがあるため、リクエストよりもイベントのほうが、例とより関連している場合があります。例えば、GSI を持つテーブル内の項目を更新する場合、テーブルへの書き込みオペレーションと各インデックスへの書き込みオペレーションという複数のイベントが発生します。これらのイベントの 1 つまたは複数がスロットリングされた場合でも、`ThrottledRequest` は 1 つだけです。
- `ReadThrottleEvents`
  - 1 つのテーブルまたは GSI のプロビジョニングされた RCU を超えるリクエストを監視します。
- `WriteThrottleEvents`
  - 1 つのテーブルまたは GSI のプロビジョニングされた WCU を超えるリクエストを監視します。

- `OnlineIndexConsumedWriteCapacity`
  - 新しい GSI をテーブルに追加するときに消費される WCU の数に注目します。GSI の `ConsumedWriteCapacityUnits` には、インデックスの作成中に消費される WCU は含まれません。
  - GSI に対する WCU の設定が低すぎると、バックフィルフェーズ中の受信書き込みアクティビティがスロットリングされることがあります。
- `Provisioned Read/Write`
  - テーブルまたは指定したグローバルセカンダリインデックスについて、指定した期間に消費された書き込み容量ユニットのプロビジョンド読み取りまたは書き込み容量ユニットの数。
  - デフォルトでは、`TableName` ディメンションはテーブルのみの `ProvisionedReadCapacityUnits` を返します。グローバルセカンダリインデックスの読み込みまたは書き込みのプロビジョンドキャパシティユニットの数を確認するには、`TableName` と `GlobalSecondaryIndexName` の両方を指定する必要があります。
- `Consumed Read/Write`
  - 指定された期間に消費された読み取りまたは書き込み容量ユニットの数。

DynamoDB CloudWatch のメトリクスの詳細については、「DynamoDB のメトリクスとディメンション」を参照してください。

DynamoDB 付録

トピック

SSL/TLS 接続の確立に関する問題をトラブルシューティングする

Amazon DynamoDB は、サードパーティの認証機関ではなく Amazon Trust Services (ATS) 認証機関によって署名された安全な証明書へのエンドポイントの移動を行っています。2017 年 12 月、Amazon Trust Services が発行する安全な証明書を使用して EU-WEST-3 (パリ) リージョンを立ち上げました。2017 年 12 月以降に提供が開始されたすべての新しいリージョンには、Amazon Trust Services 発行の証明書を持つエンドポイントがあります。このガイドでは、SSL/TLS 接続の問題を検証し、トラブルシューティングする方法を示します。

アプリケーションまたはサービスをテストする

ほとんどの AWS SDK とコマンドラインインターフェイス (CLI) は、Amazon Trust Services 証明書機関をサポートしています。2013 年 10 月 29 日以前にリリースされた AWS SDK for Python または CLI のバージョンを使用している場合は、アップグレードする必要があります。.NET、Java、PHP、Go、JavaScript、C ++、SDK、CLI には証明書がバンドルされていません。証明書は、基盤となるオペレーティングシステムから取得されます。2015 年 6 月 10 日以降、Ruby SDK には、少なくとも 1 つの必要な CA が含まれています。この日付以前の Ruby V2 SDK には証明書はバンドルされていませんでした。サポートされていない、カスタム、または変

更バージョンの AWS SDK を使用している場合、またはカスタム信頼ストアを使用している場合は、Amazon Trust Services 証明書機関のサポートがない可能性があります。

DynamoDB エンドポイントへのアクセスを検証するには、EU-WEST-3 リージョンの DynamoDB API または DynamoDB Streams API するアクセスするテストを開発し、TLS ハンドシェイクが成功することを検証する必要があります。このテストでアクセスする必要がある特定のエンドポイントは次のとおりです。

- DynamoDB: https://dynamodb.eu-west-3.amazonaws.com
- DynamoDB Streams: https://streams.dynamodb.eu-west-3.amazonaws.com

アプリケーションが Amazon Trust Services 認証機関をサポートしていない場合は、次のいずれかのエラーが表示されます。

- SSL/TLS ネゴシエーションエラー
- ソフトウェアが SSL/TLS ネゴシエーションの障害を示すエラーを受信するまでに長い遅延が発生します。遅延時間は、クライアントの再試行戦略とタイムアウト設定に応じて異なります。

クライアントブラウザのテスト

ブラウザが Amazon DynamoDB に接続できることを確認するには、https://dynamodb.eu-west-3.amazonaws.com を開きます。テストが成功すると、次のようなメッセージが表示されます。

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

テストが失敗した場合は、https://untrusted-root.badssl.com/ に見られるようなエラーが表示されます。

ソフトウェアアプリケーションクライアントの更新中

DynamoDB または DynamoDB Streams API エンドポイントに (ブラウザまたはプログラムで) アクセスするアプリケーションが以下のいずれかの CA をサポートしていない場合、クライアントマシン上の信頼できる CA リストを更新する必要があります。

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

クライアントが上記 3 つの CA のいずれかを既に信頼している場合、これらは DynamoDB で使用される証明書を信頼するため、アクションは必要ありません。ただし、クライアントが上記の CA を信頼していない場合、DynamoDB API または DynamoDB Streams API への HTTPS 接続が失敗します。詳細については、https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its-own-certificate-authority/ のブログ記事を参照してください。

クライアントブラウザを更新する

ブラウザの証明書バンドルは、ブラウザを更新するだけで更新できます。最も一般的なブラウザの手順は、それぞれのブラウザのウェブサイトに記載されています。

- Chrome: https://support.google.com/chrome/answer/95414?hl=en
- Firefox: https://support.mozilla.org/en-US/kb/update-firefox-latest-version
- Safari: https://support.apple.com/en-us/HT204416
- Internet Explorer: https://support.microsoft.com/en-us/help/17295/windows-internet-explorer-which-version#ie=other

手動で証明書バンドルを更新する

DynamoDB API または DynamoDB Streams API にアクセスできない場合は、証明書バンドルを更新する必要があります。そのためには、必要な CA の少なくとも 1 つをインポートする必要があります。これらは、https://www.amazontrust.com/repository/ から入手できます。

以下のオペレーティングシステムとプログラミング言語は、Amazon Trust Services 証明書をサポートしています。

- 2005 年 1 月以降の更新プログラムがインストールされている Microsoft Windows のバージョン、Windows Vista、Windows 7、Windows Server 2008、およびそれ以降のバージョン。
- MacOS X 10.4 Release 5 用 の MacOS X 10.4 with Java、MacOS X 10.5 およびそれ以降のバージョン。
- Red Hat Enterprise Linux 5 (2007 年 3 月)、Linux 6、Linux 7、CentOS 5、CentOS 6、CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (すべてのバージョン)
- Java 1.4.2_12、Java 5 update 2、およびそれ以降のバージョン (Java 6、Java 7、Java 8)

それでも接続できない場合は、該当するソフトウェアのドキュメントを参照するか、OS ベンダーまたは AWS サポート (https://aws.amazon.com/support) にお問い合わせください。

モニタリングツール

AWS では、DynamoDB のモニタリングに使用できるツールを提供しています。これらの中には、自動モニタリングを設定できるものもあれば、手動操作を必要とするものもあります。モニタリングタスクをできるだけ自動化することをお勧めします。

自動モニタリングツール

以下の自動化されたモニタリングツールを使用して、DynamoDB を監視し、問題が発生したときにレポートできます。

- Amazon CloudWatch アラーム - 指定した期間にわたって単一のメトリクスをモニタリングし、複数の期間にわたる特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すことはありません。状態が変更され、指定された期間維持されている必要があります。
- Amazon CloudWatch Logs - AWS CloudTrail またはその他の出典のログファイルのモニタリング、保存、アクセスを行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「ログファイルのモニタリング」を参照してください。
- Amazon CloudWatch Events - イベントに一致したものを 1 つ以上のターゲットの関数またはストリームに渡して、変更、状態の情報の収集、是正措置を行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Events とは」を参照してください。
- AWS CloudTrail のログのモニタリング – アカウント間でログファイルを共有し、CloudTrail のログファイルを CloudWatch Logs に送信してリアルタイムでモニタリングします。また、ログを処理するアプリケーションを Java で作成し、CloudTrail からの提供後にログファイルが変更されていないことを確認します。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail ログファイルの使用」を参照してください。

手動モニタリングツール

DynamoDB のモニタリングでもう 1 つ重要な点は、CloudWatch のアラームの対象外の項目を手動でモニタリングすることです。DynamoDB、CloudWatch、Trusted Advisor、その他の AWS コン

ソールのダッシュボードには、AWS 環境の状態が一目で分かるように表示されます。DynamoDB のログファイルも確認することもお勧めします。

- DynamoDB ダッシュボードには、次の内容が表示されます。
  - 最近のアラート
  - 合計容量
  - サービス状態
- CloudWatch ホームページには、次の内容が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して以下のことを行えます。
  - 重視するサービスをモニタリングするための[カスタマイズしたダッシュボード](#)を作成します
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
  - AWS リソースのすべてのメトリクスを検索して参照する。
  - 問題があることを通知するアラームを作成/編集する

テーブルとデータの例

Amazon DynamoDB デベロッパーガイドでは、サンプルテーブルを使用して DynamoDB のさまざまな機能について説明しています。

テーブル名	プライマリキー
ProductCatalog	シンプルなプライマリキー: • `Id` (数値)
Forum	シンプルなプライマリキー: • `Name` (文字列)
Thread	複合プライマリキー: • `ForumName` (文字列)

テーブル名	プライマリキー
	• Subject (文字列)
Reply	複合プライマリキー: • Id (文字列) • ReplyDateTime (文字列)

Reply テーブルには、PostedBy-Message-Index というグローバルセカンダリインデックスがあります。このインデックスは、Reply テーブルの 2 つの非キー属性でのクエリを容易にします。

インデックス名	プライマリキー
PostedBy-Message-Index	複合プライマリキー: • PostedBy (文字列) • Message (文字列)

これらのテーブルの詳細については、「ステップ 1: テーブルを作成します」および「ステップ 2: コンソールまたは AWS CLI を使用して、テーブルにデータを書き込みます」を参照してください。

サンプルデータファイル

トピック

- ProductCatalog サンプルデータ
- Forum サンプルデータ
- Thread サンプルデータ
- Reply サンプルデータ

このセクションでは、ProductCatalog、Forum、Thread、Reply テーブルのロードに使用されるサンプルデータファイルを示します。

各データファイルには複数の PutRequest 要素が含まれます。各要素には 1 つの項目が含まれます。これらの PutRequest 要素は、AWS Command Line Interface (AWS CLI) と共に BatchWriteItem オペレーションへの入力として使用されます。

詳細については、「ステップ 2: コンソールまたは AWS CLI を使用して、テーブルにデータを書き込みます」の「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。

ProductCatalog サンプルデータ

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "101"
                    },
                    "Title": {
                        "S": "Book 101 Title"
                    },
                    "ISBN": {
                        "S": "111-1111111111"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            }
                        ]
                    },
                    "Price": {
                        "N": "2"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 0.5"
                    },
                    "PageCount": {
                        "N": "500"
                    },
                    "InPublication": {
                        "BOOL": true
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "102"
                    },
                    "Title": {
                        "S": "Book 102 Title"
                    },
                    "ISBN": {
                        "S": "222-2222222222"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            },
                            {
                                "S": "Author2"
                            }
                        ]
                    },
                    "Price": {
                        "N": "20"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 0.8"
                    },
                    "PageCount": {
                        "N": "600"
                    },
                    "InPublication": {
                        "BOOL": true
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "103"
                    },
                    "Title": {
                        "S": "Book 103 Title"
                    },
                    "ISBN": {
                        "S": "333-3333333333"
                    },
                    "Authors": {
                        "L": [
                            {
                                "S": "Author1"
                            },
                            {
                                "S": "Author2"
                            }
                        ]
                    },
                    "Price": {
                        "N": "2000"
                    },
                    "Dimensions": {
                        "S": "8.5 x 11.0 x 1.5"
                    },
                    "PageCount": {
                        "N": "600"
                    },
                    "InPublication": {
                        "BOOL": false
                    },
                    "ProductCategory": {
                        "S": "Book"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "201"
                    },
                    "Title": {
                        "S": "18-Bike-201"
                    },
                    "Description": {
                        "S": "201 Description"
                    },
                    "BicycleType": {
                        "S": "Road"
                    },
                    "Brand": {
                        "S": "Mountain A"
                    },
                    "Price": {
                        "N": "100"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "202"
                    },
                    "Title": {
                        "S": "21-Bike-202"
                    },
                    "Description": {
                        "S": "202 Description"
                    },
                    "BicycleType": {
                        "S": "Road"
                    },
                    "Brand": {
                        "S": "Brand-Company A"
                    },
                    "Price": {
                        "N": "200"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Green"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "203"
                    },
                    "Title": {
                        "S": "19-Bike-203"
                    },
                    "Description": {
                        "S": "203 Description"
                    },
                    "BicycleType": {
                        "S": "Road"
                    },
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "300"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Green"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "204"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "204 Description"
                    },
                    "BicycleType": {
                        "S": "Mountain"
                    },
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "400"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "205"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "205 Description"
                    },
                    "BicycleType": {
                        "S": "Hybrid"
                    },
                    "Brand": {
                        "S": "Brand-Company C"
                    },
                    "Price": {
                        "N": "500"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        }
    ]
}
```

Forum サンプルデータ

```
{
    "Forum": [
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon DynamoDB"},
                    "Category": {"S":"Amazon Web Services"},
                    "Threads": {"N":"2"},
                    "Messages": {"N":"4"},
                    "Views": {"N":"1000"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon S3"},
                    "Category": {"S":"Amazon Web Services"}
                }
            }
        }
    ]
}
```

Thread サンプルデータ

```
{
    "Thread": [
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 1"
                    },
                    "Message": {
                        "S": "DynamoDB thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-22T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "index"
                            },
                            {
                                "S": "primarykey"
                            },
                            {
                                "S": "table"
                            }
                        ]
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 2"
                    },
                    "Message": {
                        "S": "DynamoDB thread 2 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-15T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "3"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "items"
                            },
                            {
                                "S": "attributes"
                            },
                            {
                                "S": "throughput"
                            }
                        ]
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon S3"
                    },
                    "Subject": {
                        "S": "S3 Thread 1"
                    },
                    "Message": {
                        "S": "S3 thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-29T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
                    },
                    "Tags": {
                        "L": [
                            {
                                "S": "largeobjects"
                            },
                            {
                                "S": "multipart upload"
                            }
                        ]
                    }
                }
            }
        }
    ]
}
```

Reply サンプルデータ

```
{
    "Reply": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-15T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-22T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 2 text"
                    },
                    "PostedBy": {
                        "S": "User B"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-29T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 2"
                    },
                    "ReplyDateTime": {
                        "S": "2015-10-05T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 2 Reply 2 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        }
    ]
}
```

サンプルテーブルを作成してデータをアップロードする

トピック

- AWS SDK for Java を使用してサンプルテーブルを作成してデータをアップロードする
- AWS SDK for .NET を使用してサンプルテーブルを作成してデータをアップロードする

DynamoDB でのコード例用のテーブルの作成とデータのロード で、DynamoDB コンソールを使用してテーブルを作成し、次に AWS CLI を使用してテーブルにデータを追加します。この付録では、テーブルの作成とプログラムによるデータの追加の両方を行うコードについて説明します。

AWS SDK for Java を使用してサンプルテーブルを作成してデータをアップロードする

次の Java コード例は、テーブルを作成し、テーブルにデータをアップロードします。結果のテーブル構造とデータについては、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。Eclipse を使用してこのコードを実行するための詳しい手順については、「Java コードの例」を参照してください。

```
package com.amazonaws.codesamples;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class CreateTablesLoadData {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    static String productCatalogTableName = "ProductCatalog";
    static String forumTableName = "Forum";
    static String threadTableName = "Thread";
    static String replyTableName = "Reply";

    public static void main(String[] args) throws Exception {

        try {

            deleteTable(productCatalogTableName);
            deleteTable(forumTableName);
            deleteTable(threadTableName);
            deleteTable(replyTableName);

            // Parameter1: table name
            // Parameter2: reads per second
            // Parameter3: writes per second
            // Parameter4/5: partition key and data type
            // Parameter6/7: sort key and data type (if applicable)

            createTable(productCatalogTableName, 10L, 5L, "Id", "N");
            createTable(forumTableName, 10L, 5L, "Name", "S");
            createTable(threadTableName, 10L, 5L, "ForumName", "S", "Subject", "S");
            createTable(replyTableName, 10L, 5L, "Id", "S", "ReplyDateTime", "S");

            loadSampleProducts(productCatalogTableName);
            loadSampleForums(forumTableName);
            loadSampleThreads(threadTableName);
            loadSampleReplies(replyTableName);

        } catch (Exception e) {
            System.err.println("Program failed:");
            System.err.println(e.getMessage());
        }
        System.out.println("Success.");
    }

    private static void deleteTable(String tableName) {
        Table table = dynamoDB.getTable(tableName);
        try {
            System.out.println("Issuing DeleteTable request for " + tableName);
            table.delete();
            System.out.println("Waiting for " + tableName + " to be deleted...this may
 take a while...");
            table.waitForDelete();
```

```
       } catch (Exception e) {
           System.err.println("DeleteTable request failed for " + tableName);
           System.err.println(e.getMessage());
       }
   }

   private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
           String partitionKeyName, String partitionKeyType) {

       createTable(tableName, readCapacityUnits, writeCapacityUnits, partitionKeyName,
partitionKeyType, null, null);
   }

   private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
           String partitionKeyName, String partitionKeyType, String sortKeyName,
String sortKeyType) {

       try {

           ArrayList<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
           keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition

                         // key

           ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
           attributeDefinitions
                   .add(new AttributeDefinition().withAttributeName(partitionKeyName)
                           .withAttributeType(partitionKeyType));

           if (sortKeyName != null) {
               keySchema.add(new
KeySchemaElement().withAttributeName(sortKeyName).withKeyType(KeyType.RANGE)); // Sort

                         // key
               attributeDefinitions
                       .add(new
AttributeDefinition().withAttributeName(sortKeyName).withAttributeType(sortKeyType));
           }
```

```
            CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)
                    .withProvisionedThroughput(new
ProvisionedThroughput().withReadCapacityUnits(readCapacityUnits)
                            .withWriteCapacityUnits(writeCapacityUnits));

            // If this is the Reply table, define a local secondary index
            if (replyTableName.equals(tableName)) {

                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName("PostedBy").withAttributeType("S"));

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();
                localSecondaryIndexes.add(new
LocalSecondaryIndex().withIndexName("PostedBy-Index")
                        .withKeySchema(
                                new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH), //
Partition

                               // key
                                new
KeySchemaElement().withAttributeName("PostedBy").withKeyType(KeyType.RANGE)) // Sort

                          // key
                        .withProjection(new
Projection().withProjectionType(ProjectionType.KEYS_ONLY)));

                request.setLocalSecondaryIndexes(localSecondaryIndexes);
            }

            request.setAttributeDefinitions(attributeDefinitions);

            System.out.println("Issuing CreateTable request for " + tableName);
            Table table = dynamoDB.createTable(request);
            System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
            table.waitForActive();

        } catch (Exception e) {
            System.err.println("CreateTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleProducts(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Id", 101).withString("Title", "Book 
101 Title")
                    .withString("ISBN", "111-1111111111")
                    .withStringSet("Authors", new 
HashSet<String>(Arrays.asList("Author1"))).withNumber("Price", 2)
                    .withString("Dimensions", "8.5 x 11.0 x 
0.5").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory", 
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 102).withString("Title", "Book 102 
Title")
                    .withString("ISBN", "222-2222222222")
                    .withStringSet("Authors", new 
HashSet<String>(Arrays.asList("Author1", "Author2")))
                    .withNumber("Price", 20).withString("Dimensions", "8.5 x 11.0 x 
0.8").withNumber("PageCount", 600)
                    .withBoolean("InPublication", true).withString("ProductCategory", 
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 103).withString("Title", "Book 103 
Title")
                    .withString("ISBN", "333-3333333333")
                    .withStringSet("Authors", new 
HashSet<String>(Arrays.asList("Author1", "Author2")))
                    // Intentional. Later we'll run Scan to find price error. Find
                    // items > 1000 in price.
                    .withNumber("Price", 2000).withString("Dimensions", "8.5 x 11.0 x 
1.5").withNumber("PageCount", 600)
                    .withBoolean("InPublication", false).withString("ProductCategory", 
"Book");
```

```
            table.putItem(item);

            // Add bikes.

            item = new Item().withPrimaryKey("Id", 201).withString("Title", "18-
Bike-201")
                    // Size, followed by some title.
                    .withString("Description", "201
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Mountain A")
                    // Trek, Specialized.
                    .withNumber("Price", 100).withStringSet("Color", new
 HashSet<String>(Arrays.asList("Red", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 202).withString("Title", "21-
Bike-202")
                    .withString("Description", "202
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company A").withNumber("Price", 200)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Green",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 203).withString("Title", "19-
Bike-203")
                    .withString("Description", "203
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 300)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Green", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 204).withString("Title", "18-
Bike-204")
                    .withString("Description", "204
 Description").withString("BicycleType", "Mountain")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 400)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);
            item = new Item().withPrimaryKey("Id", 205).withString("Title", "20-
Bike-205")
                    .withString("Description", "205
 Description").withString("BicycleType", "Hybrid")
                    .withString("Brand", "Brand-Company C").withNumber("Price", 500)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleForums(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Name", "Amazon DynamoDB")
                    .withString("Category", "Amazon Web
 Services").withNumber("Threads", 2).withNumber("Messages", 4)
                    .withNumber("Views", 1000);
            table.putItem(item);

            item = new Item().withPrimaryKey("Name", "Amazon
 S3").withString("Category", "Amazon Web Services")
                    .withNumber("Threads", 0);
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleThreads(String tableName) {
        try {
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000); // 7
            // days
            // ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000); // 14
            // days
            // ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000); // 21
            // days
            // ago

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("ForumName", "Amazon DynamoDB")
                    .withString("Subject", "DynamoDB Thread 1").withString("Message",
 "DynamoDB thread 1 message")
                    .withString("LastPostedBy", "User
 A").withString("LastPostedDateTime", dateFormatter.format(date2))
                    .withNumber("Views", 0).withNumber("Replies",
 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
 "primarykey", "table")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
 DynamoDB").withString("Subject", "DynamoDB Thread 2")
                    .withString("Message", "DynamoDB thread 2
 message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
 dateFormatter.format(date3)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
 "partitionkey", "sortkey")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
 S3").withString("Subject", "S3 Thread 1")
                    .withString("Message", "S3 Thread 3
 message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
 dateFormatter.format(date1)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new
 HashSet<String>(Arrays.asList("largeobjects", "multipart upload")));
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleReplies(String tableName) {
        try {
            // 1 day ago
            long time0 = (new Date()).getTime() - (1 * 24 * 60 * 60 * 1000);
            // 7 days ago
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000);
            // 14 days ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000);
            // 21 days ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000);

            Date date0 = new Date();
            date0.setTime(time0);

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);
            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            // Add threads.

            Item item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB
 Thread 1")
                    .withString("ReplyDateTime", (dateFormatter.format(date3)))
                    .withString("Message", "DynamoDB Thread 1 Reply 1
 text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 1")
                    .withString("ReplyDateTime", dateFormatter.format(date2))
                    .withString("Message", "DynamoDB Thread 1 Reply 2
 text").withString("PostedBy", "User B");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date1))
                    .withString("Message", "DynamoDB Thread 2 Reply 1
 text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date0))
                    .withString("Message", "DynamoDB Thread 2 Reply 2
 text").withString("PostedBy", "User A");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());

        }
    }

}
```

AWS SDK for .NET を使用してサンプルテーブルを作成してデータをアップロードする

次の C# コード例は、テーブルを作成し、テーブルにデータをアップロードします。結果のテーブル構造とデータについては、「DynamoDB でのコード例用のテーブルの作成とデータのロード」を参照してください。Visual Studio でこのコードを実行するための詳しい手順については、「.NET コード例」を参照してください。

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class CreateTablesLoadData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                //DeleteAllTables(client);
                DeleteTable("ProductCatalog");
                DeleteTable("Forum");
                DeleteTable("Thread");
                DeleteTable("Reply");

                // Create tables (using the AWS SDK for .NET low-level API).
                CreateTableProductCatalog();
                CreateTableForum();
                CreateTableThread(); // ForumTitle, Subject */
                CreateTableReply();

                // Load data (using the .NET SDK document API)
                LoadSampleProducts();
                LoadSampleForums();
                LoadSampleThreads();
                LoadSampleReplies();
                Console.WriteLine("Sample complete!");
                Console.WriteLine("Press ENTER to continue");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void DeleteTable(string tableName)
        {
            try
            {
                var deleteTableResponse = client.DeleteTable(new DeleteTableRequest()
                {
                    TableName = tableName
                });
                WaitTillTableDeleted(client, tableName, deleteTableResponse);
            }
            catch (ResourceNotFoundException)
            {
                // There is no such table.
            }
        }

        private static void CreateTableProductCatalog()
        {
            string tableName = "ProductCatalog";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                              {
                                  new AttributeDefinition
                                  {
                                      AttributeName = "Id",
                                      AttributeType = "N"
                                  }
                              },
                KeySchema = new List<KeySchemaElement>()
                              {
                                  new KeySchemaElement
                                  {
                                      AttributeName = "Id",
                                      KeyType = "HASH"
                                  }
                              },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void CreateTableForum()
        {
            string tableName = "Forum";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                              {
                                  new AttributeDefinition
                                  {
                                      AttributeName = "Name",
                                      AttributeType = "S"
                                  }
                              },
                KeySchema = new List<KeySchemaElement>()
                              {
                                  new KeySchemaElement
                                  {
                                      AttributeName = "Name", // forum Title
                                      KeyType = "HASH"
                                  }
                              },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });
            WaitTillTableCreated(client, tableName, response);
        }

        private static void CreateTableThread()
        {
            string tableName = "Thread";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                              {
                                  new AttributeDefinition
                                  {
                                      AttributeName = "ForumName", // Hash attribute
                                      AttributeType = "S"
                                  },
                                  new AttributeDefinition
                                  {
                                      AttributeName = "Subject",
                                      AttributeType = "S"
                                  }
                              },
                KeySchema = new List<KeySchemaElement>()
                              {
                                  new KeySchemaElement
                                  {
                                      AttributeName = "ForumName", // Hash attribute
                                      KeyType = "HASH"
                                  },
                                  new KeySchemaElement
                                  {
                                      AttributeName = "Subject", // Range attribute
                                      KeyType = "RANGE"
                                  }
                              },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void CreateTableReply()
        {
            string tableName = "Reply";
            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                              {
                                  new AttributeDefinition
                                  {
                                      AttributeName = "Id",
                                      AttributeType = "S"
                                  },
                                  new AttributeDefinition
                                  {
                                      AttributeName = "ReplyDateTime",
                                      AttributeType = "S"
                                  },
                                  new AttributeDefinition
                                  {
                                      AttributeName = "PostedBy",
                                      AttributeType = "S"
                                  }
                              },
                KeySchema = new List<KeySchemaElement>()
                              {
                                  new KeySchemaElement()
                                  {
                                      AttributeName = "Id",
                                      KeyType = "HASH"
                                  },
                                  new KeySchemaElement()
                                  {
                                      AttributeName = "ReplyDateTime",
                                      KeyType = "RANGE"
                                  }
                              },
                LocalSecondaryIndexes = new List<LocalSecondaryIndex>()
                              {
                                  new LocalSecondaryIndex()
                                  {
                                      IndexName = "PostedBy_index",
                                      KeySchema = new List<KeySchemaElement>() {
                                          new KeySchemaElement() {
                                              AttributeName = "Id", KeyType = "HASH"
                                          },
                                          new KeySchemaElement() {
                                              AttributeName = "PostedBy", KeyType =
 "RANGE"
                                          }
                                      },
                                      Projection = new Projection() {
                                          ProjectionType = ProjectionType.KEYS_ONLY
                                      }
                                  }
                              },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void WaitTillTableCreated(AmazonDynamoDBClient client, string
 tableName,
                             CreateTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable.
            while (status != "ACTIVE")
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
 res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                // Try-catch to handle potential eventual-consistency issue.
                catch (ResourceNotFoundException)
                { }
            }
        }

        private static void WaitTillTableDeleted(AmazonDynamoDBClient client, string
 tableName,
                             DeleteTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable
            try
            {
                while (status == "DELETING")
                {
                    System.Threading.Thread.Sleep(5000); // wait 5 seconds

                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
 res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
            }
            catch (ResourceNotFoundException)
            {
                // Table deleted.
            }
        }

        private static void LoadSampleProducts()
        {
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // ********** Add Books *********************
            var book1 = new Document();
            book1["Id"] = 101;
            book1["Title"] = "Book 101 Title";
            book1["ISBN"] = "111-1111111111";
            book1["Authors"] = new List<string> { "Author 1" };
            book1["Price"] = -2; // *** Intentional value. Later used to illustrate
 scan.
            book1["Dimensions"] = "8.5 x 11.0 x 0.5";
            book1["PageCount"] = 500;
            book1["InPublication"] = true;
            book1["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book1);

            var book2 = new Document();

            book2["Id"] = 102;
            book2["Title"] = "Book 102 Title";
            book2["ISBN"] = "222-2222222222";
            book2["Authors"] = new List<string> { "Author 1", "Author 2" }; ;
            book2["Price"] = 20;
            book2["Dimensions"] = "8.5 x 11.0 x 0.8";
            book2["PageCount"] = 600;
            book2["InPublication"] = true;
            book2["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book2);

            var book3 = new Document();
            book3["Id"] = 103;
            book3["Title"] = "Book 103 Title";
            book3["ISBN"] = "333-3333333333";
            book3["Authors"] = new List<string> { "Author 1", "Author2", "Author
 3" }; ;
            book3["Price"] = 2000;
            book3["Dimensions"] = "8.5 x 11.0 x 1.5";
            book3["PageCount"] = 700;
            book3["InPublication"] = false;
            book3["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book3);
            // ************ Add bikes. *******************
            var bicycle1 = new Document();
            bicycle1["Id"] = 201;
            bicycle1["Title"] = "18-Bike 201"; // size, followed by some title.
            bicycle1["Description"] = "201 description";
            bicycle1["BicycleType"] = "Road";
            bicycle1["Brand"] = "Brand-Company A"; // Trek, Specialized.
            bicycle1["Price"] = 100;
            bicycle1["Color"] = new List<string> { "Red", "Black" };
            bicycle1["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle1);

            var bicycle2 = new Document();
            bicycle2["Id"] = 202;
            bicycle2["Title"] = "21-Bike 202Brand-Company A";
            bicycle2["Description"] = "202 description";
            bicycle2["BicycleType"] = "Road";
            bicycle2["Brand"] = "";
            bicycle2["Price"] = 200;
            bicycle2["Color"] = new List<string> { "Green", "Black" };
            bicycle2["ProductCategory"] = "Bicycle";
            productCatalogTable.PutItem(bicycle2);

            var bicycle3 = new Document();
            bicycle3["Id"] = 203;
            bicycle3["Title"] = "19-Bike 203";
            bicycle3["Description"] = "203 description";
            bicycle3["BicycleType"] = "Road";
            bicycle3["Brand"] = "Brand-Company B";
            bicycle3["Price"] = 300;
            bicycle3["Color"] = new List<string> { "Red", "Green", "Black" };
            bicycle3["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle3);

            var bicycle4 = new Document();
            bicycle4["Id"] = 204;
            bicycle4["Title"] = "18-Bike 204";
            bicycle4["Description"] = "204 description";
            bicycle4["BicycleType"] = "Mountain";
            bicycle4["Brand"] = "Brand-Company B";
            bicycle4["Price"] = 400;
            bicycle4["Color"] = new List<string> { "Red" };
            bicycle4["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle4);

            var bicycle5 = new Document();
            bicycle5["Id"] = 205;
            bicycle5["Title"] = "20-Title 205";
            bicycle4["Description"] = "205 description";
            bicycle5["BicycleType"] = "Hybrid";
            bicycle5["Brand"] = "Brand-Company C";
            bicycle5["Price"] = 500;
            bicycle5["Color"] = new List<string> { "Red", "Black" };
            bicycle5["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle5);
        }

        private static void LoadSampleForums()
        {
            Table forumTable = Table.LoadTable(client, "Forum");

            var forum1 = new Document();
            forum1["Name"] = "Amazon DynamoDB"; // PK
            forum1["Category"] = "Amazon Web Services";
            forum1["Threads"] = 2;
            forum1["Messages"] = 4;
            forum1["Views"] = 1000;

            forumTable.PutItem(forum1);

            var forum2 = new Document();
            forum2["Name"] = "Amazon S3"; // PK
            forum2["Category"] = "Amazon Web Services";
            forum2["Threads"] = 1;

            forumTable.PutItem(forum2);
        }

        private static void LoadSampleThreads()
        {
            Table threadTable = Table.LoadTable(client, "Thread");

            // Thread 1.
            var thread1 = new Document();
            thread1["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread1["Subject"] = "DynamoDB Thread 1"; // Range attribute.
            thread1["Message"] = "DynamoDB thread 1 message text";
            thread1["LastPostedBy"] = "User A";
            thread1["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
 0, 0, 0));
            thread1["Views"] = 0;
            thread1["Replies"] = 0;
            thread1["Answered"] = false;
            thread1["Tags"] = new List<string> { "index", "primarykey", "table" };

            threadTable.PutItem(thread1);

            // Thread 2.
            var thread2 = new Document();
            thread2["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread2["Subject"] = "DynamoDB Thread 2"; // Range attribute.
            thread2["Message"] = "DynamoDB thread 2 message text";
            thread2["LastPostedBy"] = "User A";
            thread2["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
 0, 0, 0));
            thread2["Views"] = 0;
            thread2["Replies"] = 0;
            thread2["Answered"] = false;
            thread2["Tags"] = new List<string> { "index", "primarykey", "rangekey" };

            threadTable.PutItem(thread2);

            // Thread 3.
            var thread3 = new Document();
            thread3["ForumName"] = "Amazon S3"; // Hash attribute.
            thread3["Subject"] = "S3 Thread 1"; // Range attribute.
            thread3["Message"] = "S3 thread 3 message text";
            thread3["LastPostedBy"] = "User A";
            thread3["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7, 0,
 0, 0));
            thread3["Views"] = 0;
            thread3["Replies"] = 0;
            thread3["Answered"] = false;
            thread3["Tags"] = new List<string> { "largeobjects", "multipart upload" };
            threadTable.PutItem(thread3);
        }

        private static void LoadSampleReplies()
        {
            Table replyTable = Table.LoadTable(client, "Reply");
```

```
           // Reply 1 - thread 1.
           var thread1Reply1 = new Document();
           thread1Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
           thread1Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0)); // Range attribute.
           thread1Reply1["Message"] = "DynamoDB Thread 1 Reply 1 text";
           thread1Reply1["PostedBy"] = "User A";

           replyTable.PutItem(thread1Reply1);

           // Reply 2 - thread 1.
           var thread1reply2 = new Document();
           thread1reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
           thread1reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0)); // Range attribute.
           thread1reply2["Message"] = "DynamoDB Thread 1 Reply 2 text";
           thread1reply2["PostedBy"] = "User B";

           replyTable.PutItem(thread1reply2);

           // Reply 3 - thread 1.
           var thread1Reply3 = new Document();
           thread1Reply3["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
           thread1Reply3["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
0, 0, 0)); // Range attribute.
           thread1Reply3["Message"] = "DynamoDB Thread 1 Reply 3 text";
           thread1Reply3["PostedBy"] = "User B";

           replyTable.PutItem(thread1Reply3);

           // Reply 1 - thread 2.
           var thread2Reply1 = new Document();
           thread2Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
attribute.
           thread2Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
0, 0, 0)); // Range attribute.
           thread2Reply1["Message"] = "DynamoDB Thread 2 Reply 1 text";
           thread2Reply1["PostedBy"] = "User A";


           replyTable.PutItem(thread2Reply1);
```

```
            // Reply 2 - thread 2.
            var thread2Reply2 = new Document();
            thread2Reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(1,
 0, 0, 0)); // Range attribute.
            thread2Reply2["Message"] = "DynamoDB Thread 2 Reply 2 text";
            thread2Reply2["PostedBy"] = "User A";

            replyTable.PutItem(thread2Reply2);
        }
    }
}
```

AWS SDK for Python (Boto) を使用した DynamoDB のサンプルアプリケーション: Tic-tac-toe

トピック

この Tic-Tac-Toe ゲームのサンプルウェブアプリケーションは、Amazon DynamoDB 上に構築されます。このアプリケーションでは AWS SDK for Python (Boto) を使用して、DynamoDB テーブルにゲームデータを保存するのに必要な DynamoDB 呼び出しを行います。また、Python ウェブフレームワーク Flask を使用して、データのモデル化の方法を含む DynamoDB でのアプリケーション開発について、包括的に見ることができます。また、DynamoDB でのデータのモデル化に関するベストプラクティスを示します。これには、ゲームアプリケーション用に作成するテーブル、ユーザーが定義するプライマリキー、クエリ要件に基づいて必要になる追加のインデックス、および連結された属性値の使用などが含まれます。

Tic-Tac-Toe アプリケーションは、次のようにしてウェブ上でプレイします。

1. アプリケーションのホームページにログインします。
2. 次に、他のユーザーを対戦相手としてゲームに招待します。

他のユーザーが招待を受け入れるまで、ゲームのステータスは PENDING となります。対戦相手が招待を受け入れると、ステータスは IN_PROGRESS に変わります。

3. ゲームは、対戦相手がログインして招待を受け入れてから開始されます。
4. このアプリケーションでは、ゲームのすべての動きやステータス情報は、DynamoDB テーブルに格納されます。
5. ゲームは優勝または引き分けで終わり、これによりゲームのステータスは FINISHED に設定されます。

アプリケーション構築についてすべて網羅した演習を各ステップで示します。

- ステップ 1: ローカルにデプロイおよびテストを実行します – このセクションでは、アプリケーションをローカルコンピュータにダウンロード、デプロイし、そのテストを行います。DynamoDB のダウンロード可能バージョンで、必要なテーブルを作成していきます。
- ステップ 2: データモデルと実装の詳細を調べます – このセクションでは、最初にインデックスや連結された属性値の使用を含め、データモデルの詳細について説明します。次に、アプリケーションの動作について説明します。
- ステップ 3: DynamoDB サービスを使用して実稼働環境のデプロイを行います – このセクションでは、実稼働環境でのデプロイの考慮事項について説明します。このステップでは、Amazon DynamoDB サービスを使用してテーブルを作成し、AWS Elastic Beanstalk を使用してアプリケーションをデプロイします。実稼働状態のアプリケーションがある場合は、アプリケーションが DynamoDB テーブルにアクセスできるようにするため、適切なアクセス許可を付与します。このセクションの手順では、本稼働デプロイについて詳細に説明します。
- ステップ 4: リソースをクリーンアップする – このセクションでは、このサンプルアプリケーションでカバーしていない領域に目を向けます。また、料金が発生しないようにするため、前のステップで作成した AWS リソースを削除するためのステップについても説明します。

ステップ 1: ローカルにデプロイおよびテストを実行します

トピック

このステップでは、Tic-Tac-Toe ゲームアプリケーションをローカルコンピュータにダウンロード、デプロイ、およびテストします。Amazon DynamoDB ウェブサービスを使用する代わりに、DynamoDB をコンピュータにダウンロードして必要なテーブルを作成します。

1.1: 必要なパッケージのダウンロードとインストール

このアプリケーションをローカルでテストするには、次のものが必要です。

- Python
- Flask（Python 用のマイクロフレームワーク）
- AWS SDK for Python (Boto)
- ローカルコンピュータ上で実行される DynamoDB
- Git

これらのツールを入手するには、以下の作業を実行します。

1. Python をインストールします。詳しい手順については、「Python のダウンロード」を参照してください。

   Tic-Tac-Toe アプリケーションは、Python バージョン 2.7 を使用してテスト済みです。
2. Python パッケージインストーラ（PIP）を使用して Flask および AWS SDK for Python (Boto) をインストールします。
   - PIP をインストールします。

     手順については、「PIP のインストール」を参照してください。インストールページで、get-pip.py リンクを選択して、ファイルを保存します。次に、管理者としてコマンドターミナルを開き、コマンドプロンプトで以下のように入力します。

     ```
     python.exe get-pip.py
     ```

     Linux では、`.exe` 拡張子を指定しません。`python get-pip.py` のみを指定します。
   - PIP で、次に示すコードを使用して Flask と Boto パッケージをインストールします。

     ```
     pip install Flask
     pip install boto
     pip install configparser
     ```

3. DynamoDB をローカルのコンピュータにダウンロードします。その実行方法については、「DynamoDB local (ダウンロード可能バージョン) のセットアップ」を参照してください。
4. Tic-Tac-Toe アプリケーションをダウンロードします。
   a. Git をインストールします。手順については、「Git のダウンロード」を参照してください。
   b. アプリケーションをダウンロードするには、次のコードを実行します。

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2: ゲームアプリケーションをテストします

Tic-Tac-Toe アプリケーションをテストするには、ローカルコンピュータで DynamoDB を実行する必要があります。

tic-tac-toe アプリケーションを実行するには

1. DynamoDB を起動します。
2. Tic-Tac-Toe アプリケーション用のウェブサーバーを起動します。

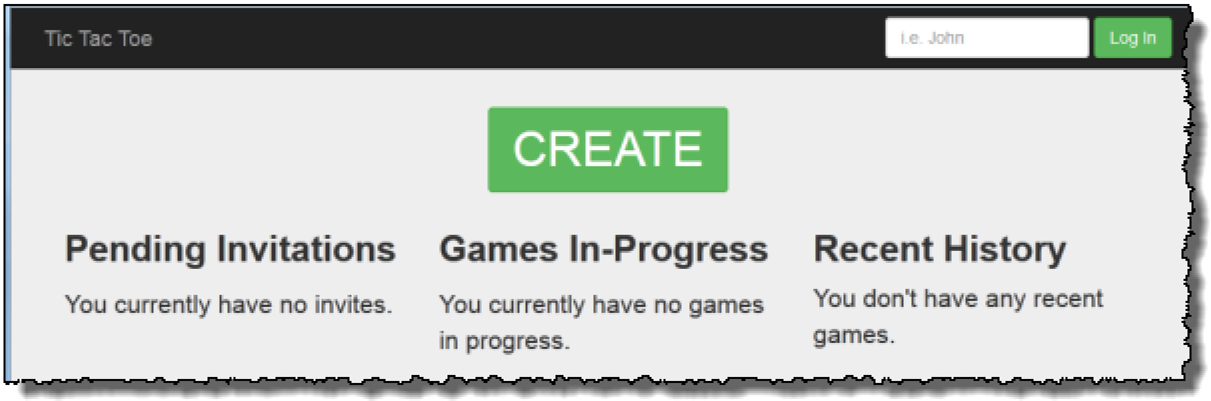
   そのためには、コマンドターミナルを開き、Tic-Tac-Toe アプリケーションをダウンロードしたフォルダに移動し、次に示すコードを使用してアプリケーションをローカルに実行します。

   ```
   python.exe application.py --mode local --serverPort 5000 --port 8000
   ```

   Linux では、`.exe` 拡張子を指定しません。
3. ウェブブラウザを開き、次のように入力します。

   ```
   http://localhost:5000/
   ```

   ブラウザにホームページが表示されます。



Tic Tac Toe

i.e. John

Log In

CREATE

Pending Invitations

You currently have no invites.

Games In-Progress

You currently have no games in progress.

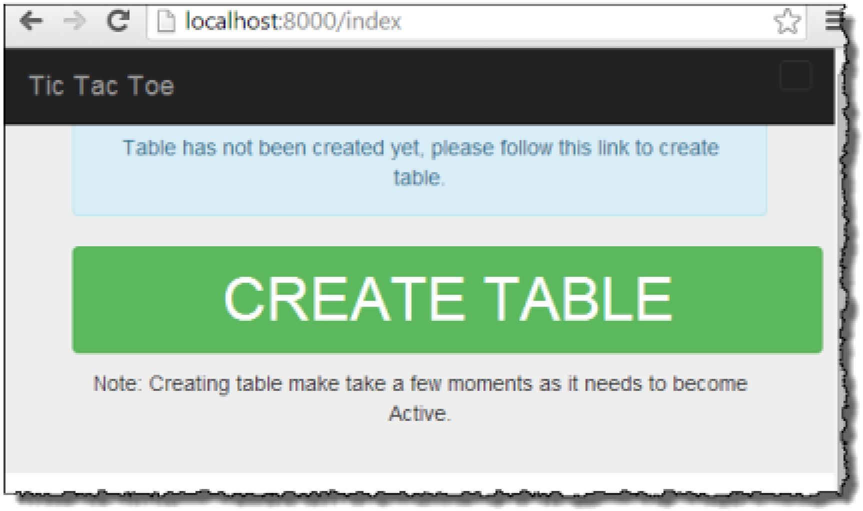
Recent History

You don't have any recent games.

4. [Log in] (ログイン) ボックスに **user1** と入力し、user1 としてログインします。

> Note
>
> このサンプルアプリケーションは、ユーザー認証を実行しません。ユーザー ID のみがプレーヤーを識別するために使用されます。2 人のプレーヤーが同じエイリアスでログインすると、アプリケーションは 2 つの別のブラウザでプレイしているかのように動作します。

5. 初めてゲームを実行する場合は、DynamoDB で必要なテーブル (`Games`) を作成するように促すページが表示されます。[テーブルの作成] を選択します。



localhost:8000/index

Tic Tac Toe

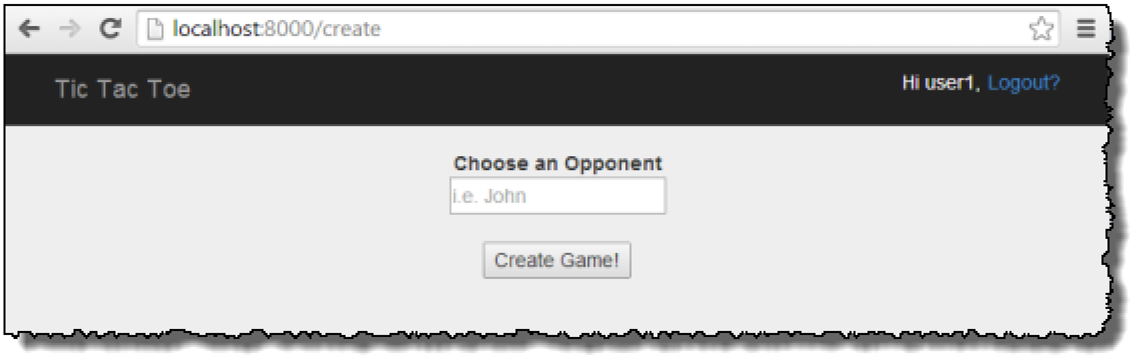
Table has not been created yet, please follow this link to create table.

CREATE TABLE

Note: Creating table make take a few moments as it needs to become Active.

6. [作成] を選択して、最初の Tic-Tac-Toe ゲームを作成します。

7. [Choose an Opponent] (対戦相手の選択) ボックスに **user2** と入力し、[Create Game!] (ゲームを作成する!) を選択します。



この操作を行うと、Games テーブルに項目を追加してゲームが作成されます。これにより、ゲームのステータスが PENDING に設定されます。

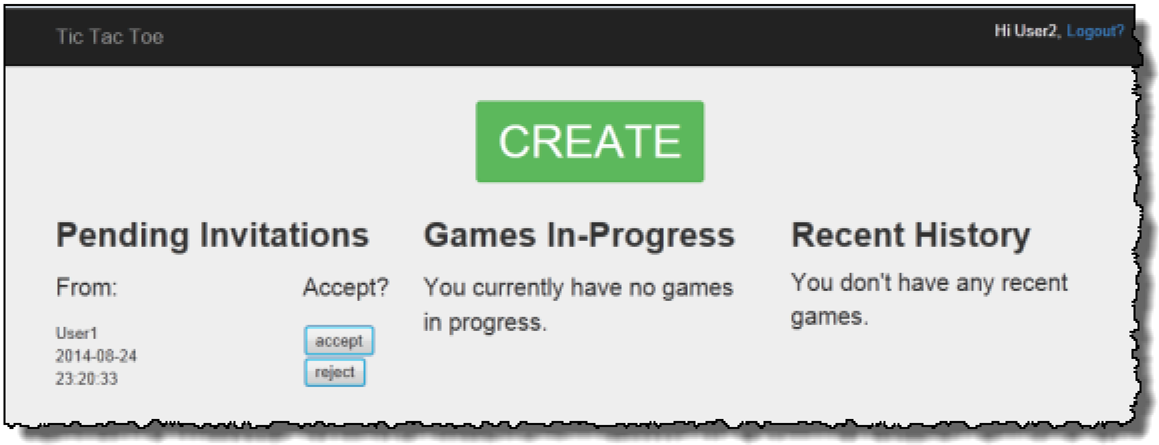
8. 別のブラウザウィンドウを開き、次のように入力します。

```
http://localhost:5000/
```

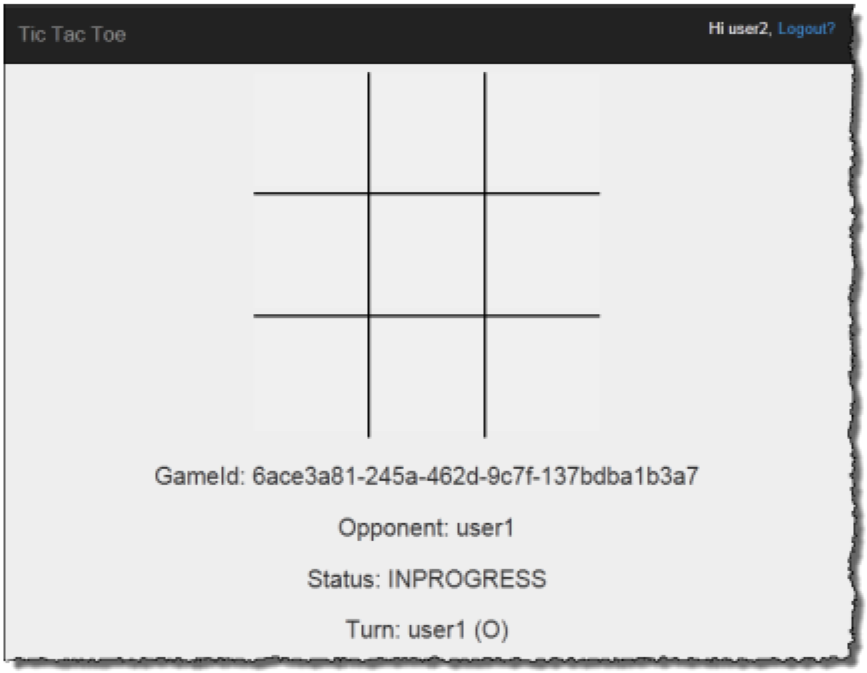
ブラウザは Cookie を通じて情報を渡すので、Cookie を引き継がないように、匿名モードまたはプライベートブラウジングを使用します。

9. user2 としてログインします。

user1 からの保留中の招待を示すページが表示されます。



10. [accept (受け入れる)] を選択して招待を受け入れます。



GameId: 6ace3a81-245a-462d-9c7f-137bdba1b3a7

Opponent: user1

Status: INPROGRESS

Turn: user1 (O)

ゲームページが、空の Tic-Tac-Toe グリッドとともに表示されます。このページには、ゲーム ID、誰の番か、ゲームのステータスなど、関連するゲーム情報も表示されます。

11. ゲームをプレイします。

ユーザーが動くたびに、ウェブサービスは、`Games` テーブルのゲーム項目に条件付き更新を実行するためのリクエストを DynamoDB に送信します。たとえば、条件により、動きが有効で、ユーザーが選択した四角形が利用可能で、動いたユーザーの番であったことを確認できます。動きが有効な場合、更新操作により、ボードでの選択に対応する新しい属性が追加されます。また、既存の属性値が、次に動くことができるユーザーに設定されます。

アプリケーションは、ゲームページから非同期の JavaScript 呼び出しを毎秒、最大 5 分にわたって実行し、DynamoDB 内のゲームの状態に変化があったかどうかを確認します。変わった場合、アプリケーションは新しい情報でページを更新します。5 分後に、アプリケーションはリクエストを中止します。ユーザーはページを更新して、更新された情報を取得する必要があります。

ステップ 2: データモデルと実装の詳細を調べます

トピック

2.1: 基本的なデータモデル

このサンプルアプリケーションでは、次の DynamoDB データモデルの概念を説明します。

- テーブル – DynamoDB では、テーブルは項目 (つまりレコード) の集合であり、各項目には、属性と呼ばれる名前と値のペアが集約されています。

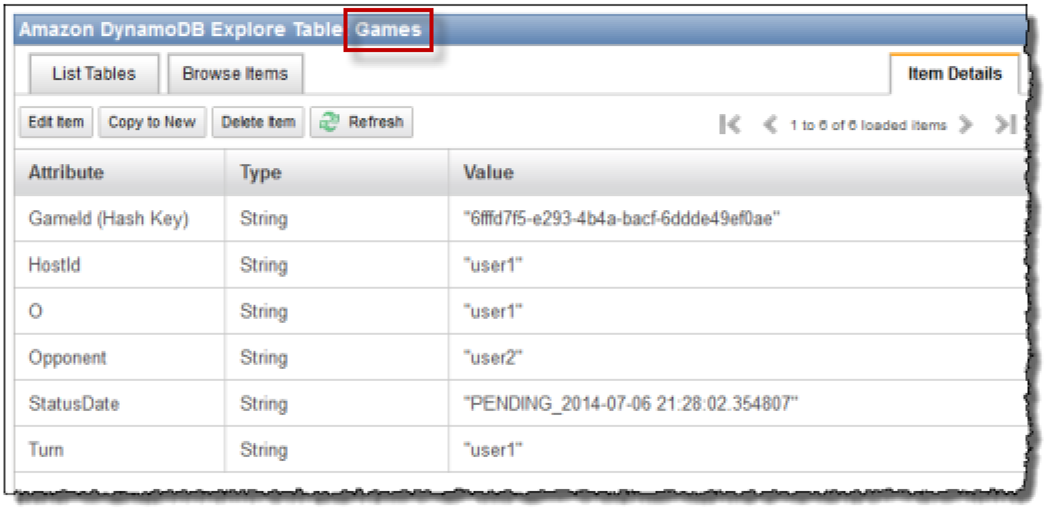
  この Tic-Tac-Toe の例では、アプリケーションは、`Games` テーブル内のすべてのゲームデータを保存します。アプリケーションはゲームごとにテーブルで 1 つの項目を作成し、すべてのゲームデータを属性として格納します。Tic-Tac-Toe ゲームでは最大 9 回の移動が可能です。DynamoDB テーブルは、プライマリキーのみが必須の属性である場合はスキーマを持たないので、アプリケーションは、ゲーム項目 ごとに異なる数の属性を格納できます。

  `Games` テーブルには、文字列型の 1 つの属性 `GameId`で構成されるシンプルなプライマリキーがあります。アプリケーションは各ゲームに一意の ID を割り当てます。DynamoDB でのプライマリキーの詳細については、「プライマリキー」を参照してください。

  ユーザーが他のユーザーを招待して Tic-Tac-Toe ゲームを開始すると、アプリケーションは、次のようなゲームメタデータを保存する属性で、`Games` テーブルに新しい項目を作成します。

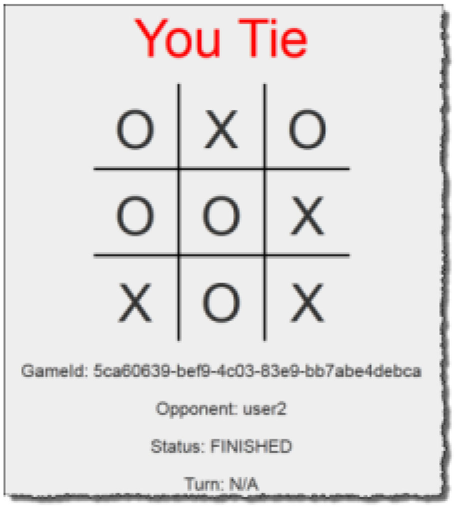
  - `HostId`ゲームを開始したユーザーである 。
  - `Opponent`ゲームに招待されたユーザーである 。
  - プレイする番のユーザー。最初にゲームを開始したユーザー。
  - ボードで O 記号を使用するユーザー。ゲームを開始するユーザーは O 記号を使用します。

  さらに、アプリケーションは `StatusDate` 連結属性を作成し、ゲームの初期状態を `PENDING` としてマークします。次のスクリーンショットに、DynamoDB コンソールに表示される項目の例を示します。



Attribute	Type	Value
GameId (Hash Key)	String	"6fffd7f5-e293-4b4a-bacf-6ddde49ef0ae"
HostId	String	"user1"
O	String	"user1"
Opponent	String	"user2"
StatusDate	String	"PENDING_2014-07-06 21:28:02.354807"
Turn	String	"user1"

アプリケーションは、ゲームの進行に合わせて、ゲームで動きがあるたびに 1 つの属性をテーブルに追加します。属性名はボードの位置 (TopLeft、BottomRight など) です。たとえば、動きには値 TopLeftの O 属性、値 TopRight の O 属性、および値 BottomRightの X 属性があるなどです。属性値は、動いたユーザーに応じて、O または X になります。たとえば、次のボードを考えてみます。


You Tie

O	X	O
O	O	X
X	O	X

GameId: 5ca60639-bef9-4c03-83e9-bb7abe4debca

Opponent: user2

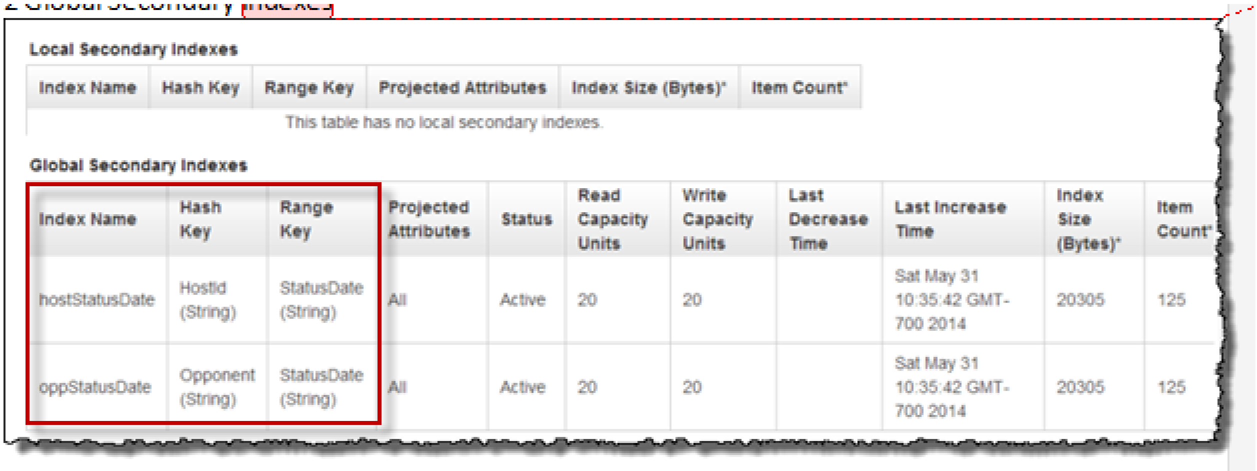
Status: FINISHED

Turn: N/A

- 連結属性値 – StatusDate 属性が、連結値属性を示します。この手法では、ゲームのステータス (PENDING、IN_PROGRESS、FINISHED) および日付 (最後の動き) を格納する個別の属性を作成する代わりに、IN_PROGRESS_2014-04-30 10:20:32 などの単一の属性としてそれらを組み合わせます。

  次に、アプリケーションは、インデックスのソートキーとして StatusDate を指定して、セカンダリインデックスの作成で StatusDate 属性を使用します。StatusDate 連結値属性を使用する利点について、さらに次のインデックスの説明で示します。

- グローバルセカンダリインデックス – テーブルのプライマリキー、GameId を使用すると、テーブルを効率的に照会してゲーム項目を検索できます。プライマリキー属性以外の属性でテーブルをクエリするため、DynamoDB ではセカンダリインデックスの作成をサポートしています。このサンプルアプリケーションでは、次の 2 つのセカンダリインデックスを構築します。



2 Global Secondary Indexes

Local Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Index Size (Bytes)*	Item Count*
This table has no local secondary indexes.					

Global Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Status	Read Capacity Units	Write Capacity Units	Last Decrease Time	Last Increase Time	Index Size (Bytes)*	Item Count*
hostStatusDate	HostId (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125
oppStatusDate	Opponent (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125

  - HostId-StatusDate-index。このインデックスはパーティションキーとして HostId を、ソートキーとして StatusDate を持ちます。このインデックスを使用して、たとえば特定のユーザーによってホストされたゲームを検索するなど、HostId でクエリを実行できます。
  - OpponentId-StatusDate-index。このインデックスはパーティションキーとして OpponentId を、ソートキーとして StatusDate を持ちます。たとえば、特定のユーザーが対戦相手であるゲームを検索するなど、このインデックスを使用して Opponent でクエリを実行できます。

これらのインデックスは、グローバルセカンダリインデックスと呼ばれます。これは、インデックスのパーティションキーが、テーブルのプライマリキーで使用されるパーティションキー (GameId) とは同じでないためです。

両方のインデックスがソートキーとして StatusDate を指定することに注意してください。これを行うと、次のことが可能になります。

- BEGINS_WITH 比較演算子を使用してクエリを実行できます。たとえば、特定のユーザーによってホストされた IN_PROGRESS 属性を持つすべてのゲームを検索できます。この場合、BEGINS_WITH 演算子は、StatusDate で始まる IN_PROGRESS 値を確認します。
- DynamoDB は、項目をソートキー値によって整列させ、インデックスに保存します。したがって、すべてのステータスのプレフィックスは同じ（たとえば、IN_PROGRESS）になり、日付部分に使用される ISO 形式には、最も古いものから最も新しいものの順にソートされた項目が含まれます。この手法により、たとえば次のように特定のクエリを効率的に実行できるようになります。

- ログインしているユーザーによってホストされている最新の IN_PROGRESS のゲームを最大 10 個取得します。このクエリでは、HostId-StatusDate-index インデックスを指定します。
- ログインしているユーザーが対戦相手である最新の IN_PROGRESS のゲームを最大 10 個取得します。このクエリでは、OpponentId-StatusDate-index インデックスを指定します。

セカンダリインデックスの詳細については、「セカンダリインデックスを使用したデータアクセス性の向上」を参照してください。

2.2: 実行中のアプリケーション (コードのウォークスルー)

このアプリケーションには 2 つのメインページがあります。

- ホームページ – このページには、簡単なログイン、新しい tic-tac-toe ゲームを作成する [CREATE] (作成) ボタン、進行中のゲームのリスト、ゲームの履歴、およびアクティブな保留中のゲームの招待が表示されます。

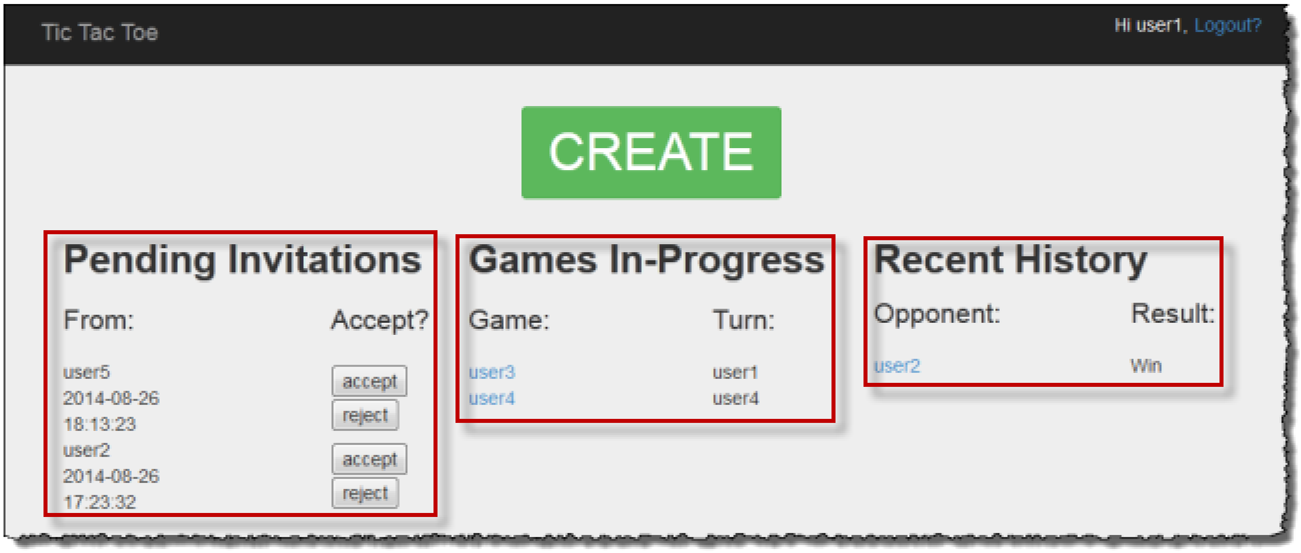
  ホームページは自動的に更新されません。リストを更新するには、ユーザーがページを更新する必要があります。
- ゲームページ – このページには、ユーザーがプレイするための、tic-tac-toe グリッドが表示されます。

  アプリケーションは、毎秒ゲームページを自動的に更新します。ブラウザの JavaScript が毎秒 Python ウェブサーバーを呼び出して、テーブルのゲーム項目が変更されたかどうかを Games テーブルに照会します。変更された場合、JavaScript はページ更新をトリガーし、更新されたボードがユーザーに表示されるようにします。

アプリケーションの動作について詳細に説明します。

ホームページ

ユーザーがログインすると、アプリケーションには、以下の 3 つの情報のリストが表示されます。



Pending Invitations

From:	Accept?
user5 2014-08-26 18:13:23	accept reject
user2 2014-08-26 17:23:32	accept reject

Games In-Progress

Game:	Turn:
user3	user1
user4	user4

Recent History

Opponent:	Result:
user2	Win

- 招待 – このリストには、ログインしているユーザーが受け入れを保留中の、他のユーザーからの最新の招待が最大 10 個表示されます。前のスクリーンショットで、user1 は user5 および user2 からの招待を保留中です。
- 進行中のゲーム – このリストには、進行中の最新のゲームが最大 10 個表示されます。これらは、ユーザーがアクティブに実行中のゲームで、そのステータスは IN_PROGRESS です。スクリーンショットでは、user1 は user3 および user4 と Tic-Tac-Toe ゲームをアクティブにプレイ中です。
- 最近の履歴 – このリストには、ユーザーが終了した最近のゲームが最大 10 個表示されます。そのステータスは FINISHED です。スクリーンショットに示したゲームで、user1 は以前に user2 とプレイしています。完了した各ゲームについて、ゲームの結果がリストに表示されます。

コードで、index 関数は (application.py で) 次の 3 つの呼び出しを行い、ゲームのステータス情報を取得します。

```
inviteGames      = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames    = controller.getGamesWithStatus(session["username"], "FINISHED")
```

これらの呼び出しはそれぞれ、Game オブジェクトでラップされた、DynamoDB からの項目のリストを返します。ビューでこれらのオブジェクトからデータを抽出することは簡単です。インデックス関数は、HTML を表示するため、これらのオブジェクトリストをビューに渡します。

```
return render_template("index.html",
                       user=session["username"],
                       invites=inviteGames,
                       inprogress=inProgressGames,
                       finished=finishedGames)
```

Tic-Tac-Toe アプリケーションは、主に DynamoDB から取得したゲームデータを格納するため、Game クラスを定義します。これらの関数は、Amazon DynamoDB の項目に関連するコードからアプリケーションの他の部分を分離できるようにするために、Game オブジェクトのリストを返します。これらの関数により、このようにしてデータストア層の詳細からアプリケーションコードを切り離すことができます。

ここで説明するアーキテクチャーパターンは、model-view-controller (MVC) UI パターンとも呼ばれます。この場合、Game オブジェクトのインスタンス (データを表す) はモデルで、HTML ページはビューです。コントローラーは 2 つのファイルに分割されます。application.py ファイルには Flask フレームワーク用のコントローラーロジックがあり、ビジネスロジックは gameController.py ファイルに分離されます。つまり、このアプリケーションでは、DynamoDB SDK に関連のあるすべてが、dynamodb フォルダ内にある独自の個別ファイルに格納されます。

3 つの関数と、それらが該当データを取得するためにグローバルセカンダリインデックスを使用して Games テーブルを照会する方法について説明します。

getGameInvites を使用して保留中のゲームの招待リストを取得します

getGameInvites 関数は保留中の最新の 10 個の招待リストを取得します。これらのゲームはユーザーによって作成されましたが、対戦相手はゲームの招待を受け入れていません。これらのゲームの場合、対戦相手が招待を受け入れるまでステータスは PENDING のままになります。対戦相手が招待を辞退した場合、アプリケーションは、対応する項目をテーブルから削除します。

関数は、以下のようなクエリを指定します。

- 関数は、以下のキー値および比較演算子とともに使用する OpponentId-StatusDate-index インデックスを指定します。
  - パーティションキーは OpponentId で、インデックスキー *user ID* を受け取ります。
  - ソートキーは StatusDate で、比較演算子およびインデックスキー値 beginswith="PENDING_" を受け取ります。

  OpponentId-StatusDate-index インデックスを使用して、ログインしているユーザーが招待されているゲーム、つまりログインしているユーザーが対戦相手であるゲームを取得します。
- クエリは結果を 10 項目に制限します。

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                            Opponent__eq=user,
                                            StatusDate__beginswith="PENDING_",
                                            index="OpponentId-StatusDate-index",
                                            limit=10)
```

このインデックスでは、OpponentId (パーティションキー) ごとに、DynamoDB が項目を StatusDate (ソートキー) により整列させて保持しています。そのため、クエリが返すゲームは最新の 10 個のゲームになります。

getGamesWithStatus を使用して特定のステータスのゲームリストの取得します

対戦相手がゲームの招待を受け入れると、ゲームのステータスは IN_PROGRESS に変わります。ゲームが完了すると、ステータスは FINISHED に変わります。

進行中または終了済みのゲームを検索するクエリは、ステータス値が異なる場合を除いて同じです。したがって、アプリケーションは getGamesWithStatus 関数を定義し、この関数はステータス値をパラメータとして受け取ります。

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

次のセクションでは進行中のゲームについて説明しますが、終了済みのゲームにも同じ説明が当てはまります。

特定のユーザーの進行中のゲームのリストには、次の両方が含まれます。

- ユーザーによってホストされた進行中のゲーム
- ユーザーが対戦相手となっている進行中のゲーム

getGamesWithStatus 関数は、毎回適切なセカンダリインデックスを使用して次の 2 つのクエリを実行します。

- この関数は Games インデックスを使用して HostId-StatusDate-index テーブルをクエリします。このインデックスのためにクエリは、プライマリキー値つまりパーティションキー (HostId) およびソートキー (StatusDate) の値と、比較演算子を指定します。

```
hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
                                               StatusDate__beginswith=status,
                                               index="HostId-StatusDate-index",
                                               limit=10)
```

比較演算子の Python の構文に注意してください。

- HostId__eq=user は等価比較演算子を指定します。
- StatusDate__beginswith=status は BEGINS_WITH 比較演算子を指定します。

- この関数は Games インデックスを使用して OpponentId-StatusDate-index テーブルをクエリします。

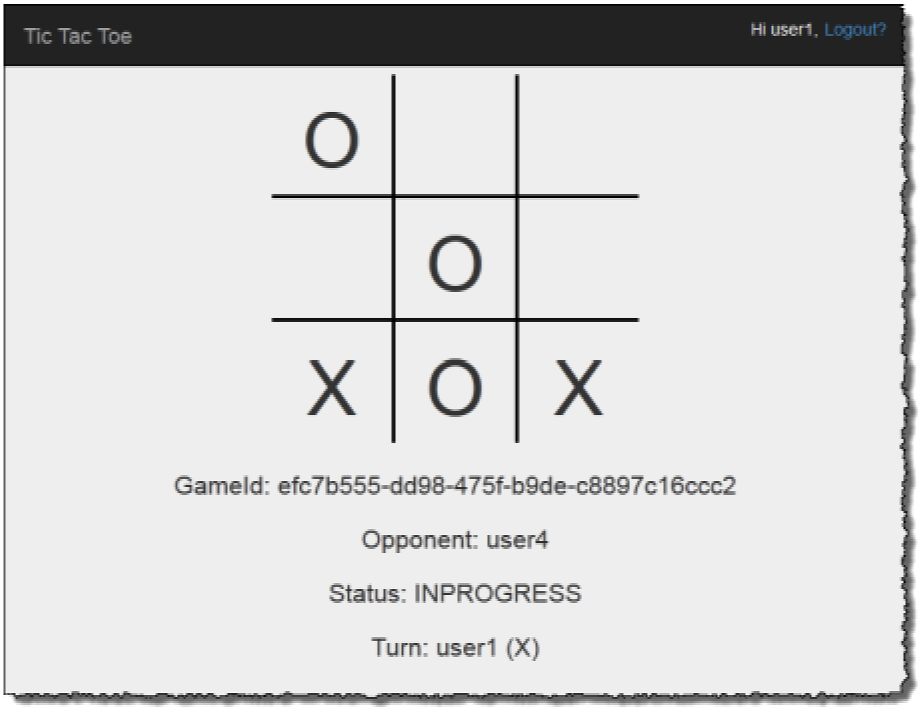
```
oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                             StatusDate__beginswith=status,
                                             index="OpponentId-StatusDate-index",
                                             limit=10)
```

- 次に、この関数は 2 つのリストを組み合わせ、ソートし、最初の 0～10 項目について Game オブジェクトのリストを作成して、呼び出し元関数 (インデックス) にリストを返します。

```
games = self.mergeQueries(hostGamesInProgress,
                        oppGamesInProgress)
return games
```

ゲームページ

ゲームページは、ユーザーが Tic-Tac-Toe ゲームをプレイする場所です。このページには、ゲームの関連情報とともに、ゲームグリッドが表示されます。次のスクリーンショットは、進行中のサンプルゲームを示しています。



GameId: efc7b555-dd98-475f-b9de-c8897c16ccc2

Opponent: user4

Status: INPROGRESS

Turn: user1 (X)

アプリケーションは次の状況でゲームページを表示します。

- ユーザーは他のユーザーを招待してゲームを作成します。

  この場合、ページはホストやゲームのステータスを PENDING としてユーザーに表示し、対戦相手が受け入れるのを待ちます。

- ユーザーは、ホームページで保留中の招待の 1 つを受け入れます。

  この場合、ページではユーザーは対戦相手として、ゲームのステータスは IN_PROGRESS として表示されます。

ボード上でのユーザーによる選択操作により、アプリケーションへのフォーム POST リクエストが生成されます。つまり、Flask は HTML フォームデータとともに selectSquare 関数を（application.py で）呼び出します。次に、この関数は updateBoardAndTurn 関数を（gameController.py で）呼び出して、次のようにゲーム項目を更新します。

- これにより、動きに固有の新しい属性が追加されます。
- Turn 属性が、次の番のユーザーに更新されます。

```
controller.updateBoardAndTurn(item, value, session["username"])
```

項目の更新が成功した場合、関数は true を返します。それ以外の場合は、false を返します。updateBoardAndTurn 関数について、以下の点に注意してください。

- この関数は、既存の項目に対する一定範囲の更新を実行するため、SDK for Python の update_item 関数を呼び出します。この関数は、DynamoDB の UpdateItem オペレーションにマッピングされます。詳細については、「UpdateItem」を参照してください。

> Note
>
> UpdateItem オペレーションと PutItem オペレーションの違いは、PutItem が項目全体を置き換えることです。詳細については、「PutItem」を参照してください。

update_item 呼び出しでは、コードは以下を識別します。

- Games テーブルのプライマリキー (ItemId)。

```
key = { "GameId" : { "S" : gameId } }
```

- 現在のユーザーの動きに固有の、追加する新しい属性とその値 (例: TopLeft="X") 。

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- 更新が実行されるために満たされる必要がある条件
  - ゲームは進行中である必要があります。つまり、StatusDate 属性値は IN_PROGRESS で始まる必要があります。
  - 現在の番は、Turn 属性で指定されている有効なユーザーの番である必要があります。
  - ユーザーが選択した四角形は使用可能である必要があります。つまり、四角形に対応する属性は存在していてはなりません。

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
    "Turn" : {"Value" : {"S" : current_player}},
    position : {"Exists" : False}}
```

ここで、関数は update_item を呼び出して項目を更新します。

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

関数に戻ると、selectSquare 関数呼び出しは次の例に示すようにリダイレクトされます。

```
redirect("/game="+gameId)
```

この呼び出しにより、ブラウザが更新されます。この更新の一環として、アプリケーションはゲームが優勝または引き分けで終了したかどうかを確認します。終了した場合、アプリケーションはそれに応じてゲーム項目を更新します。

ステップ 3: DynamoDB サービスを使用して実稼働環境のデプロイを行います

トピック

- 3.1: Amazon EC2 向けの IAM ロールを作成します
- 3.2: Amazon DynamoDB でゲームテーブルを作成します
- 3.3: tic-tac-toe アプリケーションコードのバンドルとデプロイ
- 3.4: AWS Elastic Beanstalk 環境の設定

前のセクションでは、DynamoDB local を使用して、ローカルコンピュータ上で Tic-Tac-Toe アプリケーションをデプロイし、テストを行いました。ここでは、次のようにして本稼働環境でアプリケーションをデプロイします。

- ウェブアプリケーションやサービスをデプロイ、スケーリングするための使いやすいサービスである AWS Elastic Beanstalk を使用して、アプリケーションをデプロイします。詳細については、「AWS Elastic Beanstalk への Flask アプリケーションのデプロイ」を参照してください。

  Elastic Beanstalk により、1 つ以上の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスが起動されます。Tic-Tac-Toe の設定は、そのアプリケーションが実行される Elastic Beanstalk を通じて行われます。
- Amazon DynamoDB サービスを使用して、ローカルのコンピュータ上ではなく、AWS 上に Games テーブルを作成します。

さらに、アクセス許可も設定する必要があります。DynamoDB の Games テーブルなど、ここで作成する AWS リソースは、デフォルトでプライベートになります。リソース所有者 (Games テーブルを作成した AWS アカウント) のみが、このテーブルにアクセスできます。したがって、デフォルトでは Tic-Tac-Toe アプリケーションは Games テーブルを更新することはできません。

必要なアクセス許可を付与するには、AWS Identity and Access Management (IAM) ロールを作成し、このロールに Games テーブルにアクセスするアクセス許可を付与します。Amazon EC2 インスタンスが、最初にこのロールを引き受けます。その後 AWS は、Amazon EC2 インスタンスが Tic-Tac-Toe アプリケーションに代わって Games テーブルを更新するために使用できる、一時的なセキュリティ認証情報を返します。ユーザーが Elastic Beanstalk アプリケーションを設定する際には、Amazon EC2 インスタンス、もしくは他のインスタンスが引き受けることができるIAM ロールを指定します。IAM ロールの詳細については、「Amazon EC2 Linux インスタンス用ユーザーガイド」の「Amazon EC2 の IAM ロール」を参照してください。

> Note
>
> Tic-Tac-Toe アプリケーション用の Amazon EC2 インスタンスを作成する前に、Elastic Beanstalk がインスタンスを作成する AWS リージョンを、最初に決定する必要があります。Elastic Beanstalk アプリケーションの作成が完了したら、その作成先と同じリージョン名とエンドポイントを設定ファイルで指定します。Tic-Tac-Toe アプリケーションはこのファイルの情報を使用して Games テーブルを作成し、それ以降のリクエストを特定の AWS リージョンに送信します。Elastic Beanstalk が起動する DynamoDB Games テーブルと Amazon EC2 インスタンスの両方が同じリージョンにある必要があります。　利用可能なリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon DynamoDB」を参照してください。

要約すると、Tic-Tac-Toe アプリケーションを本稼働環境にデプロイするには、以下の作業を行います。

1. IAM サービスを使用して、IAM ロールを作成します。DynamoDB アクションが Games テーブルにアクセスするためのアクセス許可を付与するポリシーを、このロールにアタッチします。
2. Tic-Tac-Toe アプリケーションコードおよび設定ファイルをバンドルし、.zip ファイルを作成します。この .zip ファイルを使用して、サーバーに配置する Tic-Tac-Toe アプリケーションコードを Elastic Beanstalk に渡します。バンドルの作成に関する詳細については、「AWS Elastic Beanstalk デベロッパーガイド」の「アプリケーションソースバンドルを作成する」を参照してください。

設定ファイル (`beanstalk.config`) で、AWS リージョンおよびエンドポイント情報を指定します。Tic-Tac-Toe アプリケーションは、この情報を使用して、通信の対象となる DynamoDB リージョンを決定します。

3. Elastic Beanstalk 環境をセットアップします。Elastic Beanstalk により、Amazon EC2 インスタンス、もしくは他のインスタンスが起動され、それらの上で Tic-Tac-Toe アプリケーションバンドルがデプロイされます。Elastic Beanstalk 環境の準備が整ったら、`CONFIG_FILE` 環境変数を追加して設定ファイル名を指定します。
4. DynamoDB テーブルを作成します。Amazon DynamoDB サービスを使用して、ローカルコンピュータ上にではなく `Games` 上に AWS テーブルを作成します。このテーブルは、文字列型の `GameId` パーティションキーで構成されたシンプルなプライマリキーを持ちます。
5. 本稼働環境でゲームをテストします。

3.1: Amazon EC2 向けの IAM ロールを作成します

Amazon EC2 型の IAM ロールを作成すると、Tic-Tac-Toe アプリケーションを実行している Amazon EC2 インスタンスが適切なロールを引き受け、`Games` テーブルへのアクセスをアプリケーションにリクエストできるようになります。ロールを作成するときに、[Custom Policy (カスタムポリシー)] オプションを選択し、次のポリシーをコピーして貼り付けます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Action":[
            "dynamodb:ListTables"
         ],
         "Effect":"Allow",
         "Resource":"*"
      },
      {
         "Action":[
            "dynamodb:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:dynamodb:us-west-2:922852403271:table/Games",
            "arn:aws:dynamodb:us-west-2:922852403271:table/Games/index/*"
         ]
      }
```

```
    ]
}
```

詳しい手順については、「IAM ユーザーガイド」の「AWS のサービスのロールを作成 (AWS Management Console)」を参照してください。

3.2: Amazon DynamoDB でゲームテーブルを作成します

ゲームのデータは、DynamoDB の Games テーブルに保存されます。テーブルが存在しない場合、アプリケーションによって自動的にテーブルが作成されます。この例では、アプリケーションで Games テーブルを作成します。

3.3: tic-tac-toe アプリケーションコードのバンドルとデプロイ

この例のステップを実行した場合、既に Tic-Tac-Toe アプリケーションをダウンロードしています。そうでない場合は、アプリケーションをダウンロードし、すべてのファイルをローカルコンピュータ上のフォルダに展開します。手順については、「ステップ 1: ローカルにデプロイおよびテストを実行します」を参照してください。

すべてのファイルを展開すると、code フォルダが作成されます。このフォルダを Elastic Beanstalk に引き渡すには、このフォルダのコンテンツを .zip ファイルとしてバンドルします。最初に、そのフォルダに設定ファイルを追加します。アプリケーションは、リージョンとエンドポイント情報を使用して、指定されたリージョンで DynamoDB テーブルを作成します。さらに、指定されたエンドポイントを使用しながら、それ以降のテーブルオペレーションをリクエストします。

1. Tic-Tac-Toe アプリケーションをダウンロードしたフォルダに切り替えます。
2. アプリケーションのルートフォルダで、次のコンテンツを使用して beanstalk.config という名前のテキストファイルを作成します。

   ```
   [dynamodb]
   region=<AWS region>
   endpoint=<DynamoDB endpoint>
   ```

   たとえば、次のコンテンツを使用します。

   ```
   [dynamodb]
   region=us-west-2
   endpoint=dynamodb.us-west-2.amazonaws.com
   ```

使用可能なリージョンのリストについては、Amazon Web Services 全般リファレンスの「Amazon DynamoDB」を参照してください

Important

設定ファイルで指定されたリージョンは、Tic-Tac-Toe アプリケーションが DynamoDB の Games テーブルを作成する場所です。次のセクションで説明する Elastic Beanstalk アプリケーションを、同じリージョンで作成する必要があります。

Note

Elastic Beanstalk アプリケーションを作成する際には、環境タイプを選択しながら、その環境を起動するリクエストを行います。Tic-Tac-Toe サンプルアプリケーションをテストするには、[Single Instance (単一インスタンス)] 環境タイプを選択し、それ以降をスキップして、次のステップに進みます。
ただし、[Load balancing, autoscaling (ロードバランシングでオートスケーリング)] 環境タイプでは、高可用性でスケーラブルな環境が提供されます。これは、他のアプリケーションを作成、デプロイする場合に検討してください。この環境タイプを選択する場合、UUID を生成し、次に示すように設定ファイルに追加する必要があります。

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
[flask]
secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
```

クライアント/サーバー通信でサーバーが応答を送信するときは、セキュリティのため、サーバーは、クライアントが次のリクエストでサーバーに送り返す署名済み Cookie を送信します。サーバーが 1 台のみの場合、サーバーは、起動時にローカルに暗号化キーを生成できます。多くのサーバーがある場合、それらのサーバーはすべて同じ暗号化キーを知る必要があります。そうしない場合、ピアサーバーによって設定された Cookie を読み取ることができません。secret_key を設定ファイルに追加することで、この暗号化キーを使用するようにすべてのサーバーに伝えます。

3. アプリケーションのルートフォルダのコンテンツを、(beanstalk.config ファイルも含めながら) Zip で圧縮します。たとえば TicTacToe.zip などとします。

4. Amazon Simple Storage Service (Amazon S3) バケットに、.zip ファイルをアップロードします。次のセクションで、この .zip ファイルを 1 つ以上のサーバーにアップロードするために Elastic Beanstalk に渡します。

   Amazon S3 バケットにアップロードする方法については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの作成」および「バケットへのオブジェクトの追加」を参照してください。

3.4: AWS Elastic Beanstalk 環境の設定

このステップでは、環境など各コンポーネントの集合である Elastic Beanstalk アプリケーションを作成します。この例では、1 つの Amazon EC2 インスタンスを起動して、Tic-Tac-Toe アプリケーションをデプロイし実行します。

1. 環境をセットアップするには、次のカスタム URL を入力して Elastic Beanstalk コンソールをセットアップします。

```
https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
newApplication
?applicationName=TicTacToeyour-name
&solutionStackName=Python
&sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
&environmentType=SingleInstance
&instanceType=t1.micro
```

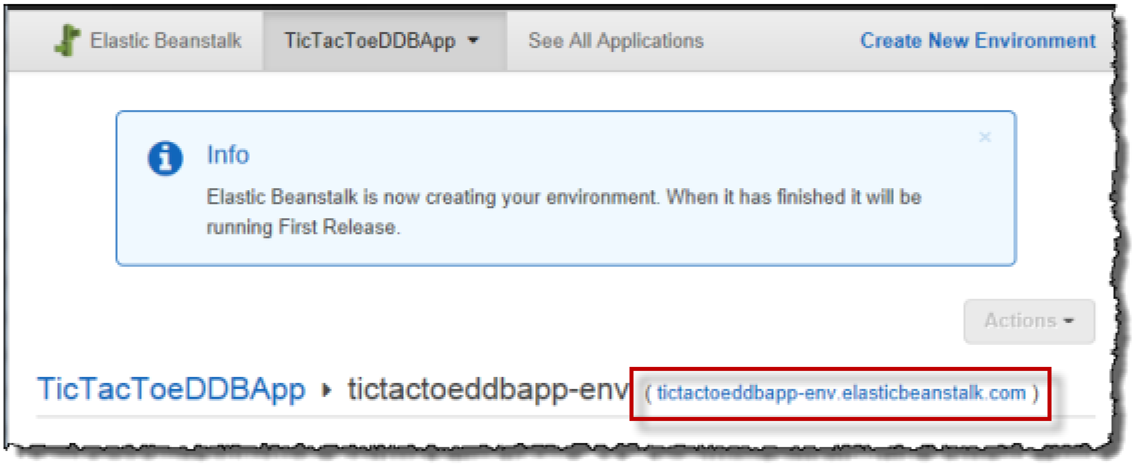
   カスタム URL の詳細については、AWS Elastic Beanstalk デベロッパーガイドの「Launch Now URL の作成」を参照してください URL については、次の点に注意してください。

   - AWS リージョンの(設定ファイルで指定したものと同じ)名前、Amazon S3 バケット名、およびオブジェクト名を指定する必要があります。
   - テストでは、URL は SingleInstance 環境タイプ、および t1.micro をインスタンスタイプとしてリクエストします。
   - アプリケーション名は一意である必要があります。そのため、前の URL では、applicationName の前に名前を追加することをお勧めします。

   これにより、Elastic Beanstalk コンソールが開きます。場合によっては、サインインが必要にあることがあります。

2. Elastic Beanstalk コンソールで、[Review and Launch] (確認して起動) を選択し、[Launch] (起動) を選択します。

3. 今後の参照用に URL を書き留めてください。この URL により、Tic-Tac-Toe アプリケーションのホームページが開きます。



Info

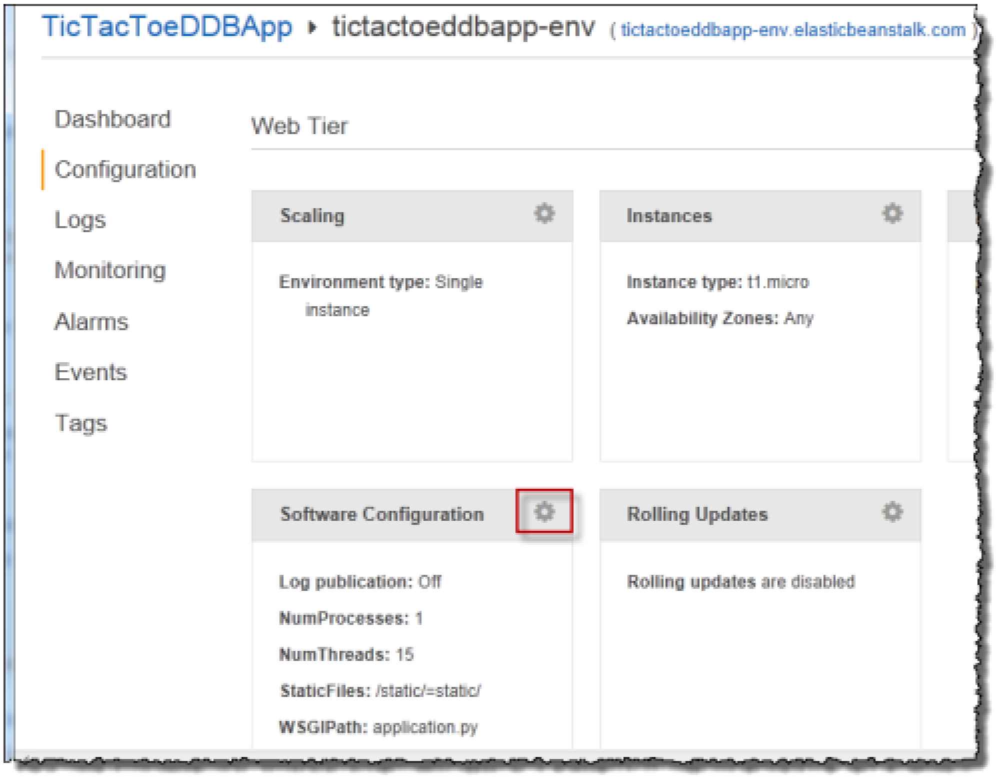
Elastic Beanstalk is now creating your environment. When it has finished it will be running First Release.

TicTacToeDDBApp ▸ tictactoeddbapp-env (tictactoeddbapp-env.elasticbeanstalk.com)

4. Tic-Tac-Toe アプリケーションを設定し、設定ファイルの場所を指定します。

   Elastic Beanstalk によりアプリケーションが作成されたら、[Configuration] (設定) を選択します。

   a. 次のスクリーンショットに示すように、[Software Configuration (ソフトウェア設定)] の横にある歯車のアイコンを選択します。



Environment type: Single instance

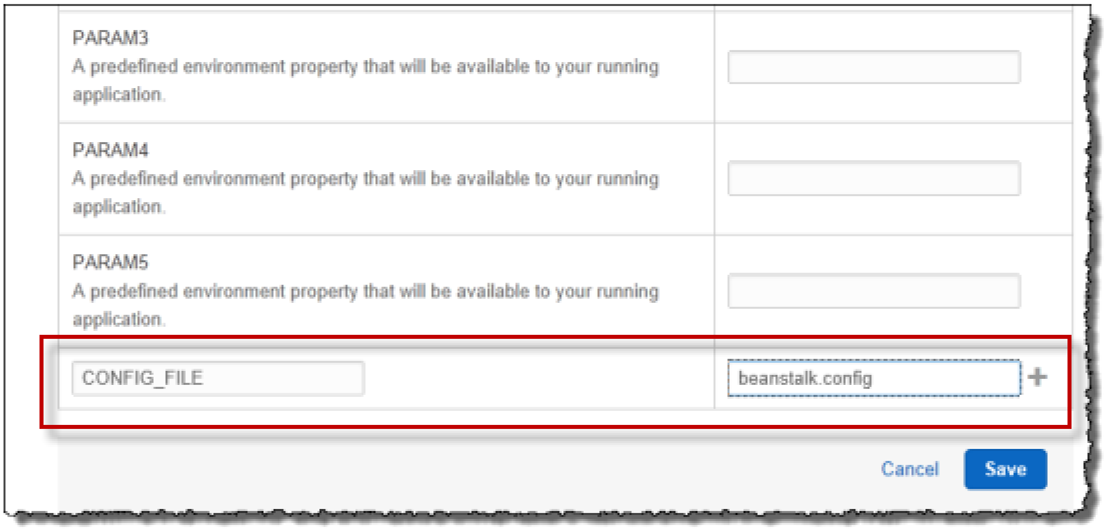
Instance type: t1.micro
Availability Zones: Any

Log publication: Off
NumProcesses: 1
NumThreads: 15
StaticFiles: /static/=static/
WSGIPath: application.py

Rolling updates are disabled

b. [Environment Properties (環境プロパティ)] セクションの最後で、**`CONFIG_FILE`** とその値 **`beanstalk.config`** を入力し、[Save (保存)] を選択します。

この環境の更新が完了するには数分かかる場合があります。



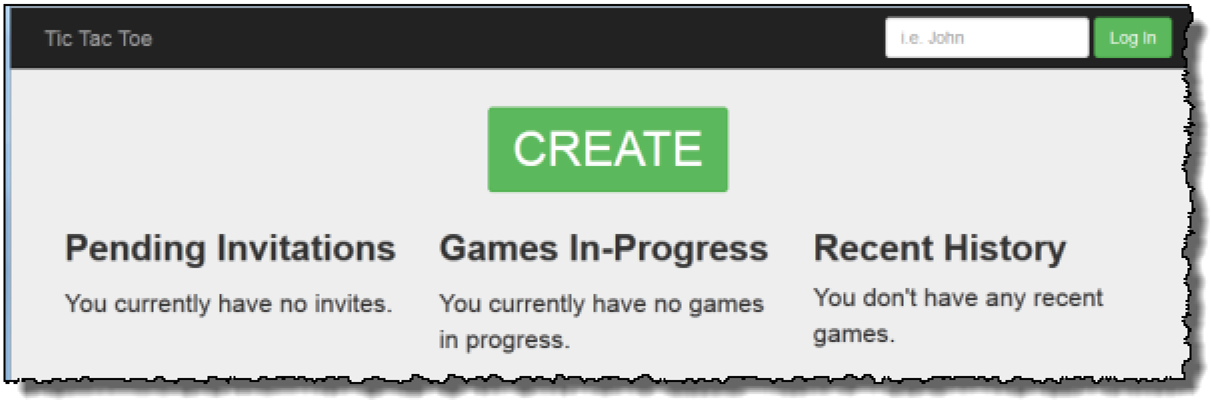
PARAM3 A predefined environment property that will be available to your running application.	
PARAM4 A predefined environment property that will be available to your running application.	
PARAM5 A predefined environment property that will be available to your running application.	
CONFIG_FILE	beanstalk.config

更新が完了したら、ゲームをプレイできます。

5. ブラウザで、前のステップでコピーした URL を、以下の例に示すように入力します。

```
http://<pen-name>.elasticbeanstalk.com
```

これにより、アプリケーションのホームページが開きます。



Pending Invitations

You currently have no invites.

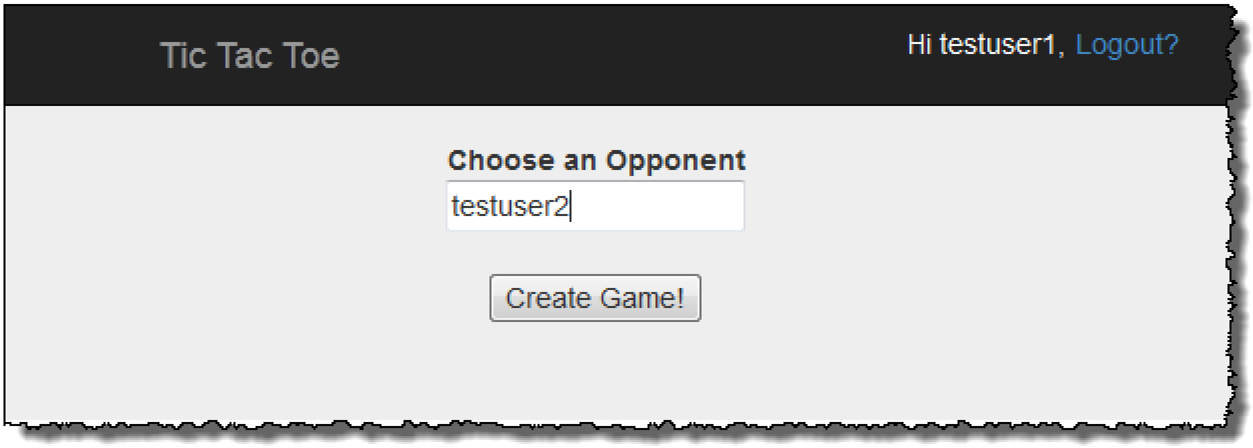
Games In-Progress

You currently have no games in progress.

Recent History

You don't have any recent games.

6. testuser1 としてログインし、[作成] を選択して新しい Tic-Tac-Toe ゲームを開始します。
7. [Choose an Opponent] (対戦相手を選択する) ボックスに **`testuser2`** を入力します。



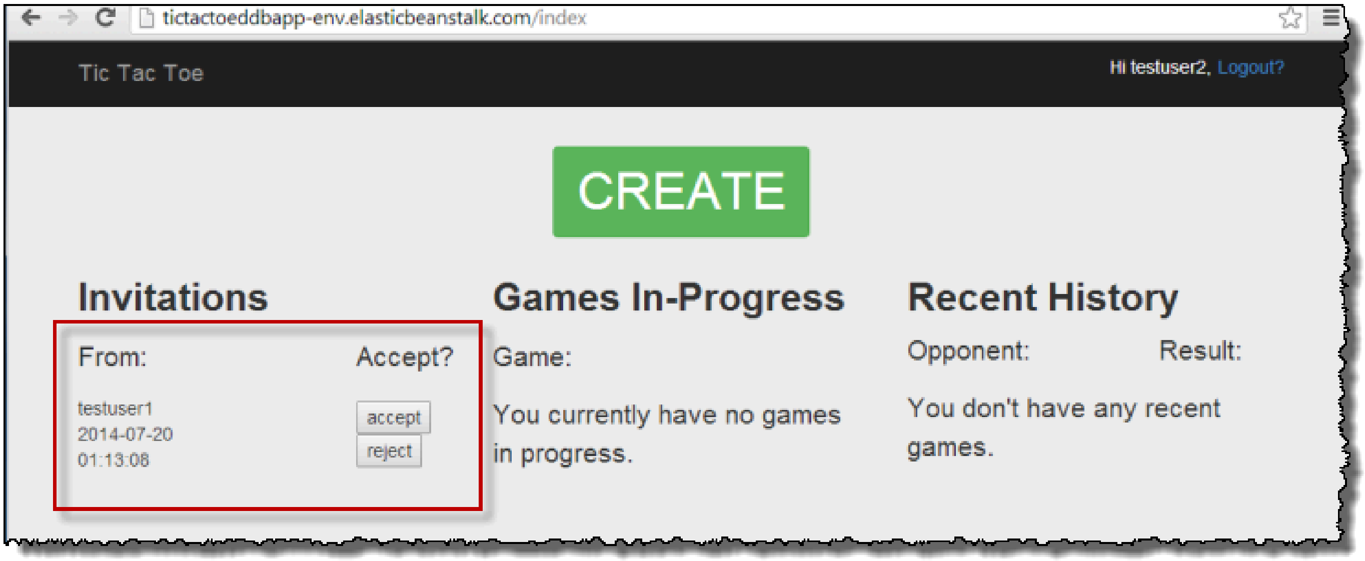
Choose an Opponent: testuser2

8. 別のブラウザウィンドウを開きます。

   ブラウザウィンドウのすべての Cookie を消去し、同じユーザーとしてログインしないようにします。
9. 以下の例に示すように、同じ URL を入力してアプリケーションのホームページを開きます。

```
http://<env-name>.elasticbeanstalk.com
```

10. testuser2 としてログインします。
11. 保留中の招待のリストで、testuser1 からの招待に対して [accept (受け入れる)] を選択します。



Invitations

From: Accept?

testuser1
2014-07-20
01:13:08

Games In-Progress

Game:

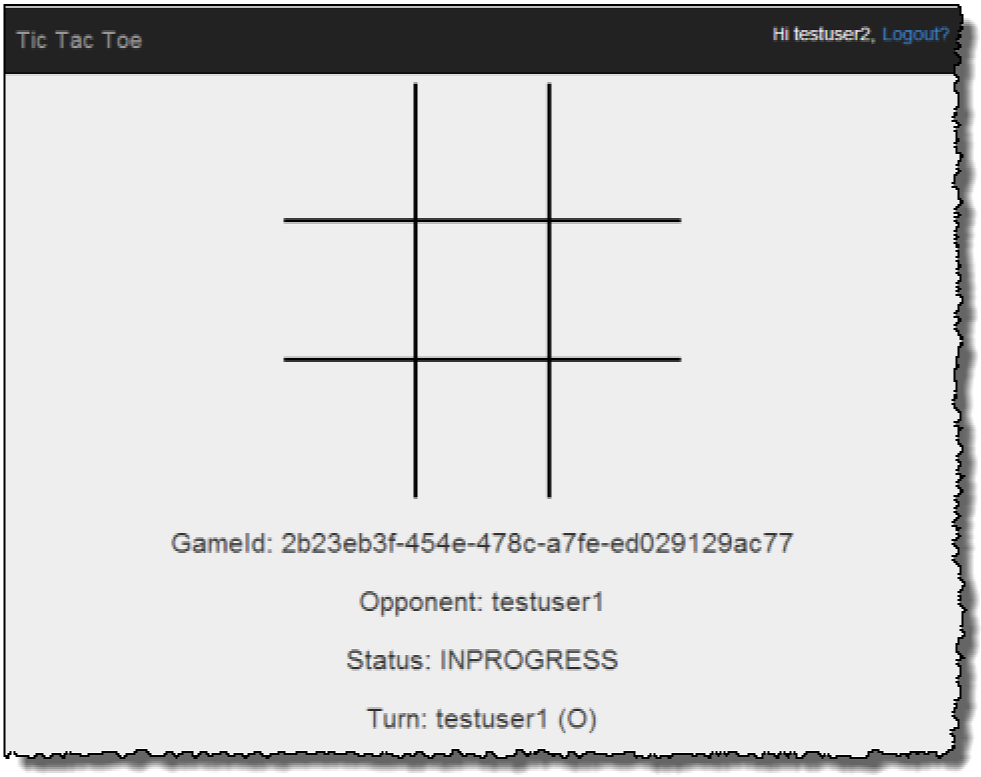
You currently have no games in progress.

Recent History

Opponent: Result:

You don't have any recent games.

12. これで、ゲームページが表示されます。



GameId: 2b23eb3f-454e-478c-a7fe-ed029129ac77

Opponent: testuser1

Status: INPROGRESS

Turn: testuser1 (O)

testuser1 と testuser2 の両者がゲームをプレイできます。動きがあるごとに、アプリケーションは `Games` テーブルの対応する項目に動きを保存します。

ステップ 4: リソースをクリーンアップする

これで Tic-Tac-Toe アプリケーションのデプロイとテストを完了しました。このアプリケーションには、ユーザー認証を除き、Amazon DynamoDB でのウェブアプリケーション開発プロセスがすべて網羅されています。このアプリケーションは、ゲームの作成時にプレーヤーの名前を追加するためにのみ、ホームページのログイン情報を使用します。本稼働アプリケーションでは、ユーザーログインおよび認証を実行するために必要なコードを追加します。

テストを終了したら、料金が発生しないようにするため、Tic-Tac-Toe アプリケーションのテスト用に作成したリソースを削除できます。

作成したリソースを削除するには

1. DynamoDB で作成した `Games` テーブルを削除します。
2. Elastic Beanstalk 環境を終了し、Amazon EC2 インスタンスを解放します。
3. 作成した IAM ロールを削除します。
4. Amazon S3 で作成したオブジェクトを削除します。

AWS Data Pipeline を使用して DynamoDB データをエクスポートおよびインポートする

AWS Data Pipeline を使用して、DynamoDB テーブルから Amazon S3 バケット内のファイルにデータをエクスポートできます。またコンソールを使用して、Amazon S3 から同じ AWS リージョンまたは別のリージョンにある DynamoDB テーブルにデータをインポートもできます。

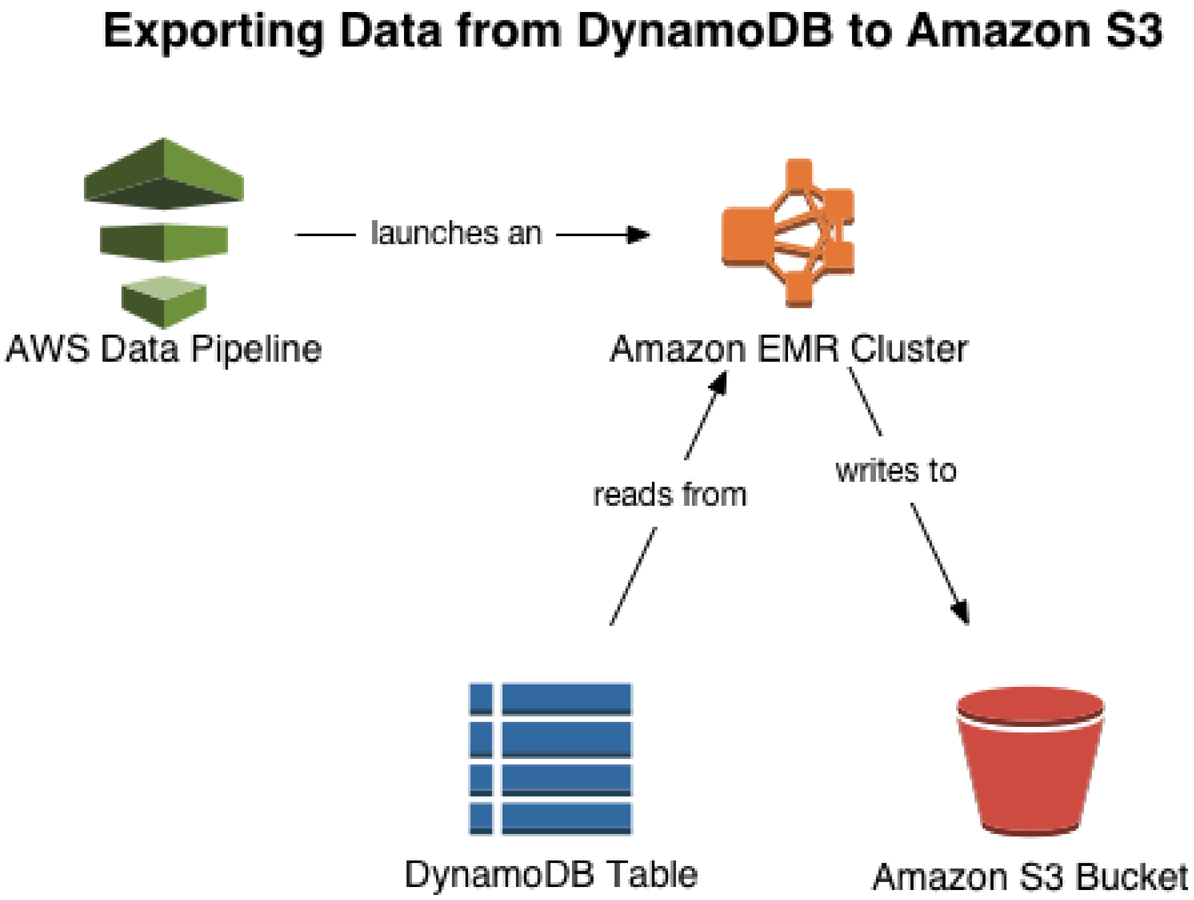
> **Note**
>
> DynamoDB コンソールは、Amazon S3 からのインポートと Amazon S3 へのエクスポートをネイティブでサポートするようになりました。これらのフローは AWS Data Pipeline インポートフローと互換性がありません。詳細については、「Amazon S3 からの DynamoDB データのインポート: 仕組み」、「Amazon S3 への DynamoDB データのエクスポート: 仕組み」、およびブログ記事「Export Amazon DynamoDB table data to your data lake in Amazon S3」(Amazon DynamoDB テーブル データを Amazon S3 のデータレイクにエクスポートする) を参照してください。

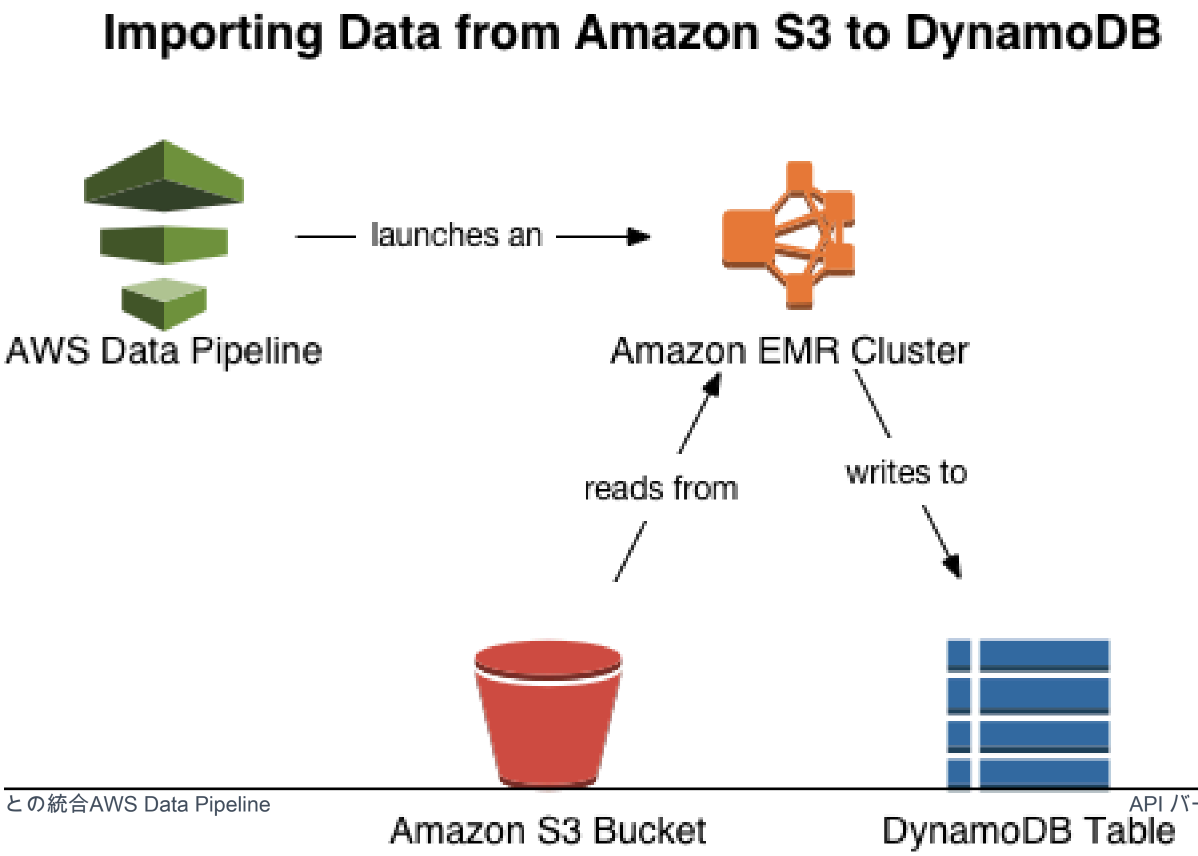
データのエクスポートおよびインポート機能は多くシナリオで役立ちます。たとえば、テスト目的で一連のベースラインデータを保守するとします。ベースラインデータを DynamoDB テーブルに入力し、Amazon S3 にエクスポートできます。続いて、テストデータを変更するアプリケーションを実行した後、Amazon S3 から DynamoDB テーブルにインポートして戻すことで、ベースラインデータを「リセット」できます。別の例としては、データの誤った削除や誤った `DeleteTable` オペレーションもあります。このような場合は、Amazon S3 にある前回のエクスポートファイルからデータを復元できます。ある AWS リージョン内の DynamoDB テーブルからデータをコピーして Amazon S3 に保存し、その後に Amazon S3 から別のリージョン内の同じ DynamoDB テーブルにインポートすることもできます。別のリージョン内のアプリケーションは、最も近い DynamoDB エンドポイントにアクセスしてデータの独自のコピーを操作できるので、ネットワークレイテンシーが短くなります。

⚠ Important

DynamoDB のバックアップと復元は、フルマネージドの機能です。本稼働アプリケーションのパフォーマンスや可用性に影響を与えることなく、数メガバイトから数百テラバイトまでのデータを持つテーブルをバックアップできます。AWS Management Console でのワンクリック操作か、単一の API コールだけでテーブルを復元できます。AWS Data Pipeline ではなく、DynamoDB のネイティブの Backup および復元機能を使用することを強くお勧めします。詳細については、「DynamoDB のオンデマンドバックアップおよび復元の使用」を参照してください。

次の図表に、AWS Data Pipeline を使用した DynamoDB データのエクスポートおよびインポートの概要を示します。


Exporting Data from DynamoDB to Amazon S3
AWS Data Pipeline
launches an
Amazon EMR Cluster
reads from
writes to
DynamoDB Table
Amazon S3 Bucket


Importing Data from Amazon S3 to DynamoDB
AWS Data Pipeline
launches an
Amazon EMR Cluster
reads from
writes to
Amazon S3 Bucket
DynamoDB Table

DynamoDB テーブルをエクスポートするには、AWS Data Pipeline コンソールを使用して新しいパイプラインを作成します。パイプラインによって Amazon EMR クラスターが起動され、実際のエクスポートが実行されます。Amazon EMR は、DynamoDB からデータを読み取り、Amazon S3 バケット内のエクスポートファイルにデータを書き込みます。

このプロセスは、データが Amazon S3 バケットから読み込まれて DynamoDB テーブルに書き込まれることを除き、インポートでも同様です。

⚠ Important

DynamoDB データのエクスポートまたはインポート時、基礎となる次の AWS サービスの使用に対して追加コストが発生します。

- AWS Data Pipeline — Import/Export のワークフローを管理します。
- Amazon S3 - DynamoDB に対してエクスポートまたはインポートするデータを格納します。
- Amazon EMR - マネージド Hadoop クラスターを実行して、Amazon S3 と DynamoDB の間で読み込みと書き込みを行います。クラスター構成は、1 つの `m3.xlarge` インスタンスリーダーノードと 1 つの `m3.xlarge` インスタンスコアノードです。

詳細については、「AWS Data Pipeline の料金」、「Amazon EMR の料金」、および「Amazon S3 の料金」を参照してください。

データをエクスポートおよびインポートデータするための前提条件

データのエクスポートとインポートに AWS Data Pipeline を使用するときは、パイプラインによって実行できるアクションと消費できるリソースを指定する必要があります。許可されるそれらのアクションとリソースは AWS Identity and Access Management (IAM) ロールを使用して定義します。

また、IAM ポリシーを作成し、ユーザー、ロール、またはグループにアタッチすることでアクセスを制御できます。これらのポリシーでは、DynamoDB データに対してインポートおよびエクスポートが許可されるユーザーを指定できます。

⚠ Important

AWS Management Console の外部で AWS を操作するには、プログラマチックアクセス権が必要です。プログラマチックアクセス権を付与する方法は、AWS にアクセスしているユーザーのタイプによって異なります。
ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラマチックリクエストに署名します。	使用するインターフェイス用の手引きに従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「AWS IAM Identity Center を使用するための AWS CLI の設定」を参照してください。 • AWS SDK、ツール、および AWS API については、AWS SDK とツールリファレンスガイドの「IAM Identity Center 認証」を参照してください。
IAM	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラムによるリクエストに署名します。	「IAM ユーザーガイド」の「AWS リソースでの一時的な認証情報の使用」の指示に従ってください。

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	(非推奨) 長期的な認証情報を使用して、AWS CLI、AWS SDK、AWS API へのプログラムによるリクエストに署名します。	使用するインターフェイス用の手順に従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDK とツールについては、AWS SDK とツールリファレンスガイドの「長期認証情報を使用して認証する」を参照してください。 • AWS API については、IAM ユーザーガイドの「IAM ユーザーのアクセスキーの管理」を参照してください。

AWS Data Pipeline の IAM ロールの作成

AWS Data Pipeline を使用するには、次の IAM ロールが AWS アカウントにあることが必要です。

- DataPipelineDefaultRole — パイプラインによって自動的に実行されるアクション。
- DataPipelineDefaultResourceRole — パイプラインによって自動的にプロビジョンされる AWS リソース。DynamoDB データのエクスポートとインポートの場合、これらのリソースには Amazon EMR クラスターと、そのクラスターに関連付けられている Amazon EC2 インスタンスが含まれます。

これまでに AWS Data Pipeline を使用したことがない場合は、DataPipelineDefaultRole と DataPipelineDefaultResourceRole を自分で作成する必要があります。作成したそれらのロールを使用して、必要に応じていつでも DynamoDB データをエクスポートしたりインポートしたりできます。

> **Note**
>
> 以前に AWS Data Pipeline コンソールを使用してパイプラインを作成した場合、DataPipelineDefaultRole と DataPipelineDefaultResourceRole はその時点で自動的に作成されています。その場合、これ以上の操作は必要ありません。このセクションをスキップし、 コンソールを使用したパイプラインの作成を開始できます。詳細については、「DynamoDB から Amazon S3 にデータをエクスポートする」および「Amazon S3 から DynamoDB にデータをインポートする」を参照してください。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. コンソールダッシュボードで [Roles] (ロール) をクリックします。
3. [Create Role] (ロールの作成) をクリックし、次の操作を実行します。
   a. [AWS のサービス] の信頼されたエンティティで、[データパイプライン] を選択します。
   b. [Select your use case] (ユースケースの選択) パネルで、[Data Pipeline] (データパイプライン)、[Next: Permissions] (次のステップ: アクセス許可) の順に選択します。
   c. `AWSDataPipelineRole` ポリシーが自動的にアタッチされることに注意してください。[Next: Review] (次のステップ: レビュー) を選択します。
   d. [Role name] (ロール名) フィールドで、ロール名として「`DataPipelineDefaultRole`」と入力し、[Create role] (ロールの作成) を選択します。
4. [ロールの作成] をクリックし、次の操作を実行します。
   a. [AWS のサービス] の信頼されたエンティティで、[データパイプライン] を選択します。
   b. [Select your use case] (ユースケースの選択) パネルで、[EC2 Role for Data Pipeline] (データパイプラインの EC2 ロール)、[Next: Permissions] (次のステップ: 許可) の順に選択します。
   c. `AmazonEC2RoleForDataPipelineRole` ポリシーが自動的にアタッチされることに注意してください。[Next: Review] (次のステップ: レビュー) を選択します。

d. [ロール名] フィールドで、ロール名として「DataPipelineDefaultResourceRole」と入力し、[ロールの作成] を選択します。

これらのロールを作成できたので、DynamoDB コンソールを使用したパイプラインの作成を開始できます。詳細については、「DynamoDB から Amazon S3 にデータをエクスポートする」および「Amazon S3 から DynamoDB にデータをインポートする」を参照してください。

AWS Identity and Access Management を使用してエクスポートおよびインポートタスクを実行するためのアクセス許可の、ユーザーおよびグループへの付与

他のユーザー、ロール、またはグループに DynamoDB テーブルデータのエクスポートとインポートを許可するには、IAM ポリシーを作成し、指定したユーザーまたはグループにアタッチできます。ポリシーには、これらのタスクを実行するために必要なアクセス権限のみが含まれます。

完全なアクセス権の付与

次の手順では、AWS マネージドポリシー (`AmazonDynamoDBFullAccess`、`AWSDataPipeline_FullAccess`) と Amazon EMR インラインポリシーをユーザーにアタッチする方法について説明します。これらのマネージドポリシーは、AWS Data Pipeline および DynamoDB リソースへのフルアクセスを提供し、Amazon EMR インラインポリシーとともに使用することで、ユーザーはこのドキュメントで説明するアクションを実行できます。

> **Note**
>
> 提案されたアクセス許可の範囲を制限するために、上記のインラインポリシーは、`dynamodbdatapipeline` タグの使用を強制しています。この制限なしにこのドキュメントを利用する場合は、該当するポリシーの `Condition` のセクションを削除してください。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. IAM コンソールダッシュボードで [ユーザー] をクリックし、変更するユーザーを選択します。
3. [Permissions] (アクセス権限) タブで [Add Policy] (ポリシーの追加) をクリックします。
4. [Attach permissions] (アクセス許可の付与) パネルで、[Attach existing policies directly] (既存のポリシーを直接アタッチ) を選択します。

5. AmazonDynamoDBFullAccess と AWSDataPipeline_FullAccess の両方を選択して、[Next: Review] (次のステップ: 確認) をクリックします。
6. [Add permissions] (アクセス許可の追加) をクリックします。
7. [Permissions] (アクセス権限) タブで [Add inline policy] (インラインポリシーの追加) を選択します。
8. [Create a policy] (ポリシーの作成) ページで JSON タブをクリックします。
9. 以下の内容を貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EMR",
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:DescribeStep",
                "elasticmapreduce:DescribeCluster",
                "elasticmapreduce:RunJobFlow",
                "elasticmapreduce:TerminateJobFlows"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "elasticmapreduce:RequestTag/dynamodbdatapipeline": "false"
                }
            }
        }
    ]
}
```

10. [Review policy] (ポリシーの確認) をクリックします。
11. 名前フィールドに「EMRforDynamoDBDataPipeline」と入力します。
12. [Create policy] (ポリシーの作成) をクリックします。

> Note
>
> 同様の手順を使用して、この管理ポリシーをユーザーではなく、グループにアタッチすることもできます。

特定の DynamoDB テーブルへのアクセスを制限する

アクセスを制限し、ユーザーがテーブルのサブセットのエクスポートまたはインポートのみできるようにするには、カスタマイズした IAM ポリシードキュメントを作成する必要があります。「完全なアクセス権の付与」で説明されているプロセスをカスタムポリシーの開始点として使用し、このポリシーを変更して、指定したテーブルのみの操作をユーザーに許可できます。

例えば、ユーザーに Forum、Thread、Reply のテーブルに対してのみエクスポートまたはインポートを許可するとします。この手順では、ユーザーがそれらテーブルを操作できるが、それ以外は操作できないようにするカスタムポリシーを作成する方法を示します。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. コンソールダッシュボードから [Policies] (ポリシー) をクリックし、[Create Policy] (ポリシーの作成) をクリックします。
3. [ポリシーの作成] パネルで、[AWS マネージドポリシーをコピー] に移動し、[選択] をクリックします。
4. [AWS マネージドポリシーをコピー] パネルで、`AmazonDynamoDBFullAccess` に移動し、[選択] をクリックします。
5. [Review Policy] (ポリシーの確認) パネルで、以下の作業を行います。

   a. 自動生成される [Policy Name] (ポリシー名) および [Description] (説明) を確認します。必要に応じて、これらの値を変更できます。

   b. [Policy Document] (ポリシードキュメント) テキストボックスで、特定のテーブルへのアクセスを制限するポリシーを編集します。デフォルトでは、ポリシーはすべてのテーブルの DynamoDB アクションをすべて許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:DeleteAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "cloudwatch:PutMetricAlarm",
                "dynamodb:*",
                "sns:CreateTopic",
                "sns:DeleteTopic",
                "sns:ListSubscriptions",
                "sns:ListSubscriptionsByTopic",
                "sns:ListTopics",
                "sns:Subscribe",
                "sns:Unsubscribe"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Sid": "DDBConsole"
        },

...remainder of document omitted...
```

ポリシーを制限するには、最初に次の行を削除します。

```
                 "dynamodb:*",
```

次に、Forum、Thread、および Reply テーブルへのアクセスのみを許可する新しい Action を構築します。

```
{
    "Action": [
        "dynamodb:*"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:dynamodb:us-west-2:123456789012:table/Forum",
        "arn:aws:dynamodb:us-west-2:123456789012:table/Thread",
        "arn:aws:dynamodb:us-west-2:123456789012:table/Reply"
    ]
},
```

Note

us-west-2 は DynamoDB テーブルがあるリージョンに置き換えてください。123456789012 を使用する AWS アカウント番号で置き換えます。

最後に、ポリシードキュメントに新しい Action を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Forum",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Thread",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Reply"
            ]
        },
        {
            "Action": [
                "cloudwatch:DeleteAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "cloudwatch:PutMetricAlarm",
                "sns:CreateTopic",
                "sns:DeleteTopic",
                "sns:ListSubscriptions",
                "sns:ListSubscriptionsByTopic",
                "sns:ListTopics",
                "sns:Subscribe",
                "sns:Unsubscribe"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Sid": "DDBConsole"
        },

...remainder of document omitted...
```

6. ポリシーの設定が正しいことを確認したら、[Create Policy] (ポリシーの作成) をクリックします。

ポリシーを作成したら、ユーザーにポリシーをアタッチできます。

1. IAM コンソールダッシュボードで [ユーザー] をクリックし、変更するユーザーを選択します。
2. [Permissions] (アクセス許可) タブで [Attach Policy] (ポリシーのアタッチ) をクリックします。
3. [Attach Policy] (ポリシーのアタッチ) パネルでポリシー名を選択し、[Attach Policy] (ポリシーのアタッチ) をクリックします。

> Note
>
> 同様の手順を使用して、ポリシーをユーザーではなくグループにアタッチすることもできます。

DynamoDB から Amazon S3 にデータをエクスポートする

このセクションでは、1 つ以上の DynamoDB テーブルから Amazon S3 バケットにデータをエクスポートする方法について説明します。エクスポートを実行する前に、Amazon S3 バケットを作成する必要があります。

> Important
>
> これまでに AWS Data Pipeline を使用したことがない場合は、この手順を実行する前に 2 つの IAM ロールを設定する必要があります。詳細については、「AWS Data Pipeline の IAM ロールの作成」を参照してください。

1. AWS Management Console にサインインして AWS Data Pipeline コンソール (https://console.aws.amazon.com/datapipeline/) を開きます。
2. 現在の AWS リージョンにパイプラインがない場合は、[Get started now] (今すぐ始める) を選択します。

   それ以外の場合、少なくとも 1 つのパイプラインがある場合は、[Create new pipeline] (新しいパイプラインの作成) を選択します。
3. [Create Pipeline] (パイプラインの作成) ページで、次の操作を実行します。

   a. [Name] (名前) フィールドで、パイプラインの名前を入力します。例: `MyDynamoDBExportPipeline`。

b. [Source] (ソース) パラメータで、[Build using a template] (テンプレートを使用してビルドする) を選択します。ドロップダウンテンプレートのリストから、[Export DynamoDB table to S3] (DynamoDB テーブルを S3 にエクスポートする) を選択します。

c. [Source DynamoDB table name] (ソース DynamoDB テーブル名) フィールドに、エクスポートする DynamoDB テーブルの名前を入力します。

d. [Output S3 Folder] (出力 S3 フォルダ) テキストボックスに、エクスポートファイルが書き込まれる Amazon S3 の URI を入力します。例: `s3://mybucket/exports`

   この URI の形式は `s3://`*`bucketname`*`/`*`folder`* で、次のような構成になっています。

   - `bucketname` は Amazon S3 バケットの名前です。
   - `folder` はそのバケット内のフォルダーの名前です。フォルダーが存在しない場合は、自動的に作成されます。フォルダーの名前を指定しない場合は、`s3://`*`bucketname`*`/`*`region`*`/`*`tablename`* という形式で名前が付けられます。

e. [S3 location for logs] (S3 ログの場所) テキストボックスに、エクスポート用のログファイルが書き込まれる Amazon S3 の URI を入力します。例: `s3://mybucket/logs/`

   [S3 ログフォルダ] の URI 形式は [出力 S3 フォルダ] と同じです。URI はフォルダに解決される必要があります。ログファイルを S3 バケットの最上位に書き込むことはできません。

4. キー `dynamodbdatapipeline` と値 `true` を含むタグを追加します。

5. すべての設定が正しいことを確認したら、[アクティブ化] をクリックします。

これで、パイプラインが作成されます。このプロセスが完了するまでに数分かかることがあります。AWS Data Pipeline コンソールで進捗状況をモニターすることができます。

エクスポートが完了したら、Amazon S3 コンソールに移動して、エクスポートファイルを表示できます。出力ファイル名は、`ae10f955-fb2f-4790-9b11-fbfea01a871e_000000` のように識別子の値で、拡張子はありません。このファイルの内部形式については、「AWS Data Pipeline Developer Guide」の「File Structure」を参照してください。

Amazon S3 から DynamoDB にデータをインポートする

このセクションでは、すでに DynamoDB テーブルからデータをエクスポートし、エクスポートファイルが Amazon S3 バケットに書き込まれていることを前提とします。このファイルの内部形式については、「AWS Data Pipeline Developer Guide」の「File Structure」を参照してください。これは、AWS Data Pipeline を使用して DynamoDB でインポートできる唯一のファイル形式です。

ソーステーブルという用語は、データのエクスポート元となったテーブルに使用します。インポート先テーブルという用語は、データのインポート先となるテーブルに使用します。エクスポートファイルから Amazon S3 にデータをインポートできます。ただし、次のすべての条件を満たしていることが前提です。

- インポート先テーブルがすでに存在する (インポートプロセスによってテーブルは作成されません。)
- インポート先テーブルとソーステーブルのキースキーマが同じ

インポート先テーブルは空である必要はありません。ただしインポートプロセスでは、インポート先テーブルのデータ項目のうち、エクスポートファイルの項目とキーが同じものはすべて置き換えられます。たとえば、Customer テーブルに CustomerId というキーがあり、そのテーブルに 3 つの項目 (CustomerId 1、2、3) のみがあるとします。エクスポートファイルにも CustomerID 1、2、3 のデータ項目がある場合、インポート先テーブルの項目はエクスポートファイルのものと置き換えられます。エクスポートファイルに CustomerId 4 のデータ項目もある場合、その項目がインポート先テーブルに追加されます。

インポート先テーブルは異なる AWS リージョンに置くことができます。たとえば、米国西部 (オレゴン) リージョンに Customer テーブルがあり、そのデータを Amazon S3 にエクスポートするとします。その後、そのデータを欧州 (アイルランド) リージョンにある同一の Customer テーブルにインポートできます。これは、クロスリージョンのエクスポートとインポートと呼ばれています。AWS リージョンのリストについては、「AWS 全般のリファレンス」の「リージョンとエンドポイント」を参照してください。

AWS Management Console では、一度に複数のソーステーブルをエクスポートできます。ただし、一度にインポートできるのは 1 つのテーブルのみです。

1. AWS Management Console にサインインして AWS Data Pipeline コンソール (https://console.aws.amazon.com/datapipeline/) を開きます。
2. (省略可能) クロスリージョンのインポートを実行する場合、ウィンドウの右上隅にあるインポート先リージョンを選択します。
3. [Create new pipeline] (新しいパイプラインの作成) を選択します。
4. [Create Pipeline] (パイプラインの作成) ページで、次の操作を実行します。
    a. [Name] (名前) フィールドで、パイプラインの名前を入力します。例: `MyDynamoDBImportPipeline`。

b. [Source] (ソース) パラメータで、[Build using a template] (テンプレートを使用してビルドする) を選択します。ドロップダウンテンプレートのリストから、[Import DynamoDB backup data from S3] (S3 から DynamoDB バックアップデータをインポートする) を選択します。

c. [Input S3 Folder] (S3 フォルダに入力) テキストボックスに、エクスポートファイルがある Amazon S3 の URI を入力します。例: `s3://mybucket/exports`

   この URI の形式は `s3://bucketname/folder` で、次のような構成になっています。

   - `bucketname` は Amazon S3 バケットの名前です。
   - `folder` はエクスポートファイルを含むフォルダーの名前です。

   インポートジョブでは、指定した Amazon S3 の場所にファイルが見つかることが想定されます。このファイルの内部形式については、「AWS Data Pipeline デベロッパーガイド」の「データエクスポートファイルの確認」を参照してください。

d. [Target DynamoDB table name] (ターゲット DynamoDB テーブル名) フィールドに、データをインポートする DynamoDB テーブルの名前を入力します。

e. [S3 location for logs] (S3 ログの場所) テキストボックスに、インポート用のログファイルが書き込まれる Amazon S3 の URI を入力します。例: `s3://mybucket/logs/`

   [S3 ログフォルダ] の URI 形式は [出力 S3 フォルダ] と同じです。URI はフォルダに解決される必要があります。ログファイルを S3 バケットの最上位に書き込むことはできません。

f. キー `dynamodbdatapipeline` と値 `true` を含むタグを追加します。

5. すべての設定が正しいことを確認したら、[アクティブ化] をクリックします。

これで、パイプラインが作成されます。このプロセスが完了するまでに数分かかることがあります。インポートジョブはパイプラインの作成直後に開始されます。

トラブルシューティング

このセクションでは、DynamoDB のエクスポートについていくつかの基本的な障害モードとトラブルシューティングを取り上げます。

エクスポートまたはインポート時にエラーが発生した場合、AWS Data Pipeline コンソールのパイプラインステータスは「`ERROR`」として表示されます。この場合は、エラーの発生したパイプラインの名前をクリックして、その詳細ページに移動します。これにより、パイプラインのすべてのステップ

の詳細と、各ステップのステータスが表示されます。特に、表示される実行スタックトレースを確認します。

最後に、Amazon S3 バケットに移動し、そこに書き込まれたすべてのエクスポートまたはインポートログファイルを探します。

次に示しているのは、パイプラインのエラーの原因として考えられるいくつかの一般的な問題とその対処方法です。パイプラインを診断するには、表示されたエラーと次に示している問題を比較します。

- インポートの場合は、インポート先テーブルがすでに存在すること、インポート先テーブルにソーステーブルと同じキースキーマがあることを確認します。これらの条件が満たされていない場合、インポートは失敗します。
- パイプラインに `dynamodbdatapipeline` タグがあることを確認してください。このタグがない場合、Amazon EMR API コールは成功しません。
- 指定した Amazon S3 バケットが作成されていること、そのバケットに対する読み込みと書き込みのアクセス許可があることを確認します。
- パイプラインがその実行タイムアウトを超えている可能性があります (このパラメータはパイプラインの作成時に設定しています。) たとえば、1 時間の実行タイムアウトを設定している場合がありますが、エクスポートジョブでこれより長い時間が必要だった可能性があります。パイプラインを削除して作成し直してみてください。ただし実行タイムアウト間隔は前回よりも長くします。
- エクスポートが実行された元のバケットでない Amazon S3 バケット (エクスポートのコピーを含む) から復元する場合は、マニフェストファイルを更新します。
- エクスポートまたはインポートを実行するための正しいアクセス許可がない可能性があります。詳細については、「データをエクスポートおよびインポートデータするための前提条件」を参照してください。
- AWS アカウントのリソースクォータ (Amazon EC2 インスタンスの最大数や AWS Data Pipeline パイプラインの最大数など) に達した可能性があります。これらのクオータの引き上げリクエストなどの詳しい情報については、「AWS 全般のリファレンス」の「AWS のサービスクォータ」を参照してください。

Note

パイプラインのトラブルシューティングの詳細については、AWS Data Pipeline デベロッパーガイドの「トラブルシューティング」を参照してください。

AWS Data Pipeline と DynamoDB 用の定義済みテンプレート

AWS Data Pipeline のしくみをより深く理解するには、AWS Data Pipeline デベロッパーガイド を参照することをお勧めします。このガイドには、パイプラインを作成して操作する手順について説明したチュートリアルが含まれています。これらのチュートリアルは、独自のパイプラインの作成を開始するときに参考になります。また、AWS Data Pipeline のチュートリアルを参照することをお勧めします。このチュートリアルでは、要件に合わせてカスタマイズ可能なインポートおよびエクスポートパイプラインを作成するステップについて説明しています。「AWS Data Pipeline デベロッパーガイド」の「チュートリアル: AWS Data Pipeline を使用した Amazon DynamoDB のインポートとエクスポート」を参照してください。

AWS Data Pipeline には、パイプラインを作成するためのテンプレートがいくつか用意されています。次のテンプレートは DynamoDB に関連しています。

DynamoDB と Amazon S3 間でデータをエクスポートする

Note

DynamoDB コンソールは、独自の Amazon S3 へのエクスポートフローをサポートするようになりましたが、AWS Data Pipeline インポートフローとの互換性はありません。詳細については、「Amazon S3 への DynamoDB データのエクスポート: 仕組み」およびブログ記事「Export Amazon DynamoDB Table Data to Your Data Lake in Amazon S3, No Code Writing Required」(Amazon DynamoDB のテーブルデータを Amazon S3 のデータレイクにエクスポート、コード作成不要) を参照してください。

AWS Data Pipeline コンソールには、DynamoDB と Amazon S3 間でデータをエクスポートするための定義済みテンプレートが 2 つ用意されています。これらのテンプレートの詳細については、AWS Data Pipeline デベロッパーガイドの次のセクションを参照してください。

- DynamoDB テーブルを S3 にエクスポートする
- Amazon S3 を DynamoDB にエクスポートする

Titan 用 Amazon DynamoDB ストレージバックエンド

Titan 用 Amazon DynamoDB ストレージバックエンドプロジェクトは、GitHub で利用可能な JanusGraph 用 Amazon DynamoDB ストレージバックエンドに置き換えられました。

JanusGraph 用 Amazon DynamoDB ストレージバックエンドに関する最新の手順については、README.md ファイルを参照してください。

DynamoDB の予約語

次のキーワードは DynamoDB によって予約されています。これらの単語を式の属性名として使用しないでください。このリストでは大文字と小文字が区別されません。

DynamoDB の予約語と競合する属性名を含む式を記述する必要が生じた場合は、予約語の代わりに使用する式属性名を定義できます。詳細については、「DynamoDB の式の属性名」を参照してください。

```
ABORT
ABSOLUTE
ACTION
ADD
AFTER
AGENT
AGGREGATE
ALL
ALLOCATE
ALTER
ANALYZE
AND
ANY
ARCHIVE
ARE
ARRAY
AS
ASC
ASCII
ASENSITIVE
ASSERTION
ASYMMETRIC
AT
ATOMIC
ATTACH
ATTRIBUTE
AUTH
AUTHORIZATION
AUTHORIZE
```

```
AUTO
AVG
BACK
BACKUP
BASE
BATCH
BEFORE
BEGIN
BETWEEN
BIGINT
BINARY
BIT
BLOB
BLOCK
BOOLEAN
BOTH
BREADTH
BUCKET
BULK
BY
BYTE
CALL
CALLED
CALLING
CAPACITY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHECK
CLASS
CLOB
CLOSE
CLUSTER
CLUSTERED
CLUSTERING
CLUSTERS
COALESCE
COLLATE
COLLATION
COLLECTION
```

```
COLUMN
COLUMNS
COMBINE
COMMENT
COMMIT
COMPACT
COMPILE
COMPRESS
CONDITION
CONFLICT
CONNECT
CONNECTION
CONSISTENCY
CONSISTENT
CONSTRAINT
CONSTRAINTS
CONSTRUCTOR
CONSUMED
CONTINUE
CONVERT
COPY
CORRESPONDING
COUNT
COUNTER
CREATE
CROSS
CUBE
CURRENT
CURSOR
CYCLE
DATA
DATABASE
DATE
DATETIME
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DEFINE
DEFINED
```

```
DEFINITION
DELETE
DELIMITED
DEPTH
DEREF
DESC
DESCRIBE
DESCRIPTOR
DETACH
DETERMINISTIC
DIAGNOSTICS
DIRECTORIES
DISABLE
DISCONNECT
DISTINCT
DISTRIBUTE
DO
DOMAIN
DOUBLE
DROP
DUMP
DURATION
DYNAMIC
EACH
ELEMENT
ELSE
ELSEIF
EMPTY
ENABLE
END
EQUAL
EQUALS
ERROR
ESCAPE
ESCAPED
EVAL
EVALUATE
EXCEEDED
EXCEPT
EXCEPTION
EXCEPTIONS
EXCLUSIVE
EXEC
EXECUTE
```

```
EXISTS
EXIT
EXPLAIN
EXPLODE
EXPORT
EXPRESSION
EXTENDED
EXTERNAL
EXTRACT
FAIL
FALSE
FAMILY
FETCH
FIELDS
FILE
FILTER
FILTERING
FINAL
FINISH
FIRST
FIXED
FLATTERN
FLOAT
FOR
FORCE
FOREIGN
FORMAT
FORWARD
FOUND
FREE
FROM
FULL
FUNCTION
FUNCTIONS
GENERAL
GENERATE
GET
GLOB
GLOBAL
GO
GOTO
GRANT
GREATER
GROUP
```

```
GROUPING
HANDLER
HASH
HAVE
HAVING
HEAP
HIDDEN
HOLD
HOUR
IDENTIFIED
IDENTITY
IF
IGNORE
IMMEDIATE
IMPORT
IN
INCLUDING
INCLUSIVE
INCREMENT
INCREMENTAL
INDEX
INDEXED
INDEXES
INDICATOR
INFINITE
INITIALLY
INLINE
INNER
INNTER
INOUT
INPUT
INSENSITIVE
INSERT
INSTEAD
INT
INTEGER
INTERSECT
INTERVAL
INTO
INVALIDATE
IS
ISOLATION
ITEM
ITEMS
```

```
ITERATE
JOIN
KEY
KEYS
LAG
LANGUAGE
LARGE
LAST
LATERAL
LEAD
LEADING
LEAVE
LEFT
LENGTH
LESS
LEVEL
LIKE
LIMIT
LIMITED
LINES
LIST
LOAD
LOCAL
LOCALTIME
LOCALTIMESTAMP
LOCATION
LOCATOR
LOCK
LOCKS
LOG
LOGED
LONG
LOOP
LOWER
MAP
MATCH
MATERIALIZED
MAX
MAXLEN
MEMBER
MERGE
METHOD
METRICS
MIN
```

```
MINUS
MINUTE
MISSING
MOD
MODE
MODIFIES
MODIFY
MODULE
MONTH
MULTI
MULTISET
NAME
NAMES
NATIONAL
NATURAL
NCHAR
NCLOB
NEW
NEXT
NO
NONE
NOT
NULL
NULLIF
NUMBER
NUMERIC
OBJECT
OF
OFFLINE
OFFSET
OLD
ON
ONLINE
ONLY
OPAQUE
OPEN
OPERATOR
OPTION
OR
ORDER
ORDINALITY
OTHER
OTHERS
OUT
```

```
OUTER
OUTPUT
OVER
OVERLAPS
OVERRIDE
OWNER
PAD
PARALLEL
PARAMETER
PARAMETERS
PARTIAL
PARTITION
PARTITIONED
PARTITIONS
PATH
PERCENT
PERCENTILE
PERMISSION
PERMISSIONS
PIPE
PIPELINED
PLAN
POOL
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVATE
PRIVILEGES
PROCEDURE
PROCESSED
PROJECT
PROJECTION
PROPERTY
PROVISIONING
PUBLIC
PUT
QUERY
QUIT
QUORUM
RAISE
RANDOM
```

```
RANGE
RANK
RAW
READ
READS
REAL
REBUILD
RECORD
RECURSIVE
REDUCE
REF
REFERENCE
REFERENCES
REFERENCING
REGEXP
REGION
REINDEX
RELATIVE
RELEASE
REMAINDER
RENAME
REPEAT
REPLACE
REQUEST
RESET
RESIGNAL
RESOURCE
RESPONSE
RESTORE
RESTRICT
RESULT
RETURN
RETURNING
RETURNS
REVERSE
REVOKE
RIGHT
ROLE
ROLES
ROLLBACK
ROLLUP
ROUTINE
ROW
ROWS
```

```
RULE
RULES
SAMPLE
SATISFIES
SAVE
SAVEPOINT
SCAN
SCHEMA
SCOPE
SCROLL
SEARCH
SECOND
SECTION
SEGMENT
SEGMENTS
SELECT
SELF
SEMI
SENSITIVE
SEPARATE
SEQUENCE
SERIALIZABLE
SESSION
SET
SETS
SHARD
SHARE
SHARED
SHORT
SHOW
SIGNAL
SIMILAR
SIZE
SKEWED
SMALLINT
SNAPSHOT
SOME
SOURCE
SPACE
SPACES
SPARSE
SPECIFIC
SPECIFICTYPE
SPLIT
```

```
SQL
SQLCODE
SQLERROR
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATIC
STATUS
STORAGE
STORE
STORED
STREAM
STRING
STRUCT
STYLE
SUB
SUBMULTISET
SUBPARTITION
SUBSTRING
SUBTYPE
SUM
SUPER
SYMMETRIC
SYNONYM
SYSTEM
TABLE
TABLESAMPLE
TEMP
TEMPORARY
TERMINATED
TEXT
THAN
THEN
THROUGHPUT
TIME
TIMESTAMP
TIMEZONE
TINYINT
TO
TOKEN
TOTAL
TOUCH
```

```
TRAILING
TRANSACTION
TRANSFORM
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
TRUNCATE
TTL
TUPLE
TYPE
UNDER
UNDO
UNION
UNIQUE
UNIT
UNKNOWN
UNLOGGED
UNNEST
UNPROCESSED
UNSIGNED
UNTIL
UPDATE
UPPER
URL
USAGE
USE
USER
USERS
USING
UUID
VACUUM
VALUE
VALUED
VALUES
VARCHAR
VARIABLE
VARIANCE
VARINT
VARYING
VIEW
VIEWS
```

```
VIRTUAL
VOID
WAIT
WHEN
WHENEVER
WHERE
WHILE
WINDOW
WITH
WITHIN
WITHOUT
WORK
WRAPPED
WRITE
YEAR
ZONE
```

レガシー条件パラメータ

このセクションでは、DynamoDB のレガシーの条件パラメータと表現パラメータを比較します。

⚠ Important

可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。さらに、DynamoDB では、1 つの呼び出しでレガシー条件パラメータと式パラメータを混在させることはできません。例えば、`AttributesToGet` と `ConditionExpression` で `Query` オペレーションを呼び出すとエラーになります。

次の表は、これらのレガシーパラメータを引き続きサポートする DynamoDB API オペレーション、および代替として使用が推奨される式パラメータを示しています。このテーブルは、式パラメータを使用するようにアプリケーションを更新することを検討している場合に役立ちます。

この API オペレーションを使用すると...	これらのレガシーパラメータでは...	代わりにこの式パラメータを使用
`BatchGetItem`	`AttributesToGet`	`ProjectionExpression`
`DeleteItem`	`Expected`	`ConditionExpression`

この API オペレーションを使用すると...	これらのレガシーパラメータでは...	代わりにこの式パラメータを使用
`GetItem`	`AttributesToGet`	`ProjectionExpression`
`PutItem`	`Expected`	`ConditionExpression`
`Query`	`AttributesToGet`	`ProjectionExpression`
	`KeyConditions`	`KeyConditionExpression`
	`QueryFilter`	`FilterExpression`
`Scan`	`AttributesToGet`	`ProjectionExpression`
	`ScanFilter`	`FilterExpression`
`UpdateItem`	`AttributeUpdates`	`UpdateExpression`
	`Expected`	`ConditionExpression`

以下のセクションでは、レガシー条件パラメータについて詳しく説明します。

トピック

- AttributesToGet (レガシー)
- AttributeUpdates (レガシー)
- ConditionalOperator (レガシー)
- Expected (レガシー)
- KeyConditions (レガシー)
- QueryFilter (レガシー)
- ScanFilter (レガシー)
- レガシーパラメータを使用した条件の記述

AttributesToGet (レガシー)

> **Note**
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。このパラメータに代わる新しいパラメータの具体的な情報については、「代わりに ProjectionExpression を使用します。」を参照してください。

レガシー条件パラメータ `AttributesToGet` は、DynamoDB から取得する 1 つ以上の属性の配列です。属性名が指定されていない場合、すべての属性が返されます。リクエストした属性が見つからない場合、その属性は結果に表示されません。

`AttributesToGet` では、リストまたはマップ型の属性を取得できますが、リストまたはマップ内の個々の要素を取得することはできません。

`AttributesToGet` は、プロビジョニングされたスループットの消費には影響しません。DynamoDB は、アプリケーションに返されるデータ量ではなく、項目のサイズに基づいて、消費される読み込み容量ユニットの数を決定します。

代わりに ProjectionExpression を使用 - 例

Music テーブルから項目を取得する際、一部の属性のみが返されるようにする場合を考えてみます。次の AWS CLI の例に示すように、`AttributesToGet` パラメータを含む `GetItem` リクエストを使用できます。

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

代わりに `ProjectionExpression` を使用できます。

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (レガシー)

> Note
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。このパラメータに代わる新しいパラメータの具体的な情報については、「代わりに UpdateExpression を使用します。」を参照してください。

UpdateItem オペレーションでは、レガシー条件パラメータ AttributeUpdates に、変更する属性の名前、それぞれに対して実行するアクション、およびそれぞれの新しい値が含まれます。そのテーブル上のインデックスのインデックスキー属性である属性を更新する場合、属性の型は、テーブルの説明の AttributesDefinition で定義されたインデックスキーの型に一致する必要があります。UpdateItem を使用して非キー属性を更新できます。

属性値を null にすることはできません。文字列型およびバイナリ型属性の長さは、ゼロより大きくなければなりません。セット型の属性を空にすることはできません。空の値を持つリクエストは、ValidationException 例外で拒否されます。

各 AttributeUpdates 要素は、変更する属性名と次の属性で構成されます。

- Value - この属性の新しい値 (該当する場合)。
- Action - 更新の実行方法を指定する値。このアクションは、データ型が数値またはセットである既存の属性に対してのみ有効です。ADD は他のデータ型に使用しないでください。

  指定したプライマリキーを持つ項目がテーブル内に見つかった場合、次の値によって以下のアクションが実行されます。

  - PUT - 指定した属性を項目に追加します。属性がすでに存在する場合は、新しい値で置き換えられます。
  - DELETE - DELETE の値が指定されていない場合、属性とその値を削除します。指定した値のデータ型は、既存の値のデータ型と一致する必要があります。

値のセットが指定されている場合、それらの値が古いセットから削除されます。例えば、属性値が [a,b,c] のセットの場合に DELETE アクションで [a,c] を指定すると、最終的な属性値は [b] になります。空のセットを指定するとエラーになります。

- ADD - 属性が存在しない場合、指定された値を項目に追加します。属性が存在する場合、ADD の動作は属性のデータ型によって決まります。
  - 既存の属性が数値で、Value も数値である場合、Value は既存の属性に数学的に追加されます。Value が負の数値である場合は、既存の属性から削除されます。

    > Note
    >
    > ADD を使用して、更新の前に存在しない項目の数値をインクリメントまたはデクリメントした場合、DynamoDB は初期値として 0 を使用します。
    > 同様に、更新の前に存在しない属性値をインクリメントまたはデクリメントするために既存の項目に対して ADD を使用した場合、DynamoDB は初期値として 0 を使用します。例えば、itemcount という属性を持たない項目を更新する際に ADD を使用して、この属性に 3 を追加する場合を考えています。この場合、DynamoDB によって itemcount 属性が作成され、その初期値が 0 に設定されます。最後に 3 が追加されます。結果は、3 の値を含む新しい itemcount 属性になります。

  - 既存のデータ型がセットで、Value もセットである場合、Value が既存のセットに追加されます。例えば、属性値が [1,2] のセットで、ADD アクションで [3] が指定されている場合、最終的な属性値は [1,2,3] になります。セット属性に対して ADD アクションが指定されていて、指定した属性の型が既存のセット型に一致しない場合、エラーが発生します。

    両方のセットは同じプリミティブデータ型を持つ必要があります。例えば、既存のデータ型が文字列のセットである場合、Value も文字列のセットである必要があります。

指定されたキーを持つ項目がテーブルに見つからない場合、次の値は次のアクションを実行します。

- PUT - DynamoDB で、指定されたプライマリキーで新しい項目が作成され、属性が追加されます。
- DELETE - 存在しない項目から属性を削除することはできないので何の処理も行われません。オペレーションは成功しますが、DynamoDB は新しい項目を作成しません。
- ADD - DynamoDB で、指定されたプライマリキーと属性値の番号 (または数字のセット) を持つ項目を作成します。使用できるデータ型は、数値型と数値セット型のみです。

インデックスキーの一部である属性を指定する場合、それらの属性のデータ型は、テーブルの属性定義内のスキーマのデータ型と一致する必要があります。

代わりに UpdateExpression を使用 - 例

Music テーブル内の項目を変更する場合を考えてみます。次の AWS CLI の例に示すように、`AttributeUpdates` パラメータを含む `UpdateItem` リクエストを使用できます。

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --attribute-updates '{
        "Genre": {
            "Action": "PUT",
            "Value": {"S":"Rock"}
        }
    }'
```

代わりに `UpdateExpression` を使用できます。

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --update-expression 'SET Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

属性の更新の詳細については、「DynamoDB テーブルで項目を更新する」を参照してください。

ConditionalOperator (レガシー)

> Note
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。

レガシー条件パラメータ `ConditionalOperator` は、`Expected`、`QueryFilter`、または `ScanFilter` マップ内の条件に適用するために使用される論理演算子です。

- AND - すべての条件が true に評価された場合、マップ全体が true に評価されます。
- OR - 条件の少なくとも 1 つが true に評価された場合、マップ全体が true に評価されます。

`ConditionalOperator` を省略した場合、`AND` がデフォルトです。

オペレーションは、マップ全体が true と評価された場合にのみ成功します。

> Note
>
> このパラメータは、リストおよびマップ型の属性をサポートしません。

Expected (レガシー)

> Note
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。このパラメータに代わる新しいパラメータの具体的な情報については、「代わりに ConditionExpression を使用します。」を参照してください。

レガシー条件パラメータ `Expected` は `UpdateItem` オペレーションの条件付きブロックです。`Expected` は属性/条件ペアのマップです。マップの各要素は、属性名、比較演算子、および 1 つ以上の値で構成されます。DynamoDB は、比較演算子を使用して、指定した値と属性を比較します。各 `Expected` 要素に対して、評価の結果は true または false のいずれかです。

Expected マップで複数の要素を指定すると、デフォルトでは、すべての条件が true に評価される必要があります。つまり、すべての条件が一緒に AND 処理されます。(ConditionalOperator パラメータを使用して条件を OR に設定できます。その場合、すべての条件ではなく、少なくとも 1 つの条件が true に評価される必要があります)。

Expected マップが true に評価された場合、オペレーションは成功します。それ以外の場合、オペレーションは失敗します。

Expected には以下の要素が含まれます。

- AttributeValueList - 指定された属性に対して評価する 1 つ以上の値。リストの値の数は、使用される ComparisonOperator エンジンによって異なります。

  数値型の場合、値の比較は数値です。

  より大きい、等しい、またはより小さいの文字列値の比較は、UTF-8 バイナリエンコーディングの Unicode に基づきます。例えば、a は A より大きく、a は B より大きいと評価されます。

  バイナリの場合、DynamoDB がバイナリ値を比較する際、バイナリデータの各バイトは符号なしとして扱われます。

- ComparisonOperator - AttributeValueList の属性を評価するためのコンパレータ。比較を実行する際、DynamoDB は強力な整合性のある読み込みを使用します。

  次の比較演算子がサポートされています。

  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN

  各比較演算子の説明は、以下のとおりです。

  - EQ: 等しい。EQ は、リストやマップなど、すべてのデータ型でサポートされます。

    AttributeValueList には、文字列、数値、バイナリ、文字列セット、数値セット、またはバイナリセットの方の 1 つの AttributeValue 要素のみを含めることができます。項目に含まれる AttributeValue 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と等しくありません。

  - NE: 等しくない。NE は、リストやマップを始めとするすべてのデータ型でサポートされています。

AttributeValueList には、文字列、数値、バイナリ、文字列セット、数値セット、またはバイナリセットの型の 1 つの AttributeValue のみを含めることができます。項目に含まれる AttributeValue の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と等しくありません。

- LE: より小さい、または等しい。

  AttributeValueList には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの AttributeValue 要素のみ含めることができます。項目に含まれる AttributeValue 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。

- LT: より小さい。

  AttributeValueList には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの AttributeValue のみ含めることができます。項目に含まれる AttributeValue 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。

- GE: より大きい、または等しい。

  AttributeValueList には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの AttributeValue 要素のみ含めることができます。項目に含まれる AttributeValue 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。

- GT: より大きい。

  AttributeValueList には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの AttributeValue 要素のみ含めることができます。項目に含まれる AttributeValue 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。

- NOT_NULL: 属性が存在します。NOT_NULL は、リストやマップを始めとするすべてのデータ型でサポートされています。

> **Note**
>
> この演算子は、データ型ではなく、属性が存在することをテストします。NOT_NULL を使用して属性のデータ型「a」を評価する場合、結果はブール値の true です。この結果理由は、属性「a」が存在するからです。そのデータ型は NOT_NULL 比較演算子に関係ありません。

- NULL: 属性は存在しません。NULL は、リストやマップを始めとするすべてのデータ型でサポートされています。

  > **Note**
  >
  > この演算子は、データ型ではなく、属性が存在しないことをテストします。a を使用して属性のデータ型「NULL」を評価する場合、結果はブール値の false です。この理由は、属性「a」が存在するからです。そのデータ型は NULL 比較演算子に関係ありません。

- CONTAINS: サブシーケンス、またはセット内の値をチェックします。

  AttributeValueList には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの AttributeValue 要素のみ含めることができます。比較のターゲット属性が文字列型である場合、演算子は部分文字列の一致をチェックします。比較のターゲット属性が型バイナリの場合、演算子は入力に一致するターゲットのサブシーケンスを検索します。比較のターゲット属性がセット (「SS」、「NS」、または「BS」) の場合、セットのいずれかの要素との完全一致が見つかったときに true に評価されます。

  CONTAINS はリストでサポートされています。「a CONTAINS b」を評価するとき、「a」はリストである可能性があります。しかし、「b」が、セット、マップ、およびリストになることはありません。

- NOT_CONTAINS: サブシーケンスの欠如、またはセット内の値の欠如をチェックします。

  AttributeValueList には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの AttributeValue 要素のみ含めることができます。比較のターゲット属性が文字列の場合、演算子は部分文字列の一致の欠如をチェックします。比較のターゲット属性がバイナリである場合、演算子は、入力に一致するターゲットのサブシーケンスの欠如をチェックします。比較のターゲット属性がセット (「SS」、「NS」、または「BS」) のとき、セットのいずれか要素の完全一致が見つからdoes not場合、true に評価されます。

NOT_CONTAINS はリストでサポートされています。「a NOT CONTAINS b」を評価するとき、「a」はリストである可能性があります。しかし、「b」は、セット、マップ、およびリストになることはありません。

- BEGINS_WITH: プレフィックスを確認します。

  AttributeValueList には、文字列またはバイナリ型の 1 つの AttributeValue のみを含めることができます (数値やセットではありません)。比較のターゲット属性は、文字列型またはバイナリ型である必要があります (数値型またはセット型ではありません)。
- IN: 2 つのセット内で一致する要素をチェックします。

  AttributeValueList には、文字列、数値、またはバイナリ型の 1 つ以上の AttributeValue 要素を含むことができます (セット型ではありません)。これらの属性は、項目の既存のセット型属性と比較されます。入力セットのいずれかの要素が項目の属性に存在する場合、式は true に評価されます。
- BETWEEN: 最初の値よりも大きいか等しく、2 番目の値よりも小さいか等しくなります。

  AttributeValueList には、文字列、数値、またはバイナリ (セットではありません) のいずれかの同じ型の 2 つの AttributeValue 要素を含む必要があります。ターゲット属性は、ターゲット値が最初の要素より大きいか等しく、2番目の要素より小さいか等しい場合に一致します。項目に含まれる AttributeValue 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と比較されません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。

次のパラメータは、AttributeValueList および ComparisonOperator の代わりに使用できます。

- Value - DynamoDB が属性と比較する値。
- Exists - 条件付きオペレーションを実行する前に DynamoDB が値を評価するブール値。
  - Exists が true の場合、DynamoDB は、その属性値がテーブル内にすでに存在するかどうかを確認します。見つかった場合、条件は true に評価されます。それ以外の場合、条件は false に評価されます。
  - Exists が false の場合、DynamoDB は属性値がテーブル内に存在notことが予期されます。実際に値が存在しない場合、仮定は有効であり、条件は true に評価されます。存在しないという仮定にもかかわらず値が見つかった場合、条件は false に評価されます。

  Exists のデフォルト値は true です。

Value パラメータと Exists パラメータは、AttributeValueList および ComparisonOperator と互換性がありません。両方のパラメータセットを同時に使用すると、DynamoDB は ValidationException 例外を返します。

> Note
>
> このパラメータは、リストおよびマップ型の属性をサポートしません。

代わりに ConditionExpression を使用 - 例

特定の条件が true の場合のみに Music テーブル内の項目を変更する場合を考えてみます。次の AWS CLI の例に示すように、Expected パラメータを含む UpdateItem リクエストを使用できます。

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --attribute-updates '{
        "Price": {
            "Action": "PUT",
            "Value": {"N":"1.98"}
        }
    }' \
    --expected '{
        "Price": {
            "ComparisonOperator": "LE",
            "AttributeValueList": [ {"N":"2.00"} ]
        }
    }'
```

代わりに ConditionExpression を使用できます。

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --update-expression 'SET Price = :p1' \
    --condition-expression 'Price <= :p2' \
    --expression-attribute-values '{
        ":p1": {"N":"1.98"},
        ":p2": {"N":"2.00"}
    }'
```

KeyConditions (レガシー)

> Note
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。このパラメータに代わる新しいパラメータの具体的な情報については、「代わりに KeyConditionExpression を使用します。」を参照してください。

レガシー条件パラメータ KeyConditions には、Query オペレーションの選択基準が含まれています。テーブルに対するクエリでは、テーブルのプライマリキー属性に対してのみ条件を設定できます。EQ 条件としてパーティションキーの名前と値を指定する必要があります。ソートキーを参照する 2 番目の条件を指定できます (オプション)。

> Note
>
> ソートキー条件を指定しない場合、パーティションキーに一致するすべての項目が取得されます。FilterExpression または QueryFilter が存在する場合、これは項目が取得された後に適用されます。

インデックスに対するクエリでは、インデックスキー属性に対してのみ条件を設定できます。EQ 条件としてインデックスパーティションキーの名前と値を指定する必要があります。インデックスソートキーを参照する 2 番目の条件を指定できます (オプション)。

各 KeyConditions 要素は、比較する属性名と次の属性で構成されます。

- AttributeValueList - 指定された属性に対して評価する 1 つ以上の値。リストの値の数は、使用される ComparisonOperator エンジンによって異なります。

  数値型の場合、値の比較は数値です。

より大きい、等しい、またはより小さいの文字列値の比較は、UTF-8 バイナリエンコーディングの Unicode に基づきます。例えば、a は A より大きく、a は B より大きいと評価されます。

バイナリの場合、DynamoDB がバイナリ値を比較する際、バイナリデータの各バイトは符号なしとして扱われます。

- `ComparisonOperator` - 属性を評価するためのコンパレータ。例えば、等しい、より大きい、より小さいなどです。

  `KeyConditions` の場合、次の比較演算子がサポートされています。

  `EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN`

  これらの比較演算子の説明を次に示します。

  - `EQ`: 等しい。

    `AttributeValueList` には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの `AttributeValue` のみ含めることができます。リクエストで指定されているもの以外の型の `AttributeValue` 要素が項目に含まれる場合、値は一致しません。例えば、`{"S":"6"}` は `{"N":"6"}` と等しくありません。また、`{"N":"6"}` は `{"NS":["6", "2", "1"]}` と等しくありません。

  - `LE`: より小さい、または等しい。

    `AttributeValueList` には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの `AttributeValue` 要素のみ含めることができます。項目に含まれる `AttributeValue` 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、`{"S":"6"}` は `{"N":"6"}` と等しくありません。また、`{"N":"6"}` は `{"NS":["6", "2", "1"]}` と比較されません。

  - `LT`: より小さい。

    `AttributeValueList` には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの `AttributeValue` のみ含めることができます。項目に含まれる `AttributeValue` 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、`{"S":"6"}` は `{"N":"6"}` と等しくありません。また、`{"N":"6"}` は `{"NS":["6", "2", "1"]}` と比較されません。

  - `GE`: より大きい、または等しい。

`AttributeValueList` には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの `AttributeValue` 要素のみ含めることができます。項目に含まれる `AttributeValue` 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、`{"S":"6"}` は `{"N":"6"}` と等しくありません。また、`{"N":"6"}` は `{"NS":["6", "2", "1"]}` と比較されません。

- `GT`: より大きい。

  `AttributeValueList` には、文字列、数値、またはバイナリの型 (セット型ではありません) の 1 つの `AttributeValue` 要素のみ含めることができます。項目に含まれる `AttributeValue` 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、`{"S":"6"}` は `{"N":"6"}` と等しくありません。また、`{"N":"6"}` は `{"NS":["6", "2", "1"]}` と比較されません。

- `BEGINS_WITH`: プレフィックスを確認します。

  `AttributeValueList` には、文字列またはバイナリ型の 1 つの `AttributeValue` のみを含めることができます (数値やセットではありません)。比較のターゲット属性は、文字列型またはバイナリ型である必要があります (数値型またはセット型ではありません)。

- `BETWEEN`: 最初の値よりも大きいか等しく、2 番目の値よりも小さいか等しくなります。

  `AttributeValueList` には、文字列、数値、またはバイナリ (セットではありません) のいずれかの同じ型の 2 つの `AttributeValue` 要素を含む必要があります。ターゲット属性は、ターゲット値が最初の要素より大きいか等しく、2番目の要素より小さいか等しい場合に一致します。項目に含まれる `AttributeValue` 要素の型がリクエストで指定されている型と異なる場合、値は一致しません。例えば、`{"S":"6"}` は `{"N":"6"}` と比較されません。また、`{"N":"6"}` は `{"NS":["6", "2", "1"]}` と比較されません。

代わりに KeyConditionExpression を使用 - 例

同じパーティションキーを持つ複数の項目を Music テーブルから取得する場合を考えてみます。次の AWS CLI の例に示すように、`Query` パラメータを含む `KeyConditions` リクエストを使用できます。

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist":{
            "ComparisonOperator":"EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        },
        "SongTitle":{
            "ComparisonOperator":"BETWEEN",
            "AttributeValueList": [ {"S": "A"}, {"S": "M"} ]
        }
    }'
```

代わりに KeyConditionExpression を使用できます。

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a AND SongTitle BETWEEN :t1 AND :t2' \
    --expression-attribute-values '{
        ":a": {"S": "No One You Know"},
        ":t1": {"S": "A"},
        ":t2": {"S": "M"}
    }'
```

QueryFilter (レガシー)

> Note
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。このパラメータに代わる新しいパラメータの具体的な情報については、「代わりに FilterExpression を使用します。」を参照してください。

Query オペレーションでは、レガシー条件パラメータ QueryFilter は、項目が読み込まれた後にクエリ結果を評価し、目的の値のみを返す条件です。

このパラメータは、リストおよびマップ型の属性をサポートしません。

> Note
>
> QueryFilter は、項目が読み込まれた後に適用されます。フィルタリングのプロセスでは、追加の読み込み容量ユニットは消費されません。

QueryFilter マップで複数の条件を指定すると、デフォルトでは、すべての条件が true に評価される必要があります。つまり、すべての条件が一緒に AND 処理されます。(ConditionalOperator (レガシー) パラメータを使用して条件を OR に設定できます。その場合、すべての条件ではなく、少なくとも 1 つの条件が true に評価される必要があります)。

QueryFilter ではキー属性を使用できません。パーティションキーまたはソートキーでフィルター条件を定義することはできません。

各 QueryFilter 要素は、比較する属性名と次の属性で構成されます。

- AttributeValueList - 指定された属性に対して評価する 1 つ以上の値。リスト内の値の数は、ComparisonOperator で指定されている演算子により異なります。

  数値型の場合、値の比較は数値です。

  より大きい、等しい、より小さい文字列値の比較は、UTF-8 バイナリエンコーディングに基づいています。例えば、a は A より大きく、a は B より大きいと評価されます。

  バイナリの場合、DynamoDB がバイナリ値を比較する際、バイナリデータの各バイトは符号なしとして扱われます。

  JSON でのデータ型指定については、「DynamoDB 低レベル API」を参照してください。
- ComparisonOperator - 属性を評価するためのコンパレータ。例えば、等しい、より大きい、より小さいなどです。

  次の比較演算子がサポートされています。

  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN

代わりに FilterExpression を使用 - 例

Music テーブルをクエリし、一致する項目に条件を適用する場合を考えてみます。次の AWS CLI の例に示すように、Query パラメータを含む QueryFilter リクエストを使用できます。

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        }
    }' \
    --query-filter '{
        "Price": {
            "ComparisonOperator": "GT",
            "AttributeValueList": [ {"N": "1.00"} ]
        }
    }'
```

代わりに FilterExpression を使用できます。

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a' \
    --filter-expression 'Price > :p' \
    --expression-attribute-values '{
        ":p": {"N":"1.00"},
        ":a": {"S":"No One You Know"}
    }'
```

ScanFilter (レガシー)

> Note
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。このパラメータに代わる新しいパラメータの具体的な情報については、「代わりに FilterExpression を使用します。」を参照してください。

Scan オペレーションでは、レガシー条件パラメータ ScanFilter は、スキャン結果を評価し、目的の値のみを返す条件です。

> Note
>
> このパラメータは、リストおよびマップ型の属性をサポートしません。

複数の条件を指定すると、ScanFilter マップの場合、デフォルトでは、すべての条件が true に評価される必要があります。つまり、すべての条件が一緒に AND 処理されます。(ConditionalOperator

(レガシー) パラメータを使用して条件を OR に設定できます。その場合、すべての条件ではなく、少なくとも 1 つの条件が true に評価される必要があります)。

各 ScanFilter 要素は、比較する属性名と次の属性で構成されます。

- AttributeValueList - 指定された属性に対して評価する 1 つ以上の値。リスト内の値の数は、ComparisonOperator で指定されている演算子により異なります。

  数値型の場合、値の比較は数値です。

  より大きい、等しい、より小さい文字列値の比較は、UTF-8 バイナリエンコーディングに基づいています。例えば、a は A より大きく、a は B より大きいと評価されます。

  バイナリの場合、DynamoDB がバイナリ値を比較する際、バイナリデータの各バイトは符号なしとして扱われます。

  JSON でのデータ型指定については、「DynamoDB 低レベル API」を参照してください。
- ComparisonOperator - 属性を評価するためのコンパレータ。例えば、等しい、より大きい、より小さいなどです。

  次の比較演算子がサポートされています。

  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN

代わりに FilterExpression を使用 - 例

Music テーブルをスキャンし、一致する項目に条件を適用する場合を考えてみます。次の AWS CLI の例に示すように、Scan パラメータを含む ScanFilter リクエストを使用できます。

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

代わりに FilterExpression を使用できます。

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

レガシーパラメータを使用した条件の記述

> Note
>
> 可能な限り、これらのレガシーパラメータの代わりに新しい式パラメータを使用することをお勧めします。詳細については、「DynamoDB での式の使用」を参照してください。

このセクションでは、Expected、QueryFilter、ScanFilter などのレガシーパラメータで使用するための条件を書き込む方法について説明します。

> Note
>
> 新しいアプリケーションでは、式パラメータを使用する必要があります。詳細については、「DynamoDB での式の使用」を参照してください。

シンプルな条件

属性値を使用すると、テーブル属性との比較条件を記述できます。条件は常にtrueまたはfalseに評価され、以下のもので構成されます。

- ComparisonOperator — より大きい、より小さい、等しいなど。
- AttributeValueList (オプション) — 比較対象の属性値。使用する ComparisonOperator に応じて、AttributeValueList には、1 つ、2 つ、またはそれ以上の値が含まれることがあります。まったく存在しないこともあります。

このセクションでは、さまざまな比較演算子と、条件での使用方法の例について説明します。

属性値を含まない比較演算子

- NOT_NULL - 属性が存在する場合は true です。

- NULL - 属性が存在しない場合は true です。

これらの演算子を使用して、属性が存在するか、存在しないかをチェックします。比較する値がないので、AttributeValueList は指定しません。

例

Dimensions 属性が存在する場合、次の式は true と評価されます。

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

属性値を 1 つ含む比較演算子

- EQ - 属性が値と等しい場合、true です。

  AttributeValueList には、文字列、数値、バイナリ、文字列セット、数値セット、またはバイナリセットの型の 1 つの値のみを含めることができます。リクエストで指定されているもの以外の型の値が項目に含まれる場合、値は一致しません。例えば、文字列 "3" は数値 3 と等しくありません。また、数値 3 は数値セット [3, 2, 1] と等しくありません。
- NE - 属性が値と等しくない場合、true です。

  AttributeValueList には、文字列、数値、バイナリ、文字列セット、数値セット、またはバイナリセットの型の 1 つの値のみを含めることができます。リクエストで指定されているもの以外の型の値が項目に含まれる場合、値は一致しません。
- LE - 属性が値より小さいか、等しい場合、true です。

  AttributeValueList は、文字列、数値、またはバイナリのうち 1 つの値のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。
- LT - 属性が値より小さい場合、true です。

  AttributeValueList は、文字列、数値、またはバイナリのうち 1 つの値のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の値が項目に含まれる場合、値は一致しません。
- GE - 属性が値より大きいか、等しい場合、true です。

AttributeValueList は、文字列、数値、またはバイナリのうち 1 つの値のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の値が項目に含まれる場合、値は一致しません。

- GT - 属性が値より大きい場合、true です。

  AttributeValueList は、文字列、数値、またはバイナリのうち 1 つの値のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の値が項目に含まれる場合、値は一致しません。

- CONTAINS - 値がセット内に存在する場合、またはある値に別の値が含まれている場合、true です。

  AttributeValueList は、文字列、数値、またはバイナリのうち 1 つの値のみを含むことができます (セットではありません)。比較のターゲット属性が文字列の場合、演算子は部分文字列の一致をチェックします。比較のターゲット属性がバイナリの場合、演算子は入力に一致するターゲットのサブシーケンスを検索します。比較のターゲット属性がセットの場合、セットのいずれかの要素との完全一致が見つかると、演算子は true と評価されます。

- NOT_CONTAINS - 値がセット内に存在しない場合、またはある値に別の値が含まれている場合、true です。

  AttributeValueList は、文字列、数値、またはバイナリのうち 1 つの値のみを含むことができます (セットではありません)。比較のターゲット属性が文字列の場合、演算子は部分文字列の一致の欠如をチェックします。比較のターゲット属性がバイナリである場合、演算子は、入力に一致するターゲットのサブシーケンスの欠如をチェックします。比較のターゲット属性がセットの場合、セットのいずれかの要素との完全一致が見つからないと、演算子は true と評価されます。

- BEGINS_WITH - 属性の最初の数文字が指定された値と一致する場合、true です。この演算子は数値の比較に使用しないでください。

  AttributeValueList には、文字列またはバイナリ型の 1 つの値のみを含めることができます (数値やセットではありません)。比較のターゲット属性は、(数値またはセットではなく) 文字列またはバイナリである必要があります。

これらの演算子を使用して、属性と値を比較します。1 つの値で構成される AttributeValueList を指定する必要があります。ほとんどの演算子では、この値はスカラーである必要があります。ただし、EQ 演算子と NE 演算子はセットもサポートします。

例

以下の条件が満たされる場合、次の式は true と評価されます。

- 商品の価格が 100 より大きい。

  ```
  ...
      "Price": {
          ComparisonOperator: "GT",
          AttributeValueList: [ {"N":"100"} ]
      }
  ...
  ```

- 商品カテゴリが「Bo」で始まる。

  ```
  ...
      "ProductCategory": {
          ComparisonOperator: "BEGINS_WITH",
          AttributeValueList: [ {"S":"Bo"} ]
      }
  ...
  ```

- 製品には、赤、緑、または黒がある。

  ```
  ...
      "Color": {
          ComparisonOperator: "EQ",
          AttributeValueList: [
              [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
          ]
      }
  ...
  ```

  > Note
  >
  > セットデータ型を比較する場合、要素の順序は関係ありません。DynamoDB は、リクエストで指定した順序に関係なく、同じ値のセットを持つ項目のみを返します。

属性値を 2 つ含む比較演算子

- BETWEEN - 値が下限と上限の間にある場合 (エンドポイントを含む) は true です。

AttributeValueList には、文字列、数値、またはバイナリ (セットではありません) のいずれかの同じ型の 2 つの要素を含む必要があります。ターゲット属性は、ターゲット値が最初の要素より大きいか等しく、2番目の要素より小さいか等しい場合に一致します。リクエストで指定されているもの以外の型の値が項目に含まれる場合、値は一致しません。

この演算子を使用して、属性値が範囲内にあるかどうかを判断します。AttributeValueList は、同じ型 (文字列、数値、またはバイナリ) の 2 つのスカラー要素を含む必要があります。

例

製品の料金が 100 ~ 200 の場合、次の表現は true と評価されます。

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

属性値 N を含む比較演算子

- IN - 値が列挙リスト内のいずれかの値と等しい場合、true です。リストではスカラー値のみがサポートされ、セットはサポートされません。一致するには、ターゲット属性は、同じ型と値である必要があります。

  AttributeValueList は、文字列、数値、またはバイナリの 1 つ以上の要素を含むことができます (セットではありません)。これらの属性は、項目の既存の非セット型属性と比較されます。入力セットのいずれかの要素が項目の属性に存在する場合、式は true に評価されます。

  AttributeValueList は、文字列、数値、またはバイナリの 1 つまたは複数の値を含むことができます (セットではありません)。比較のターゲット属性は、一致対象と同じ型および同じ値である必要があります。文字列は、文字列セットと一致しません。

この演算子を使用して、指定された値が列挙リスト内にあるかどうかを判断します。AttributeValueList ではスカラー値はいくつでも指定できますが、すべて同じデータ型である必要があります。

例

ID の値が 201、203、205 の場合、次の式は true と評価されます。

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

複数の条件の使用

DynamoDB では、複数の条件を組み合わせて、複雑な表現を作成できます。これを行うには、オプションの ConditionalOperator (レガシー) を使用して、少なくとも 2 つの式を指定します。

デフォルトでは、複数の条件を指定した場合、式全体が true に評価されるには、すべての条件が true に評価される必要があります。言い換えると、暗黙の AND オペレーションが行われます。

例

商品が 600 ページ以上の本の場合、次の表現は true と評価されます。どちらの条件も暗黙的に AND で処理されるので、true と評価される必要があります。

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600"} ]
    }
...
```

ConditionalOperator (レガシー) を使用して、AND オペレーションが行われることを明確にすることができます。次の例は、前の例と同様に動作します。

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600"} ]
    }
...
```

ConditionalOperator を OR に設定することもできます。その場合、少なくとも 1 つの条件が true に評価される必要があります。

例

製品がマウンテンバイクの場合、特定のブランド名である場合、または価格が 100 より大きい場合、次の表現は true と評価されます。

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> Note
>
> 複雑な表現では、条件は最初の条件から最後の条件まで順番に処理されます。1 つの表現で AND と OR の両方を使用することはできません。

その他の条件演算子

DynamoDB の以前のリリースでは、Expected パラメータの動作は、条件付き書き込みでは異なりました。Expected マップ内の各項目は、以下とともに、チェックする DynamoDB の属性名を表していました。

- `Value` - 属性に対して比較する値。
- `Exists` — オペレーションを試みる前に値が存在するかどうかを調べます。

`Value` と `Exists` のオプションは、引き続き DynamoDB でサポートされていますが、等価条件または属性が存在するかどうかのテストのみが可能です。`ComparisonOperator` と `AttributeValueList` オプションを使用すると、より広い範囲の条件を構築できるので、これらのオプションを使用することをお勧めします。

Example

`DeleteItem` は、本が出版されていないかどうかを確認し、この条件が true である場合にのみ削除できます。レガシー条件を使用する AWS CLI の例を次に示します。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

次の例では、同じことを実行しますが、レガシー条件は使用しません。

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

PutItem オペレーションは、同じプライマリキー属性を持つ既存の項目を上書きしないように保護できます。レガシー条件を使用する例を次に示します。

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

次の例では、同じことを実行しますが、レガシー条件は使用しません。

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

Note

Expected マップ内の条件には、レガシーの Value オプションと Exists オプションを ComparisonOperator と AttributeValueList と一緒に使用しないでください。一緒に使用すると、条件付き書き込みは失敗します。

以前の低レベル API バージョン (2011-12-05)

このセクションでは、以前の DynamoDB 低レベル API (2011-12-05) で利用可能なオペレーションについて説明します。このバージョンの低レベル API は、既存のアプリケーションとの下位互換性のために維持されています。

新しいアプリケーションでは、最新の API バージョン (2012-08-10) を使用してください。詳細については、「低レベル API リファレンス」を参照してください。

Note

DynamoDB の新機能は以前の API バージョンにバックポートされないため、既存のアプリケーションを新しい API バージョン (2012-08-10) に移行することをお勧めします。

トピック

- BatchGetItem
- BatchWriteItem
- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables
- GetItem
- ListTables
- PutItem
- Query
- Scan
- UpdateItem
- UpdateTable

BatchGetItem

Important

##########API ##### 2011-12-05 ### ***#######***

現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

BatchGetItem オペレーションは、プライマリキーを使用して、複数のテーブルの複数の項目の属性を返します。1 回のオペレーションで取得できる項目の最大数は 100 です。また、取得される項目の数は、1 MB のサイズ制限によって制限されます。テーブルのプロビジョニングされたスループットを超えたことや内部処理が失敗したことが原因でレスポンスサイズの制限を超えた場合または結果の一部だけが返された場合、DynamoDB は UnprocessedKeys 値を返すので、取得する次の項目からオペレーションを再試行できます。DynamoDB は、この制限を適用するために、ページごとに返される項目の数を自動的に調整します。例えば、100 個の項目を取得するようにリクエストしても、個々の項目のサイズが 50 KB の場合、返されるのは 20 個の項目だけなので、適切な UnprocessedKeys 値を設定して結果の次のページを取得できます。必要に応じて、アプリケーションに独自のロジックを組み込んで、結果のページを 1 つのセットにまとめることができます。

リクエストに含まれる各テーブルのプロビジョニングされたスループットが不十分だったことが原因で処理される項目がなかった場合、DynamoDB は ProvisionedThroughputExceededException エラーを返します。

> Note
>
> デフォルトでは、BatchGetItem は、リクエスト内の各テーブルに対して結果整合性のある読み込みを実行します。整合性のある読み込みが必要な場合は、テーブルごとに ConsistentRead パラメータを true に設定できます。
> BatchGetItem は、レスポンスのレイテンシーを最小限に抑えるために項目を並列で取得します。
> アプリケーションを設計する際は、DynamoDB では、返されるレスポンスで属性の順序が保証されないことに注意してください。リクエスト内の項目の AttributesToGet にプライマリキーバリューを含めると、項目ごとにレスポンスを解析するのに役立ちます。
> リクエストされた項目が存在しない場合、その項目のレスポンスでは何も返されません。存在しない項目に対するリクエストは、読み込みのタイプに応じた最小読み込み容量ユニットを消費します。詳細については、「DynamoDB 項目のサイズと形式」を参照してください。

リクエスト

構文

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
    {"Table1":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
    "Table2":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}}],
        "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
        }
    }
}
```

名前	説明	必須
`RequestItems`	プライマリキーで取得するテーブル名および対応する項目のコンテナ。項目をリクエストしている間、各テーブル名はオペレーションごとに 1 回だけ呼び出すことができます。 型: 文字列 デフォルト: なし	はい
`Table`	取得する項目を含んでいるテーブルの名前。エントリは、ラベルのない既存のテーブルを指定する文字列です。 型: 文字列	はい

名前	説明	必須
	デフォルト: なし	
`Table:Keys`	指定されたテーブルの項目を定義するプライマリのキーバリュー。プライマリキーの詳細については、「プライマリキー」を参照してください。 型: キー	はい
`Table:AttributesToGet`	指定したテーブル内の属性名の配列。属性名が指定されていない場合、すべての属性が返されます。見つからなかった属性は結果に表示されません。 型: 配列	いいえ
`Table:ConsistentRead`	`true` に設定されている場合、整合性のある読み込みが発行されます。それ以外の場合、結果整合性が使用されます。 型: ブール値	いいえ

レスポンス

構文

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 855

{"Responses":
    {"Table1":
        {"Items":
```

```
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue",
 "AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1},
    "Table2":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue","AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":
        {"Table3":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]}
        }
}
```

名前	説明
Responses	テーブル名とテーブルのそれぞれの項目属性。 型: マップ

名前	説明
`Table`	項目を含んでいるテーブルの名前。エントリは、ラベルのないテーブルを指定する文字列です。 型: 文字列
`Items`	オペレーションパラメータを満たす属性名と値のコンテナ。 型: 属性名およびそのデータ型と値のマップ。
`ConsumedCapacityUnits`	各テーブルで消費された読み込み容量ユニットの数。この値は、プロビジョニングされたスループットに適用される数を示します。存在しない項目に対するリクエストは、読み込みのタイプに応じた最小読み込み容量ユニットを消費します。詳細については、「プロビジョンドキャパシティモード」を参照してください。 型: 数値
`UnprocessedKeys`	レスポンスサイズの制限に達したなどの理由で、現在のレスポンスで処理されなかったテーブルの配列とそれぞれのキーが含まれます。`UnprocessedKeys` 値は `RequestItems` パラメータと同じ形式です (値は後続の `BatchGetItem` オペレーションに直接渡すことができます)。詳細については、上記の `RequestItems` パラメータを参照してください。 型: 配列

名前	説明
UnprocessedKeys : Table: Keys	項目と、項目に関連付けられた属性を定義するプライマリキー属性値。プライマリキーの詳細については、「プライマリキー」を参照してください。 型: 属性の名前と値のペアの配列。
UnprocessedKeys : Table: AttributesToGet	指定したテーブル内の属性名。属性名が指定されていない場合、すべての属性が返されます。見つからなかった属性は結果に表示されません。 型: 属性名の配列。
UnprocessedKeys : Table: ConsistentRead	true に設定されている場合、指定されたテーブルに対して整合性のある読み込みが使用されます。それ以外の場合は、結果整合性のある読み込みが使用されます。 型: ブール。

特殊なエラー

エラー	説明
ProvisionedThroughputExceededException	許可されているプロビジョニングスループットの最大値を超えました。

例

次の例は、BatchGetItem オペレーションを使用した HTTP POST リクエストとレスポンスを示しています。AWS SDK を使用した例については、「項目と属性の操作」を参照してください。

リクエスト例

次の例では、2 つの異なるテーブルの属性をリクエストします。

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
    {"comp1":
        {"Keys":
            [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
            {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
            {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
        "AttributesToGet":["user","status"]},
    "comp2":
        {"Keys":
            [{"HashKeyElement":{"S":"Julie"}},{"HashKeyElement":{"S":"Mingus"}}],
        "AttributesToGet":["user","friends"]}
    }
}
```

レスポンス例

以下の例がレスポンスです。

```
HTTP/1.1 200 OK
x-amzn-RequestId: GTPQVRM4VJS792J1UFJTKUBVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 373
Date: Fri, 02 Sep 2011 23:07:39 GMT

{"Responses":
    {"comp1":
        {"Items":
            [{"status":{"S":"online"},"user":{"S":"Casey"}},
            {"status":{"S":"working"},"user":{"S":"Riley"}},
            {"status":{"S":"running"},"user":{"S":"Dave"}}],
        "ConsumedCapacityUnits":1.5},
    "comp2":
        {"Items":
            [{"friends":{"SS":["Elisabeth", "Peter"]},"user":{"S":"Mingus"}},
            {"friends":{"SS":["Dave", "Peter"]},"user":{"S":"Julie"}}],
        "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":{}
}
```

BatchWriteItem

> Important
>
> ***##########API ##### 2011-12-05 ###***
>
> 現在の低レベルの API に関するドキュメントについては、[Amazon DynamoDB API リファレンス](#)を参照してください。

説明

このオペレーションでは、単一の呼び出しで、複数のテーブルの複数の項目を配置または削除できます。

1 つの項目をアップロードするには、`PutItem` を使用できます。1 つの項目を削除するには、`DeleteItem` を使用できます。ただし、Amazon EMR (Amazon EMR) から大量のデータをアップロードする場合や別のデータベースのデータを DynamoDB に移行する場合など、大量のデータをアップロードまたは削除する場合、`BatchWriteItem` では効率的な方法が用意されています。

Java などの言語を使用する場合は、スレッドを使用して項目を並列でアップロードできます。これにより、スレッドを処理するアプリケーションの複雑さが増します。他の言語はスレッドの使用をサポートしていません。例えば、PHP を使用している場合は、項目を 1 つずつアップロードまたは削除する必要があります。どちらの状況でも、`BatchWriteItem` は、指定されたアップロードおよび削除オペレーションを並列で処理する代替手段を提供し、アプリケーションを複雑にすることなくスレッドプールのアプローチの威力を発揮します。

`BatchWriteItem` オペレーションで指定された個々のアップロードと削除のコストは、消費される容量単位と同じです。ただし、指定したオペレーションが `BatchWriteItem` によって並列で実行されると、レイテンシーが小さくなります。存在しない項目に対する削除オペレーションは、書き込み容量ユニットを 1 つ消費します。消費される容量ユニットの詳細については、「[DynamoDB でのテーブルとデータの操作](#)」を参照してください。

`BatchWriteItem` を使用する場合、以下の制限に注意してください。

- 1 つのリクエストにおける最大オペレーション数 - 最大 25 個のアップロードまたは削除オペレーションを指定できます。ただし、リクエストの合計サイズは 1 MB (HTTP ペイロード) を超えることはできません。
- BatchWriteItem オペレーションは、項目のアップロードと削除行うためにのみ使用できます。既存の項目を更新するためには使用できません。
- アトミック操作ではありません - 1 つの BatchWriteItem で指定されている個々の操作はアトミックです。ただし、BatchWriteItem 全体が「ベストエフォート」操作であり、アトミック操作ではありません。したがって、BatchWriteItem リクエストでは、一部のオペレーションは成功し、他のオペレーションは失敗する可能性があります。失敗したオペレーションは、レスポンスの UnprocessedItems フィールドに返されます。これらの失敗の原因としては、テーブルに設定されているプロビジョニングされたスループットを超えたか、ネットワークエラーなどの一時的な障害が考えられます。リクエストを調査し、必要に応じてリクエストを再送信できます。通常、ループ内で BatchWriteItem を呼び出し、各反復で未処理の項目をチェックして、未処理の項目とともに新しい BatchWriteItem リクエストを送信します。
- 項目は返されません - BatchWriteItem は大量のデータを効率的にアップロードするように設計されています。PutItem および DeleteItem で提供されている高度な機能の一部は提供されていません。例えば、DeleteItem は、レスポンス内の削除済み項目をリクエストするためにリクエストボディ内の ReturnValues フィールドをサポートします。BatchWriteItem オペレーションはレスポンスで項目を返しません。
- PutItem や DeleteItem とは異なり、BatchWriteItem では、オペレーション内の個々の書き込みリクエストに対して条件を指定することはできません。
- 属性値は NULL であってはなりません。文字列型およびバイナリ型属性の長さは 0 より大きくなければなりません。また、セット型属性は空であってはなりません。空の値を持つリクエストは、ValidationException で拒否されます。

DynamoDB は、次のいずれかが真である (true) 場合バッチ書き込みオペレーション全体を拒否します。

- BatchWriteItem リクエスト内で指定したテーブルが存在しない。
- リクエスト内の項目で指定されたプライマリキー属性が、対応するテーブルのプライマリキースキーマと一致しない。
- 同じ BatchWriteItem リクエストで同じ項目に複数のオペレーションを実行する。例えば、同じ項目を同一の BatchWriteItem リクエストでアップロードおよび削除することはできません。
- 合計リクエストサイズが 1 MB のリクエストサイズ (HTTP ペイロード) 制限を超過する。
- バッチ内の個々の項目が 64 KB の項目のサイズ制限を超えている。

リクエスト

Syntax

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems" :  RequestItems
}

RequestItems
{
     "TableName1" :  [ Request, Request, ... ],
     "TableName2" :  [ Request, Request, ... ],
     ...
}

Request ::=
  PutRequest | DeleteRequest

PutRequest ::=
{
  "PutRequest" : {
     "Item" : {
        "Attribute-Name1" : Attribute-Value,
        "Attribute-Name2" : Attribute-Value,
         ...
     }
  }
}

DeleteRequest ::=
{
   "DeleteRequest" : {
      "Key" : PrimaryKey-Value
   }
}


PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK
```

```
HashTypePK ::=
{
   "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
   "HashKeyElement" : Attribute-Value,
   "RangeKeyElement" : Attribute-Value,
}

Attribute-Value ::= String | Numeric| Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
   "N": "Number"
}

String ::=
{
   "S": "String"
}

Binary ::=
{
    "B": "Base64 encoded binary data"
}

StringSet ::=
{
   "SS": [ "String1", "String2", ... ]
}

NumberSet ::=
{
   "NS": [ "Number1", "Number2", ... ]
}

BinarySet ::=
{
   "BS": [ "Binary1", "Binary2", ... ]
}
```

リクエストボディでは、RequestItems JSON オブジェクトは、実行するオペレーションを記述します。オペレーションはテーブルごとにグループ化されます。BatchWriteItem を使用して、複数のテーブル間で複数の項目を更新または削除できます。書き込みリクエストごとに、リクエストのタイプ (PutItem、DeleteItem) を指定した後にオペレーションに関する詳細情報を指定する必要があります。

- PutRequest では、項目 (属性とその値のリスト) を指定します。
- DeleteRequest では、プライマリキーの名前と値を指定します。

レスポンス

Syntax

レスポンスで返される JSON 本文の構文を以下に示します。

```
{
  "Responses" :          ConsumedCapacityUnitsByTable
  "UnprocessedItems" :   RequestItems
}

ConsumedCapacityUnitsByTable
{
    "TableName1" : { "ConsumedCapacityUnits", : NumericValue },
    "TableName2" : { "ConsumedCapacityUnits", : NumericValue },
     ...
}

RequestItems
This syntax is identical to the one described in the JSON syntax in the request.
```

特殊なエラー

このオペレーションに固有のエラーはありません。

例

次の例は、HTTP POST リクエストと BatchWriteItem オペレーションのレスポンスを示しています。リクエストは、Reply テーブルと Thread テーブルに対して以下のオペレーションを指定します。

- Reply テーブルに対して 1 つの項目の配置と削除を行う

- Thread テーブルに項目を配置する

AWS SDK を使用した例については、「項目と属性の操作」を参照してください。

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems":{
    "Reply":[
      {
        "PutRequest":{
          "Item":{
            "ReplyDateTime":{
              "S":"2012-04-03T11:04:47.034Z"
            },
            "Id":{
              "S":"DynamoDB#DynamoDB Thread 5"
            }
          }
        }
      },
      {
        "DeleteRequest":{
          "Key":{
            "HashKeyElement":{
              "S":"DynamoDB#DynamoDB Thread 4"
            },
            "RangeKeyElement":{
              "S":"oops - accidental row"
            }
          }
        }
      }
    ],
    "Thread":[
      {
        "PutRequest":{
          "Item":{
            "ForumName":{
              "S":"DynamoDB"
            },
            "Subject":{
              "S":"DynamoDB Thread 5"
            }
          }
        }
      }
    ]
  }
}
```

レスポンス例

次のレスポンスの例は、Thread テーブルと Reply テーブルの両方に対する配置オペレーションが成功し、Reply テーブルに対する削除オペレーションが (テーブルでプロビジョニングされたスループットを超えた場合に発生するスロットリングなどの理由で) 失敗したことを示しています。JSON レスポンスでは、以下の点に注意してください。

- `Responses` オブジェクトは、`Thread` テーブルと `Reply` テーブルのそれぞれで成功した配置オペレーションの結果として両方のテーブルで 1 つの容量ユニットが消費されます。
- `UnprocessedItems` オブジェクトには、`Reply` テーブルで正常に完了しなかった削除オペレーションが示されます。その後、新しい `BatchWriteItem` 呼び出しを発行して、これらの未処理のリクエストに対処できます。

```
HTTP/1.1 200 OK
x-amzn-RequestId: G8M9ANLOE5QA26AEUHJKJE0ASBVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 536
Date: Thu, 05 Apr 2012 18:22:09 GMT

{
   "Responses":{
      "Thread":{
         "ConsumedCapacityUnits":1.0
      },
      "Reply":{
         "ConsumedCapacityUnits":1.0
      }
```

```
    },
    "UnprocessedItems":{
       "Reply":[
          {
             "DeleteRequest":{
                "Key":{
                   "HashKeyElement":{
                      "S":"DynamoDB#DynamoDB Thread 4"
                   },
                   "RangeKeyElement":{
                      "S":"oops - accidental row"
                   }
                }
             }
          }
       ]
    }
 }
```

CreateTable

Important

***##########API ##### 2011-12-05 ######################################
#######***

現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

`CreateTable` オペレーションは新しいテーブルをアカウントに追加します。

テーブル名は、リクエストを発行する AWS アカウントに関連付けられている名前と、リクエストを受信する AWS リージョン (dynamodb.us-west-2.amazonaws.com など) の中で一意である必要があります。各 DynamoDB エンドポイントは完全に独立しています。例えば、"MyTable" というテーブルが dynamodb.us-west-2.amazonaws.com と dynamodb.us-west-1.amazonaws.com にある場合、この 2 つの "MyTable" テーブルは相互に独立していてデータを共有しません。

CreateTable オペレーションは、テーブルの作成を開始する非同期ワークフローをトリガーします。DynamoDB は、テーブルが ACTIVE 状態になるまで直ちにテーブルの状態 (CREATING) を返します。テーブルが ACTIVE 状態になると、データプレーンオペレーションを実行できます。

テーブルのステータスを確認するには、DescribeTables オペレーションを使用します。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

名前	説明	必須
TableName	作成するテーブルの名前。 使用できる文字は、a～z、A～Z、0～9、'_' (アンダースコア)、'-' (ダッシュ)、'.' (ドット) です。名前は 3～255 文字の長さです。 型: 文字列	はい
KeySchema	テーブルのプライマリキー (シンプルまたは複合) 構造。HashKeyElement の名前と値のペアは必須です。RangeKeyElement の	はい

名前	説明	必須
	名前と値のペアはオプションです (複合プライマリキーにのみ必要)。プライマリキーの詳細については、「プライマリキー」を参照してください。 プライマリキー要素の名前は 1～255 文字の長さで、使用できる文字の制限はありません。 AttributeType に指定できる値は、「S」(文字列)、「N」(数値)、または「B」(バイナリ) です。 型: 複合プライマリキーの `HashKeyElement` 、または `HashKeyElement` および `RangeKeyElement` のマップ。	

名前	説明	必須
`ProvisionedThroughput`	指定したテーブルの新しいスループット (`ReadCapacityUnits` と `WriteCapacityUnits` の値で構成されます)。詳細については、「プロビジョンドキャパシティモード」を参照してください。 ⓘ Note 現在の最大値/最小値については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。 型: 配列	はい

名前	説明	必須
ProvisionedThroughput : ReadCapacityUnits	DynamoDB がロードとその他のオペレーションのバランスを取る前に、指定したテーブルで 1 秒あたりに消費される整合性のある ReadCapacityUnits の最小数を設定します。 結果整合性のある読み込みオペレーションに必要な労力は、整合性のある読み込みオペレーションより少ないので、1 秒あたり 50 の整合性のある ReadCapacityUnits の設定では、1 秒あたり 100 の結果整合性のある ReadCapacityUnits が実現します。 型: 数値	はい
ProvisionedThroughput : WriteCapacityUnits	DynamoDB がロードとその他のオペレーションのバランスを取る前に、指定したテーブルで 1 秒あたりに消費される整合性のある WriteCapacityUnits の最小数を設定します 型: 数値	はい

レスポンス

構文

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"CREATING"
    }
}
```

名前	説明
`TableDescription`	テーブルプロパティのコンテナ。
`CreationDateTime`	テーブルの作成日 (UNIX エポック時間)。 型: 数値
`KeySchema`	テーブルのプライマリキー (シンプルまたは複合) 構造。`HashKeyElement` の名前と値のペアは必須です。`RangeKeyElement` の名前と値のペアはオプションです (複合プライマリキーにのみ必要)。プライマリキーの詳細については、「プライマリキー」を参照してください。 型: 複合プライマリキーの `HashKeyElement` 、または `HashKeyElement` および `RangeKeyElement` のマップ。

名前	説明
ProvisionedThroughput	指定したテーブルのスループット (ReadCapacityUnits と WriteCapacityUnits の値で構成されます)。「プロビジョンドキャパシティモード」を参照してください。 型: 配列
ProvisionedThroughput :ReadCapacityUnits	DynamoDB がロードとその他のオペレーションのバランスを取る前に 1 秒あたりに消費される ReadCapacityUnits の最小数。 型: 数値
ProvisionedThroughput :WriteCapacityUnits	WriteCapacityUnits がロードとその他のオペレーションのバランスを取る前に 1 秒あたりに消費される ReadCapacityUnits の最小数。 型: 数値
TableName	作成されたテーブルの名前。 型: 文字列
TableStatus	テーブルの現在の状態 (CREATING)。テーブルが ACTIVE 状態になったらデータを配置できます。 テーブルのステータスを確認するには、DescribeTables API を使用します。 型: 文字列

特殊なエラー

エラー	説明
`ResourceInUseException`	既存のテーブルを再度作成しようとしました。
`LimitExceededException`	同時テーブルリクエストの数 (`CREATING`、`DELETING`、または `UPDATING` の状態にある累積的なテーブル数) が許容最大値を超えています。 Note 現在の最大値/最小値については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。 .

例

次の例は、文字列と数値を含む複合プライマリキーを持つテーブルを作成します。AWS SDK を使用した例については、「DynamoDB でのテーブルとデータの操作」を参照してください。

リクエスト例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0


{"TableName":"comp-table",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

レスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"comp-table",
    "TableStatus":"CREATING"
    }
}
```

関連アクション

- DescribeTables
- DeleteTable

DeleteItem

Important

***##########API ##### 2011-12-05 ######################################
#######***

現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

プライマリキーを使用してテーブルから 1 つの項目を削除します。項目が存在する場合、または予期される属性値が項目にある場合は、項目を削除する条件付き削除オペレーションを実行できます。

> Note
>
> 属性または値なしで `DeleteItem` を指定すると、項目のすべての属性が削除されます。
> 条件を指定した場合を除き、`DeleteItem` は、べき等性の操作です。同じ項目または属性に対して複数回実行してもエラーレスポンスが返されません。
> 条件付き削除は、特定の条件が満たされた場合にのみ項目と属性を削除する場合に便利です。条件が満たされると、DynamoDB は削除を実行します。それ以外の場合、項目は削除されません。
> 1 度のオペレーションにつき 1 つの属性に対して、予期される条件チェックを実行できます。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
    "ReturnValues":"ALL_OLD"}
}
```

名前	説明	必須
`TableName`	削除する項目を含むテーブルの名前。	はい

名前	説明	必須
	型: 文字列	
Key	項目を定義するプライマリキー。プライマリキーの詳細については、「プライマリキー」を参照してください。 型: HashKeyElement の値に対するマップと RangeKeyElement の値に対するマップ。	はい
Expected	条件付き削除の属性を指定します。Expected パラメータを使用すると、属性名を指定できます。また、DynamoDB が属性を削除する前に特定の値があるかどうかを確認するかどうかも指定できます。 型: 属性名のマップ。	いいえ
Expected:AttributeName	条件付き配置の属性の名前。 型: 文字列	いいえ

名前	説明	必須

`Expected:AttributeName: ExpectedAttributeValue`	このパラメータを使用して、属性名と値のペアに値が既に存在するかどうかを指定します。
	次の JSON 表記法では、「Color」属性が存在しない項目が削除されます。
	`"Expected" : {"Color":{"Exists":false}}`
	次の JSON 表記法では、項目を削除する前に、「Color」という名前の属性の既存の値が「Yellow」であるかどうかがチェックされます。
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	デフォルトでは、`Expected` パラメータを使用し、`Value` を指定した場合、DynamoDB は属性が存在し、置換する現在の値があると仮定します。したがって、`{"Exists":true}` は暗示されているので指定する必要はありません。リクエストを短縮するには、次のようにします。
	`"Expected" :`
		いいえ

名前	説明	必須
	`{"Color":{"Value":{"S":"Yellow"}}}` Note チェックする属性値なしで `{"Exists":true}` を指定すると、DynamoDB はエラーを返します。	
`ReturnValues`	削除する前に属性名と値のペアを取得する場合、このパラメータを使用します。使用できるパラメータの値は、`NONE` (デフォルト) または `ALL_OLD` です。`ALL_OLD` を指定した場合、古い項目の内容が返されます。このパラメータを指定しない場合、または `NONE` の場合、何も返されません。 型: 文字列	いいえ

レスポンス

構文

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"Attributes":
    {"AttributeName3":{"SS":["AttributeValue3","AttributeValue4","AttributeValue5"]},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName1":{"N":"AttributeValue1"}
    },
"ConsumedCapacityUnits":1
}
```

名前	説明
Attributes	リクエストで ReturnValues パラメータが ALL_OLD として指定されている場合、DynamoDB は属性名と値のペア (実質的に、削除された項目) の配列を返します。それ以外の場合、レスポンスには空のセットが含まれます。 型: 属性の名前と値のペアの配列。
ConsumedCapacityUnits	オペレーションによって消費される書き込み容量ユニットの数。この値は、プロビジョニングされたスループットに適用される数を示します。存在しない項目に対する削除リクエストは、書き込み容量ユニットを 1 つ消費します。詳細については、「プロビジョンドキャパシティモード」を参照してください。 型: 数値

特殊なエラー

エラー	説明
ConditionalCheckFailedException	条件チェックに失敗しました。予期された属性値が見つかりませんでした。

例

リクエスト例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
    "Key":
        {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
    "Expected":
        {"status":{"Value":{"S":"shopping"}}},
    "ReturnValues":"ALL_OLD"
}
```

レスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: U9809LI6BBFJA5N2R0TB0P017JVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 22:31:23 GMT

{"Attributes":
    {"friends":{"SS":["Dooley","Ben","Daisy"]},
    "status":{"S":"shopping"},
    "time":{"N":"200"},
    "user":{"S":"Mingus"}
    },
"ConsumedCapacityUnits":1
}
```

関連アクション

- PutItem

DeleteTable

Important

##########API ##### 2011-12-05 ###

現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

`DeleteTable` オペレーションは、テーブルとそのすべての項目を削除します。`DeleteTable` リクエストの後、指定されたテーブルは、DynamoDB が削除を完了するまで `DELETING` 状態になります。`ACTIVE` 状態になったらテーブルを削除できます。テーブルが `CREATING` または `UPDATING` 状態の場合、DynamoDB は `ResourceInUseException` エラーを返します。指定されたテーブルが存在しない場合、DynamoDB は、`ResourceNotFoundException` を返します。テーブルが既に `DELETING` 状態である場合、エラーは返されません。

Note

DynamoDB は、テーブルの削除が完了するまで、`DELETING` 状態のテーブルで引き続きデータプレーンオペレーションのリクエスト (`GetItem` および `PutItem` など) を受け入れることがあります。

テーブルは、リクエストを発行する AWS アカウントに関連付けられているテーブルと、リクエストを受信する AWS リージョン(dynamodb.us-west-1.amazonaws.com など) の中で一意です。各 DynamoDB エンドポイントは完全に独立しています。例えば、"MyTable" というテーブルが dynamodb.us-west-2.amazonaws.com と dynamodb.us-west-1.amazonaws.com にある場合、この 2 つの "MyTable" テーブルは相互に独立していてデータを共有しません。一方を削除しても、もう一方は削除されません。

テーブルのステータスを確認するには、DescribeTables オペレーションを使用します。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

名前	説明	必須
TableName	削除するテーブルの名前。 型: 文字列	はい

レスポンス

構文

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Sun, 14 Aug 2011 22:56:22 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"DELETING"
    }
}
```

名前	説明
TableDescription	テーブルプロパティのコンテナ。
CreationDateTime	テーブルの作成日。 型: 数値
KeySchema	テーブルのプライマリキー (シンプルまたは複合) 構造。HashKeyElement の名前と値のペアは必須です。RangeKeyElement の名前と値のペアはオプションです (複合プライマリキーにのみ必要)。プライマリキーの詳細については、「プライマリキー」を参照してください。 型: 複合プライマリキーの HashKeyElement 、または HashKeyElement および RangeKeyElement のマップ。
ProvisionedThroughput	指定したテーブルのスループット (ReadCapacityUnits と WriteCapacityUnits の値で構成されます)。「プロビジョンドキャパシティモード」を参照してください。
ProvisionedThroughput : ReadCapacityUnits	DynamoDB がロードとその他のオペレーションのバランスを取る前に、指定したテーブルで 1 秒あたりに消費される ReadCapacityUnits の最小数。 型: 数値
ProvisionedThroughput :WriteCapacityUnits	DynamoDB がロードとその他のオペレーションのバランスを取る前に、指定したテーブルで 1 秒あたりに消費される WriteCapacityUnits の最小数。 型: 数値

名前	説明
TableName	削除されたテーブルの名前。 型: 文字列
TableStatus	テーブルの現在の状態 (DELETING)。テーブルが削除されると、テーブルに対する後続のリクエストは resource not found を返します。 テーブルのステータスを確認するには、DescribeTables オペレーションを使用します。 型: 文字列

特殊なエラー

エラー	説明
ResourceInUseException	CREATING または UPDATING の状態のテーブルは削除できません。

例

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0
content-length: 40

{"TableName":"favorite-movies-table"}
```

レスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 160
Date: Sun, 14 Aug 2011 17:20:03 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"name","AttributeType":"S"}},
    "TableName":"favorite-movies-table",
    "TableStatus":"DELETING"
}
```

関連アクション

- CreateTable
- DescribeTables

DescribeTables

Important

***##########API ##### 2011-12-05 ######################################
#######***

現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

テーブルに関する情報 (テーブルの現在のステータス、プライマリキーのスキーマ、テーブルが作成された日時など) を返します。DescribeTable の結果は結果整合性です。テーブル作成のプロセスで DescribeTable を使用するタイミングが早すぎる場合、DynamoDB は ResourceNotFoundException を返します。テーブル更新のプロセスで DescribeTable を使用するタイミングが早すぎる場合、新しい値がすぐに使用できないことがあります。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

名前	説明	必須
`TableName`	説明するテーブルの名前。 型: 文字列	はい

レスポンス

構文

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
Content-Length: 543


{"Table":
    {"CreationDateTime":1.309988345372E9,
    ItemCount:1,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": Date, "LastDecreaseDateTime":
 Date, "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableSizeBytes":1,
    "TableStatus":"ACTIVE"
    }
}
```

名前	説明
Table	記述されているテーブルのコンテナ。 型: 文字列
CreationDateTime	テーブルの作成日 (UNIX エポック時間)。
ItemCount	指定されたテーブルに含まれる項目の数。DynamoDB は、この値を約 6 時間ごとに更新します。最近の変更は、この値に反映されないことがあります。 型: 数値
KeySchema	テーブルのプライマリキー (シンプルまたは複合) 構造。HashKeyElement の名前と値のペアは必須です。RangeKeyElement の名前と値のペアはオプションです (複合プライマリキーにのみ必要)。ハッシュキーの最大サイズは 2048 バイトです。範囲キーの最大サイズは 1024 バイトです。両方の制限は別々に強制されます (ハッシュと範囲の合計で 2048 + 1024 キーを組み合わせることができます)。プライマリキーの詳細については、「プライマリキー」を参照してください。
ProvisionedThroughput	指定されたテーブルのスループット。LastIncreaseDateTime (該当する場合)、LastDecreaseDateTime (該当する場合)、ReadCapacityUnits 、および WriteCapacityUnits で構成されます。テーブルのスループットが増減したことがない場合、DynamoDB はこれらの要素の値を返しません。「プロビジョンドキャパシティモード」を参照してください。

名前	説明
	型: 配列
`TableName`	リクエストされたテーブルの名前。 型: 文字列
`TableSizeBytes`	指定されたテーブルの合計サイズ (バイト単位)。DynamoDB は、この値を約 6 時間ごとに更新します。最近の変更は、この値に反映されないことがあります。 型: 数値
`TableStatus`	テーブルの現在の状態 (`CREATING`、`ACTIVE`、`DELETING`、または `UPDATING`)。テーブルが `ACTIVE` の状態になったらデータを追加できます。

特殊なエラー

このオペレーションに固有のエラーはありません。

例

次の例は、「comp-table」という名前のテーブルに対する DescribeTable オペレーションを使用した HTTP POST リクエストとレスポンスを示しています。テーブルには複合プライマリキーがあります。

リクエスト例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

レスポンス例

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 543

{"Table":
    {"CreationDateTime":1.309988345372E9,
    "ItemCount":23,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": 1.309988345384E9,
 "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"users",
    "TableSizeBytes":949,
    "TableStatus":"ACTIVE"
    }
}
```

関連アクション

- CreateTable
- DeleteTable
- ListTables

GetItem

> Important
>
> ***##########API ##### 2011-12-05 ######################################
> #######***
>
> 現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

GetItem オペレーションは、プライマリキーに一致する項目の Attributes のセットを返します。一致する項目がない場合、GetItem はデータを返しません。

デフォルトでは、GetItem オペレーションは、結果整合性のある読み込みを提供します。結果整合性のある読み込みがアプリケーションで受け入れられない場合は、ConsistentRead を使用します。このオペレーションは標準の読み込みよりも時間がかかることがありますが、常に最後に更新された値を返します。詳細については、「読み込み整合性」を参照してください。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
  "Key":
  {"HashKeyElement": {"S":"AttributeValue1"},
  "RangeKeyElement": {"N":"AttributeValue2"}
 },
 "AttributesToGet":["AttributeName3","AttributeName4"],
 "ConsistentRead":Boolean
}
```

名前	説明	必須
TableName	リクエストされた項目を含むテーブルの名前。 型: 文字列	はい
Key	項目を定義するプライマリキーバリュー。プライマリキーの詳細については、「プライマリキー」を参照してください。	はい

名前	説明	必須
	型: HashKeyElement の値に対するマップと RangeKeyElement の値に対するマップ。	
AttributesToGet	属性名の配列。属性名が指定されていない場合、すべての属性が返されます。見つからなかった属性は結果に表示されません。 型: 配列	いいえ
ConsistentRead	true に設定されている場合、整合性のある読み込みが発行されます。それ以外の場合、結果整合性が使用されます。 型: ブール値	いいえ

レスポンス

構文

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 144

{"Item":{
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName4":{"N":"AttributeValue4"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits": 0.5
}
```

名前	説明
Item	リクエストされた属性が含まれます。 型: 属性の名前と値のペアのマップ。
ConsumedCapacityUnits	オペレーションによって消費される読み込み容量ユニットの数。この値は、プロビジョニングされたスループットに適用される数を示します。存在しない項目に対するリクエストは、読み込みのタイプに応じた最小読み込み容量ユニットを消費します。詳細については、「プロビジョンドキャパシティモード」を参照してください。 型: 数値

特殊なエラー

このオペレーションに固有のエラーはありません。

例

AWS SDK を使用した例については、「項目と属性の操作」を参照してください。

リクエスト例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"comptable",
 "Key":
  {"HashKeyElement":{"S":"Julie"},
  "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

レスポンス例

ConsumedCapacityUnits の値が 1 であることに注意してください。これは、オプションのパラメータ `ConsistentRead` が `true` に設定されているからです。同じリクエストで `ConsistentRead` が `false` に設定されている場合 (または指定されていない場合)、レスポンスは結果整合性で、ConsumedCapacityUnits の値は 0.5 になります。

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 72

{"Item":
 {"friends":{"SS":["Lynda, Aaron"]},
 "status":{"S":"online"}
 },
"ConsumedCapacityUnits": 1
}
```

ListTables

Important

##########API ##### 2011-12-05 ### #######

現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

現在のアカウントおよびエンドポイントに関連付けられたテーブルの配列を返します。

各 DynamoDB エンドポイントは完全に独立しています。例えば、"MyTable" というテーブルが dynamodb.us-west-2.amazonaws.com と dynamodb.us-east-1.amazonaws.com にある場合、この 2 つの "MyTable" テーブルは相互に独立していてデータを共有しません。ListTables オペレーションは、リクエストを受信するエンドポイントについて、リクエストを作成するアカウントに関連付けられたすべてのテーブル名を返します。

リクエスト

Syntax

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

デフォルトでは、ListTables オペレーションは、リクエストを受信するエンドポイントについて、リクエストを作成するアカウントに関連付けられたすべてのテーブル名をリクエストします。

名前	説明	必須
`Limit`	返されるテーブル名の最大数。 型: 整数	いいえ
`ExclusiveStartTableName`	リストの最初のテーブルの名前。すでに ListTables オペレーションを実行し、レスポンスで `LastEvaluatedTableName` 値を受け取っている場合、その値をここで使用してリストを続行します。 型: 文字列	[いいえ]

レスポンス

Syntax

```
HTTP/1.1 200 OK
```

```
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"TableNames":["Table1","Table2","Table3"], "LastEvaluatedTableName":"Table3"}
```

名前	説明
`TableNames`	現在のエンドポイントの現在のアカウントに関連付けられているテーブルの名前。 型: 配列
`LastEvaluatedTableName`	現在のリストの最後のテーブルの名前 (アカウントとエンドポイントの一部のテーブルが返されていない場合のみ)。すべてのテーブル名がすでに返されている場合、この値はレスポンスに存在しません。すべてのテーブル名が返されるまでリストを続行するには、この値を新しいリクエスト内で `ExclusiveStartTableName` として使用します。 型: 文字列

特殊なエラー

このオペレーションに固有のエラーはありません。

例

次の例は、ListTables オペレーションを使用した HTTP POST リクエストとレスポンスを示しています。

リクエスト例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

レスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"LastEvaluatedTableName":"comp5","TableNames":["comp3","comp4","comp5"]}
```

関連アクション

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> Important
>
> ***##########API ##### 2011-12-05 ######################################
> #######***
> 現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

新しい項目を作成するか、古い項目を新しい項目で置き換えます (すべての属性を含む)。指定されたテーブルに同じプライマリキーを持つ項目がすでに存在する場合、新しい項目によって既存の項目が完全に置き換えられます。条件付き配置 (指定されたプライマリキーを持つ項目が存在しない場合に新しい項目を挿入) を実行することや、特定の属性値を持つ既存の項目を置き換えることができます。

属性値は NULL であってはなりません。文字列型およびバイナリ型属性の長さは 0 より大きくなければなりません。また、セット型属性は空であってはなりません。空の値を持つリクエストは、ValidationException で拒否されます。

> Note
>
> 新しい項目によって既存の項目が置き換えられないようにするには、プライマリキー属性に対して Exists を false に設定した条件付き配置オペレーション、または属性を使用します。

PutItem の使用の詳細については、「項目と属性の操作」を参照してください。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Item":{
        "AttributeName1":{"S":"AttributeValue1"},
        "AttributeName2":{"N":"AttributeValue2"},
        "AttributeName5":{"B":"dmFsdWU="}
    },
    "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
    "ReturnValues":"ReturnValuesConstant"}
```

名前	説明	必須
TableName	項目を含めるテーブルの名前。 型: 文字列	はい
Item	項目の属性のマップ。項目を定義するプライマリキーバ	はい

名前	説明	必須
	リューを含む必要があります。項目には、他の属性名と値のペアを指定できます。プライマリキーの詳細については、「プライマリキー」を参照してください。 型: 属性名から属性値へのマップ。	
`Expected`	条件付き配置の属性を指定します。`Expected` パラメータを使用すると、属性名を指定できます。また、DynamoDB で更新を行う前に、属性値が既に存在するかどうか、属性値が存在する場合は特定の値があるかどうかの確認の有無も指定できます。 型: 属性名の属性値へのマップ、および属性値が存在するかどうか。	いいえ
`Expected:AttributeName`	条件付き配置の属性の名前。 型: 文字列	いいえ

名前	説明	必須

`Expected:AttributeName: ExpectedAttributeValue`	このパラメータを使用して、属性名と値のペアに値が既に存在するかどうかを指定します。
	次の JSON 表記法では、項目に「Color」属性がまだ存在しない場合、その項目が置き換えられます。
	`"Expected" : {"Color":{"Exists":false}}`
	次の JSON 表記法では、項目を置き換える前に、「Color」という名前の属性の既存の値が「Yellow」であるかどうかがチェックされます。
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}`
	デフォルトでは、`Expected` パラメータを使用し、`Value` を指定した場合、DynamoDB は属性が存在し、置換する現在の値があると仮定します。したがって、`{"Exists":true}` は暗示されているので指定する必要はありません。リクエストを短縮するには、次のようにします。
		いいえ

名前	説明	必須
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` Note チェックする属性値なしで `{"Exists":true}` を指定すると、DynamoDB はエラーを返します。	
`ReturnValues`	`PutItem` リクエストで属性名と値のペアを更新する前に属性名と値のペアを取得する場合、このパラメータを使用します。使用できるパラメータの値は、`NONE` (デフォルト) または `ALL_OLD` です。`ALL_OLD` を指定し、`PutItem` によって属性名と値のペアが上書きされた場合、古い項目の内容が返されます。このパラメータを指定しない場合、または `NONE` の場合、何も返されません。 型: 文字列	いいえ

レスポンス

構文

次の構文例では、`ALL_OLD` の `ReturnValues` パラメータが指定されたリクエストを引き受けます。指定されていない場合、レスポンスには `ConsumedCapacityUnits` 要素のみが含まれます。

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 85

{"Attributes":
 {"AttributeName3":{"S":"AttributeValue3"},
 "AttributeName2":{"SS":"AttributeValue2"},
 "AttributeName1":{"SS":"AttributeValue1"},
 },
"ConsumedCapacityUnits":1
}
```

名前	説明
Attributes	配置オペレーションの前の属性値 (リクエストで ReturnValues　パラメータが ALL_OLD として指定されている場合のみ)。 型: 属性の名前と値のペアのマップ。
ConsumedCapacityUnits	オペレーションによって消費される書き込み容量ユニットの数。この値は、プロビジョニングされたスループットに適用される数を示します。詳細については、「プロビジョンドキャパシティモード」を参照してください。 型: 数値

特殊なエラー

エラー	説明
ConditionalCheckFailedException	条件チェックに失敗しました。予期された属性値が見つかりませんでした。
ResourceNotFoundException	指定された項目または属性が見つかりませんでした。

例

AWS SDK を使用した例については、「項目と属性の操作」を参照してください。

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Item":
  {"time":{"N":"300"},
  "feeling":{"S":"not surprised"},
  "user":{"S":"Riley"}
  },
 "Expected":
  {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
 "ReturnValues":"ALL_OLD"
}
```

レスポンス例

```
HTTP/1.1 200
x-amzn-RequestId: 8952fa74-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 84

{"Attributes":
 {"feeling":{"S":"surprised"},
 "time":{"N":"300"},
 "user":{"S":"Riley"}},
"ConsumedCapacityUnits":1
}
```

関連アクション

- UpdateItem
- DeleteItem
- GetItem

- BatchGetItem

Query

Important

##########API ##### 2011-12-05 ###

現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

Query オペレーションオペレーションは、1 つ以上の項目の値とその属性をプライマリキーごとに取得します (Query はハッシュと範囲のプライマリキーテーブルでのみ使用できます)。特定の HashKeyValue を指定する必要があります。プライマリキーの RangeKeyValue で比較オペレーションを使用してクエリの範囲を絞り込むことができます。範囲キーごとに正順または逆順で結果を取得するには、ScanIndexForward パラメータを使用します。

結果を返さないクエリは、読み込みのタイプに応じて最小読み込み容量ユニットを消費します。

Note

クエリパラメータを満たす項目の合計数が 1 MB の制限を超えるとクエリが停止し、結果が後続のオペレーションでクエリを続行するための LastEvaluatedKey とともにユーザーに返されます。スキャン操作とは異なり、クエリ操作では、空の結果セットおよび LastEvaluatedKey が返されません。結果が 1 MB を超える場合、または Limit パラメータを使用した場合、LastEvaluatedKey のみが返されます。

ConsistentRead パラメータを使用して、結果を整合性のある読み込みに設定できます。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
```

```
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
[{"N":"AttributeValue2"}],"ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"AttributeName1"},
  "RangeKeyElement":{"N":"AttributeName2"}
 },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

名前	説明	必須
`TableName`	リクエストされた項目を含むテーブルの名前。 型: 文字列	はい
`AttributesToGet`	属性名の配列。属性名が指定されていない場合、すべての属性が返されます。見つからなかった属性は結果に表示されません。 型: 配列	いいえ
`Limit`	返される項目の最大数 (一致する項目の数であるとは限りません)。テーブルをクエリする間に制限までの項目数を処理した場合、DynamoDB はクエリを停止し、その時点までの一致する値とともに、後続のオペレーションでクエ	いいえ

名前	説明	必須
	リを続行するために適用する `LastEvaluatedKey` を返します。また、この制限に達する前に結果セットのサイズが 1 MB を超えた場合、DynamoDB はクエリを停止し、一致する値とともに、後続のオペレーションでクエリをクリックするために適用する `LastEvaluatedKey` を返します。 型: 数値	
`ConsistentRead`	`true` に設定されている場合、整合性のある読み込みが発行されます。それ以外の場合、結果整合性が使用されます。 型: ブール値	いいえ

名前	説明	必須
Count	true に設定されている場合、DynamoDB は、一致する項目とその属性のリストではなく、クエリパラメータに一致する項目の合計数を返します。Limit パラメータをカウントのみのクエリに適用できます。 AttributesToGet のリストを提供する場合、Count を true に設定しないでください。そうでないと、DynamoDB は検証エラーを返します。詳細については、「結果での項目のカウント」を参照してください。 型: ブール値	いいえ
HashKeyValue	複合プライマリキーのハッシュコンポーネントの属性値。 型: 文字列、数値、またはバイナリ	はい

名前	説明	必須
RangeKeyCondition	クエリに使用する属性値および比較演算子のコンテナ。クエリリクエストには RangeKeyCondition は必要ありません。HashKeyValue だけを指定した場合、DynamoDB は、指定されたハッシュキー要素値を持つすべての項目を返します。 型: マップ	いいえ
RangeKeyCondition : AttributeValueList	クエリパラメータについて評価する属性値。BETWEEN 比較を指定した場合を除き、AttributeValueList には属性値が 1 つ含まれます。BETWEEN 比較では、AttributeValueList には、2 つの属性値が含まれます。 型: ComparisonOperator への AttributeValue のマップ。	いいえ

名前	説明	必須
`RangeKeyCondition : ComparisonOperator`	指定された属性 (等しい、より大きい、など) を評価するための基準。クエリオペレーションで有効な比較演算子は次の通りです。 Note より大きい、等しい、より小さいに関する文字列値の比較は、ASCII 文字コード値に基づきます。例えば、a は A より大きく、aa は B より大きいと評価されます。コードの値のリストについては、http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters を参照してください。バイナリの場合、DynamoDB は、バイナリ値を比較する際 (クエリ表現の評価など)、バイナリデータの各バイトを符号なしとしてて扱います。 型: 文字列またはバイナリ	いいえ

名前	説明	必須
	EQ: 等しい。 EQ の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と等しくありません。	

名前	説明	必須
	LE: より小さい、または等しい。 LE の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
	LT: より小さい。 LT の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
	GE: より大きい、または等しい。 GE の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
	GT: より大きい。 GT の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	
	BEGINS_WITH : プレフィックスを確認します。 BEGINS_WITH の場合、AttributeValueList は、文字列またはバイナリの 1 つの AttributeValue のみを含むことができます (数値やセットではありません)。比較のターゲット属性は、(数値またはセットではなく) 文字列またはバイナリである必要があります。	

名前	説明	必須
	BETWEEN: 最初の値より大きいまたは等しい、および 2 番目の値より小さいまたは等しい。 BETWEEN の場合、AttributeValueList は、文字列、数値、またはバイナリのいずれかの同じ型の 2 つの AttributeValue 要素を含む必要があります (セットではありません)。ターゲット属性は、ターゲット値が最初の要素より大きいか等しく、2番目の要素より小さいか等しい場合に一致します。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と比較されません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
ScanIndexForward	インデックスの昇順または降順トラバーサルを指定します。DynamoDB は、範囲キーによって決定されたリクエストされた順序を反映した結果を返します。データ型が数値の場合、結果は数値順に返されます。それ以外の場合は、トラバーサルは ASCII 文字コード値に基づきます。 型: ブール値 デフォルトは true (昇順) です。	いいえ
ExclusiveStartKey	以前のクエリを続行する項目のプライマリキー。クエリが完了する前に結果セットサイズまたは Limit パラメータのいずれかが原因でクエリオペレーションが中断した場合、そのクエリで、この値が LastEvaluatedKey として提供されることがあります。新しいクエリリクエストで LastEvaluatedKey を渡し、その時点からオペレーションを続行できます。 型: 複合プライマリキーの HashKeyElement 、または HashKeyElement および RangeKeyElement 。	いいえ

レスポンス

構文

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"N":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue3"},
    "AttributeName2":{"N":"AttributeValue4"},
    "AttributeName3":{"S":"AttributeValue3"},
    "AttributeName5":{"B":"dmFsdWU="}
}],
    "LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},
                        "RangeKeyElement":{"AttributeValue4":"N"}
     },
     "ConsumedCapacityUnits":1
}
```

名前	説明
`Items`	クエリパラメータを満たす項目属性。 型: 属性名およびそのデータ型と値のマップ。
`Count`	レスポンス内の項目数。詳細については、「結果での項目のカウント」を参照してください。 型: 数値
`LastEvaluatedKey`	クエリオペレーションが停止した項目のプライマリキー (以前の結果セットを含みます)。この値を使用して、新しいリクエストでこの値を除く新しいオペレーションをスタートします。

名前	説明
	クエリ結果セット全体が完了したとき (オペレーションが「最後のページ」を処理したとき)、LastEvaluatedKey は null です。 型: 複合プライマリキーの HashKeyElement 、または HashKeyElement および RangeKeyElement 。
ConsumedCapacityUnits	オペレーションによって消費される読み込み容量ユニットの数。この値は、プロビジョニングされたスループットに適用される数を示します。詳細については、「プロビジョンドキャパシティモード」を参照してください。 型: 数値

特殊なエラー

エラー	説明
ResourceNotFoundException	指定されたターゲットが見つかりませんでした。

例

AWS SDK を使用した例については、「DynamoDB のクエリオペレーション」を参照してください。

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0
```

```
{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"John"},
  "RangeKeyElement":{"S":"The Matrix"}
 }
}
```

レスポンス例

```
HTTP/1.1 200
x-amzn-RequestId: 3647e778-71eb-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The End"}
 },{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The Beatles"}
 }],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"John"},"RangeKeyElement":{"S":"The
 Beatles"}},
"ConsumedCapacityUnits":1
}
```

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0
```

```
{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
 "ScanIndexForward":false}
```

レスポンス例

```
HTTP/1.1 200
x-amzn-RequestId: 8b9ee1ad-774c-11e0-9172-d954e38f553a
content-type: application/x-amz-json-1.0
content-length: 119

{"Count":1,"Items":[{
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"},
 "year":{"N":"1980"}
 }],
"ConsumedCapacityUnits":1
}
```

関連アクション

- Scan

Scan

> Important
>
> ***##########API ##### 2011-12-05 ###***
>
> 現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

Scan オペレーションは、テーブルのフルスキャンを実行することにより、1 つ以上の項目とその属性を返します。より詳細な結果を取得するには、ScanFilter を使用します。

Note

スキャンされた項目の合計数が 1 MB の制限を超えると、スキャンが停止し、結果が後続のオペレーションでスキャンを続行するための `LastEvaluatedKey` とともにユーザーに返されます。結果には、制限を超えた項目数も含まれます。スキャンの結果には、フィルタ基準を満たすテーブルデータがないことがあります。
結果セットは結果整合性です。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Limit": 2,
    "ScanFilter":{
        "AttributeName1":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
    },
    "ExclusiveStartKey":{
        "HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"}
    },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

名前	説明	必須
`TableName`	リクエストされた項目を含むテーブルの名前。 型: 文字列	はい

名前	説明	必須
AttributesToGet	属性名の配列。属性名が指定されていない場合、すべての属性が返されます。見つからなかった属性は結果に表示されません。 型: 配列	いいえ
Limit	評価する項目の最大数 (一致する項目の数であるとは限りません)。結果を処理する間に制限までの項目数を処理した場合、DynamoDB は処理を停止し、その時点までの一致する値とともに、後続のオペレーションで項目の取得を続行するために適用する LastEvaluatedKey を返します。また、この制限に達する前に、スキャンされたデータセットのサイズが 1MB を超えた場合、DynamoDB はスキャンを停止し、制限までの一致値とともに、後続のオペレーションでスキャンを続行するために適用する LastEvaluatedKey を返します。 型: 数値	いいえ

名前	説明	必須
`Count`	`true` に設定されている場合、DynamoDB は、割り当てられたフィルターに一致する項目がなくても、スキャンオペレーションの項目の合計数を返します。Limit パラメータは、カウントのみのスキャンに適用できます。 `AttributesToGet` のリストを提供する場合、`Count` を `true` に設定しないでください。そうでないと、DynamoDB は検証エラーを返します。詳細については、「結果での項目のカウント」を参照してください。 型: ブール値	いいえ
`ScanFilter`	スキャン結果を評価し、目的の値のみを返します。複数の条件は「AND」オペレーションとして扱われます。すべての条件を満たす値だけが結果に含まれます。 型: 比較演算子を含む値への属性名のマップ。	いいえ
`ScanFilter :AttributeValueList`	フィルターのスキャン結果を評価するための値と条件。 型: `Condition` への `AttributeValue` のマップ。	いいえ

名前	説明	必須
`ScanFilter : ComparisonOperator`	指定された属性 (等しい、より大きい、など) を評価するための基準。スキャンオペレーションで有効な比較演算子は次の通りです。 Note より大きい、等しい、より小さいに関する文字列値の比較は、ASCII 文字コード値に基づきます。例えば、a は A より大きく、aa は B より大きいと評価されます。コードの値のリストについては、http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters を参照してください。 バイナリの場合、DynamoDB は、バイナリ値を比較する際 (クエリ表現の評価など)、バイナリデータの各バイトを符号なしとしてて扱います。 型: 文字列またはバイナリ	いいえ

名前	説明	必須
	EQ: 等しい。 EQ の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と等しくありません。	
	NE: 等しくない。 NE の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と等しくありません。	

名前	説明	必須
	LE: より小さい、または等しい。 LE の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
	LT: より小さい。 LT の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
	GE: より大きい、または等しい。 GE の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
	GT: より大きい。 GT の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と等しくありません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	
	NOT_NULL : 属性が存在します。	
	NULL: 属性は存在しません。	

名前	説明	必須
	CONTAINS: サブシーケンス、またはセット内の値をチェックします。 CONTAINS の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。比較のターゲット属性が文字列の場合、オペレーションは部分文字列の一致をチェックします。比較のターゲット属性がバイナリの場合、オペレーションは入力に一致するターゲットのサブシーケンスを検索します。比較のターゲット属性がセット (「SS」、「NS」、または「BS」) である場合、オペレーションは、(部分文字列としてではなく) セットの要素をチェックします。	

名前	説明	必須
	NOT_CONTAINS ：サブシーケンスの欠如、またはセット内の値の欠如をチェックします。 NOT_CONTAINS の場合、AttributeValueList は、文字列、数値、またはバイナリのうち、1 つの AttributeValue のみを含むことができます (セットではありません)。比較のターゲット属性が文字列の場合、オペレーションは部分文字列の欠如がないかどうかをチェックします。比較のターゲット属性がバイナリである場合、オペレーションは入力に一致するターゲットのサブシーケンスの欠如がないかどうかをチェックします。比較のターゲット属性がセット (「SS」、「NS」、または「BS」) である場合、オペレーションは、(部分文字列としてではなく) セットの要素の欠如ないかどうかをチェックします。	

名前	説明	必須
	BEGINS_WITH : プレフィックスを確認します。 BEGINS_WITH の場合、AttributeValueList は、文字列またはバイナリの 1 つの AttributeValue のみを含むことができます (数値やセットではありません)。比較のターゲット属性は、(数値またはセットではなく) 文字列またはバイナリである必要があります。	
	IN: 完全一致をチェックします。 IN の場合、AttributeValueList は、文字列、数値、またはバイナリの 1 つ以上の AttributeValue を含むことができます (セットではありません)。比較のターゲット属性は、一致対象と同じ型および同じ値である必要があります。文字列は、文字列セットと一致しません。	

名前	説明	必須
	BETWEEN: 最初の値より大きいまたは等しい、および 2 番目の値より小さいまたは等しい。 BETWEEN の場合、AttributeValueList は、文字列、数値、またはバイナリのいずれかの同じ型の 2 つの AttributeValue 要素を含む必要があります (セットではありません)。ターゲット属性は、ターゲット値が最初の要素より大きいか等しく、2番目の要素より小さいか等しい場合に一致します。リクエストで指定されているもの以外の型の AttributeValue が項目に含まれる場合、値は一致しません。例えば、{"S":"6"} は {"N":"6"} と比較されません。また、{"N":"6"} は {"NS":["6", "2", "1"]} と比較されません。	

名前	説明	必須
ExclusiveStartKey	以前のスキャンを続行する項目のプライマリキー。以前のスキャンでは、テーブル全体をスキャンする前にスキャンオペレーションが中断された場合、結果セットのサイズまたは Limit パラメータのいずれかにより、この値が提供されることがあります。新しいスキャンリクエストで LastEvaluatedKey を渡し、その時点からオペレーションを続行できます。 型: 複合プライマリキーの HashKeyElement 、または HashKeyElement および RangeKeyElement 。	いいえ

レスポンス

構文

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 229

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue4"},
    "AttributeName2":{"S":"AttributeValue5"},
    "AttributeName3":{"S":"AttributeValue6"},
    "AttributeName5":{"B":"dmFsdWU="}
    }],
    "LastEvaluatedKey":
        {"HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"},
    "ConsumedCapacityUnits":1,
    "ScannedCount":2}
}
```

名前	説明
Items	オペレーションパラメータを満たす属性のコンテナ。 型: 属性名およびそのデータ型と値のマップ。
Count	レスポンス内の項目数。詳細については、「結果での項目のカウント」を参照してください。 型: 数値
ScannedCount	フィルターが適用されるまでの完全なスキャン内の項目の数。大きい ScannedCount 値がほとんどない (またはない) 場合、Count の結果は、不十分なスキャンオペレーションを示します。詳細については、「結果での項目のカウント」を参照してください。 型: 数値
LastEvaluatedKey	スキャンオペレーションが停止した項目のプライマリキー。その時点からオペレーションを続行するには、後続のスキャンオペレーションでこの値を指定します。 スキャン結果セット全体が完了したとき (オペレーションが「最後のページ」を処理したとき)、LastEvaluatedKey は null です。
ConsumedCapacityUnits	オペレーションによって消費される読み込み容量ユニットの数。この値は、プロビジョニング

名前	説明
	されたスループットに適用される数を示します。詳細については、「プロビジョンドキャパシティモード」を参照してください。 型: 数値

特殊なエラー

エラー	説明
ResourceNotFoundException	指定されたターゲットが見つかりませんでした。

例

AWS SDK を使用した例については、「DynamoDB でのスキャンの使用」を参照してください。

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

レスポンス例

```
HTTP/1.1 200
x-amzn-RequestId: 4e8a5fa9-71e7-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 465

{"Count":4,"Items":[{
 "date":{"S":"1980"},
 "fans":{"SS":["Dave","Aaron"]},
"name":{"S":"Airplane"},
"rating":{"S":"***"}
},{
"date":{"S":"1999"},
"fans":{"SS":["Ziggy","Laura","Dean"]},
"name":{"S":"Matrix"},
"rating":{"S":"*****"}
},{
"date":{"S":"1976"},
"fans":{"SS":["Riley"]},"
name":{"S":"The Shaggy D.A."},
"rating":{"S":"**"}
},{
"date":{"S":"1985"},
"fans":{"SS":["Fox","Lloyd"]},
"name":{"S":"Back To The Future"},
"rating":{"S":"****"}
}],
   "ConsumedCapacityUnits":0.5
"ScannedCount":4}
```

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 }
}
```

レスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
```

```
content-length: 262
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":2,
 "Items":[
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Casey"}},
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Fredy"}
  }],
"ConsumedCapacityUnits":0.5
"ScannedCount":4
}
```

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 },
 "ExclusiveStartKey":
  {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
}
```

レスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 232
```

```
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":1,
 "Items":[
  {"friends":{"SS":["Jane","James","John"]},
  "status":{"S":"exercising"},
  "time":{"N":"2200"},
  "user":{"S":"Roger"}}
 ],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"250"}},
"ConsumedCapacityUnits":0.5
"ScannedCount":2
}
```

関連アクション

- Query
- BatchGetItem

UpdateItem

> Important
>
> ***##########API ##### 2011-12-05 ######################################
> #######***
> 現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

既存の項目の属性を編集します。条件付き更新を実行できます (属性名と値のペアが存在しない場合は新しい属性名と値のペアを挿入し、予期される属性値が存在する場合は既存の名前と値のペアを置き換えます)。

> Note
>
> UpdateItem を使用してプライマリキー属性を更新することはできません。その場合、項目を削除し、PutItem を使用して新しい属性で新しい項目を作成します。

UpdateItem オペレーションには、更新の実行方法を定義する `Action` パラメータが含まれます。属性値を配置、削除、または追加できます。

属性値は NULL であってはなりません。文字列型およびバイナリ型属性の長さは 0 より大きくなければなりません。また、セット型属性は空であってはなりません。空の値を持つリクエストは、`ValidationException` で拒否されます。

指定したプライマリキーが既存の項目に存在する場合:

- PUT - 指定した属性を追加します。属性が存在する場合は、新しい値で置き換えられます。
- DELETE - 値を指定しない場合、属性とその値が削除されます。値のセットが指定されている場合、指定されたセット内の値が古いセットから削除されます。したがって、属性値に [a,b,c] が含まれ、削除アクションに [a,c] が含まれる場合、最終的な属性値は [b] です。指定した値の型は、既存の値の型と一致する必要があります。空のセットの指定は無効です。
- ADD — 数値の追加アクションの場合、またはターゲット属性がセット (文字列セットを含む) の場合にのみ使用します。ターゲット属性が単一の文字列値またはスカラーバイナリ値の場合、ADD は機能しません。指定された値は、数値に追加されるか (既存の数値のインクリメントまたはデクリメント)、文字列セット内の追加値として追加されます。値のセットが指定されている場合、値は既存のセットに追加されます。例えば、元のセットが [1,2] で、指定された値が [3] の場合、追加オペレーション後のセットは [4,5] ではなく [1,2,3] になります。セット属性に対して Add アクションが指定されていて、指定した属性の型が既存のセットの型と一致しない場合、エラーが発生します。

  存在しない属性に対して ADD を使用すると、属性とその値が項目に追加されます。

指定したプライマリキーに一致する項目がない場合:

- PUT - 指定されたプライマリキーで新しい項目を作成します。その後、指定された属性を追加します。
- DELETE - 何の処理も行われません。
- ADD — 属性値に対して指定されたプライマリキーと数値 (または数値のセット) で項目を作成します。文字列またはバイナリ型の場合は無効です。

> Note
>
> `ADD` を使用して、更新の前に存在しない項目の数値をインクリメントまたはデクリメントした場合、DynamoDB は初期値として `0` を使用します。また、`ADD` を使用して項目を更新

し、更新の前に (項目は存在しても) 存在しない属性の数値をインクリメントまたはデクリメントした場合、DynamoDB は初期値として 0 を使用します。例えば、ADD を使用して、更新の前に存在しなかった属性に +3 を追加するとします。DynamoDB は初期値に 0 を使用します。更新後の値は 3 です。

このオペレーションの使用に関する詳細については、項目と属性の操作 を参照してください。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},
        "RangeKeyElement":{"N":"AttributeValue2"}},
    "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
    "ReturnValues":"ReturnValuesConstant"
}
```

名前	説明	必須
TableName	更新する項目を含めるテーブルの名前。 型: 文字列	はい
Key	項目を定義するプライマリキー。プライマリキーの詳細については、「プライマリキー」を参照してください。	はい

名前	説明	必須
	型: `HashKeyElement` の値に対するマップと `RangeKeyElement` の値に対するマップ。	
`AttributeUpdates`	更新のための値、およびアクションに対する属性名マップ。属性名は変更する属性を指定します。プライマリキー属性を含めることはできません。 型: 属性更新のための属性名、値、およびアクション。	
`Attribute Updates :Action`	更新の実行方法を指定します。使用できる値: PUT (デフォルト)、ADD、または DELETE。セマンティクスについては、UpdateItem の説明を参照してください。 型: 文字列 デフォルト: PUT	いいえ
`Expected`	条件付き更新の属性を指定します。`Expected` パラメータを使用すると、属性名を指定できます。また、DynamoDB で更新を行う前に、属性値が既に存在するかどうか、属性値が存在する場合は特定の値があるかどうかの確認の有無も指定できます。 型: 属性名のマップ。	いいえ

名前	説明	必須
`Expected:AttributeName`	条件付き配置の属性の名前。 型: 文字列	いいえ

名前	説明	必須

`Expected:AttributeName: ExpectedAttributeValue`	このパラメータを使用して、属性名と値のペアに値が既に存在するかどうかを指定します。
	次の JSON 表記法では、項目に「Color」属性がまだ存在しない場合、その項目が更新されます。
	`"Expected" : {"Color":{"Exists":false}}`
	次の JSON 表記法では、項目を更新する前に、「Color」という名前の属性の既存の値が「Yellow」であるかどうかがチェックされます。
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	デフォルトでは、`Expected` パラメータを使用し、`Value` を指定した場合、DynamoDB は属性が存在し、置換する現在の値があると仮定します。したがって、`{"Exists":true}` は暗示されているので指定する必要はありません。リクエストを短縮するには、次のようにします。
		いいえ

名前	説明	必須
	`"Expected" : {"Color":{"Value": {"S":"Yellow"}}}` Note チェックする属性値なしで `{"Exists" :true}` を指定すると、DynamoDB はエラーを返します。	

名前	説明	必須
ReturnValues	UpdateItem リクエストで属性名と値のペアを更新する前に属性名と値のペアを取得する場合、このパラメータを使用します。使用できるパラメータの値は、NONE (デフォルト) または ALL_OLD、UPDATED_OLD 、ALL_NEW、または UPDATED_NEW です。ALL_OLD を指定し、UpdateItem によって属性名と値のペアが上書きされた場合、古い項目の内容が返されます。このパラメータを指定しない場合、または NONE の場合、何も返されません。ALL_NEW が指定されている場合、新しいバージョンの項目のすべての属性が返されます。UPDATED_NEW が指定されている場合、更新された属性の新しいバージョンのみが返されます。 型: 文字列	いいえ

レスポンス

構文

次の構文例では、ALL_OLD の ReturnValues パラメータが指定されたリクエストを引き受けます。指定されていない場合、レスポンスには ConsumedCapacityUnits 要素のみが含まれます。

```
HTTP/1.1 200
```

```
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 140

{"Attributes":{
 "AttributeName1":{"S":"AttributeValue1"},
 "AttributeName2":{"S":"AttributeValue2"},
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits":1
}
```

名前	説明
Attributes	属性名と値のペアのマップ (ReturnValues パラメータがリクエストの NONE 以外のものとして指定されている場合)。 型: 属性の名前と値のペアのマップ。
ConsumedCapacityUnits	オペレーションによって消費される書き込み容量ユニットの数。この値は、プロビジョニングされたスループットに適用される数を示します。詳細については、「プロビジョンドキャパシティモード」を参照してください。 型: 数値

特殊なエラー

エラー	説明
ConditionalCheckFailedException	条件チェックに失敗しました。属性 ("+ name +") の値は ("+ name +") ですが、("+ expValue +") が予期されていました。

エラー	説明
ResourceNotFoundExceptions	指定された項目または属性が見つかりませんでした。

例

AWS SDK を使用した例については、「項目と属性の操作」を参照してください。

リクエスト例

```
// This header is abbreviated. For a sample of a complete header, see DynamoDB ####
 API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
    "Key":
        {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
    "AttributeUpdates":
        {"status":{"Value":{"S":"online"},
        "Action":"PUT"}},
    "Expected":{"status":{"Value":{"S":"offline"}}},
    "ReturnValues":"ALL_NEW"
}
```

レスポンス例

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMHO7F01Q9P7Q6QMKMMI3R3QRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 121
Date: Fri, 26 Aug 2011 21:05:00 GMT

{"Attributes":
    {"friends":{"SS":["Lynda, Aaron"]},
    "status":{"S":"online"},
    "time":{"N":"1307654350"},
    "user":{"S":"Julie"}},
"ConsumedCapacityUnits":1
}
```

関連アクション

- PutItem
- DeleteItem

UpdateTable

> Important
>
> ##########API ##### 2011-12-05 ##
> #######
>
> 現在の低レベルの API に関するドキュメントについては、Amazon DynamoDB API リファレンスを参照してください。

説明

指定されたテーブルのプロビジョンされたスループットを更新します。テーブルのスループットを設定すると、パフォーマンスの管理に役立ちます。これは、DynamoDB のプロビジョニングされたスループット機能の一部です。詳細については、「プロビジョンドキャパシティモード」を参照してください。

プロビジョニングされたスループット値は、Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ にリストされている最大値と最小値に基づいてアップグレードまたはダウングレードできます。

このオペレーションが正常に完了するには、テーブルが ACTIVE の状態である必要があります。UpdateTable は非同期オペレーションです。オペレーションの実行中、テーブルは UPDATING の状態です。テーブルが UPDATING の状態のとき、テーブルのプロビジョニングされたスループットは呼び出しの前のものです。新しいプロビジョニングされたスループット設定は、UpdateTable オペレーションの後にテーブルが ACTIVE の状態に戻ったときにのみ有効になります。

リクエスト

構文

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

名前	説明	必須
TableName	更新するテーブルの名前です。 型: 文字列	はい
ProvisionedThroughput	指定したテーブルの新しいスループット (ReadCapacityUnits と WriteCapacityUnits の値で構成されます)。「プロビジョンドキャパシティモード」を参照してください。 型: 配列	はい
ProvisionedThroughput :ReadCapacityUnits	DynamoDB がロードとその他のオペレーションのバランスを取る前に、指定したテーブルで 1 秒あたりに消費される整合性のある ReadCapacityUnits の最小数を設定します。 結果整合性のある読み込みオペレーションに必要な労力は、整合性のある読み込みオペレーションより少な	はい

名前	説明	必須
	いので、1 秒あたり 50 の整合性のある ReadCapacityUnits の設定では、1 秒あたり 100 の結果整合性のある ReadCapacityUnits が実現します。 型: 数値	
ProvisionedThroughput :WriteCapacityUnits	DynamoDB がロードとその他のオペレーションのバランスを取る前に、指定したテーブルで 1 秒あたりに消費される整合性のある WriteCapacityUnits の最小数を設定します 型: 数値	はい

レスポンス

構文

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/json
Content-Length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeValue1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeValue2","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"UPDATING"}}
```

名前	説明
CreationDateTime	テーブルの作成日。 型: 数値
KeySchema	テーブルのプライマリキー (シンプルまたは複合) 構造。HashKeyElement の名前と値のペアは必須です。RangeKeyElement の名前と値のペアはオプションです (複合プライマリキーにのみ必要)。ハッシュキーの最大サイズは 2048 バイトです。範囲キーの最大サイズは 1024 バイトです。両方の制限は別々に強制されます (ハッシュと範囲の合計で 2048 + 1024 キーを組み合わせることができます)。プライマリキーの詳細については、「プライマリキー」を参照してください。 型: 複合プライマリキーの HashKeyElement 、または HashKeyElement および RangeKeyElement のマップ。
ProvisionedThroughput	指定されたテーブルの現在のスループット設定。LastIncreaseDateTime の値 (該当する場合)、LastDecreaseDateTime (該当する場合) を含みます。 型: 配列
TableName	更新されたテーブルの名前。 型: 文字列

名前	説明
TableStatus	テーブルの現在の状態 (CREATING、ACTIVE、DELETING、または UPDATING)。UPDATING である必要があります。 テーブルのステータスをチェックするには、DescribeTables オペレーションを使用します。 型: 文字列

特殊なエラー

エラー	説明
ResourceNotFoundException	指定されたターゲットが見つかりませんでした。
ResourceInUseException	テーブルは ACTIVE の状態ではありません。

例

リクエスト例

```
// This header is abbreviated.
// For a sample of a complete header, see DynamoDB #### API.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

レスポンス例

```
HTTP/1.1 200 OK
content-type: application/x-amz-json-1.0
content-length: 390
Date: Sat, 19 Nov 2011 00:46:47 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"comp1",
    "TableStatus":"UPDATING"}
}
```

関連アクション

- CreateTable
- DescribeTables
- DeleteTable

AWS SDK for Java 1.x の例

このセクションでは、SDK for Java 1.x を使用した DAX アプリケーションのコード例について説明します。

トピック

- DAX を AWS SDK for Java 1.x で使用する
- 既存の SDK for Java 1.x アプリケーションを DAX を使用するように変更する
- SDK for Java 1.x を使用したグローバルセカンダリインデックスのクエリ

DAX を AWS SDK for Java 1.x で使用する

この手順に従って、Amazon EC2 インスタンスで Amazon DynamoDB Accelerator (DAX) の Java サンプルを実行します。

> Note
>
> これらの手順は、AWS SDK for Java 1.x を使用するアプリケーション向けです。AWS SDK for Java 2.x を使用するアプリケーションの場合は、「Java および DAX」を参照してください。

DAX の Java サンプルを実行するには

1. Java 開発キット (JDK) をインストールします。

```
sudo yum install -y java-devel
```

2. AWS SDK for Java (`.zip` ファイル) をダウンロードして解凍します。

```
wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

unzip aws-java-sdk.zip
```

3. DAX Java クライアント (`.jar` ファイル) の最新バージョンをダウンロードします。

```
wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
latest.jar
```

> Note
>
> DAX SDK for Java のクライアントは、Apache Maven で利用可能です。詳細については、「クライアントを Apache Maven 依存関係として使用する」を参照してください。

4. `CLASSPATH` 変数を設定します。この例では、*`sdkVersion`* を AWS SDK for Java の実際のバージョン番号に置き換えます (例: `1.11.112`)。

```
export SDKVERSION=sdkVersion
export CLASSPATH=$(pwd)/TryDax/java:$(pwd)/DaxJavaClient-latest.jar:$(pwd)/
aws-java-sdk-$SDKVERSION/lib/aws-java-sdk-$SDKVERSION.jar:$(pwd)/aws-java-sdk-
$SDKVERSION/third-party/lib/*
```

5. サンプルプログラムソースコード (`.zip` ファイル) をダウンロードします。

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   ダウンロードが完了したら、ソースファイルを解凍します。

   ```
   unzip TryDax.zip
   ```

6. Java コードディレクトリに移動し、次のようにコードをコンパイルします。

   ```
   cd TryDax/java/
   javac TryDax*.java
   ```

7. プログラムを実行します。

   ```
   java TryDax
   ```

   次のような出力が表示されます。

   ```
   Creating a DynamoDB client

   Attempting to create table; please wait...
   Successfully created table.  Table status: ACTIVE
   Writing data to the table...
   Writing 10 items for partition key: 1
   Writing 10 items for partition key: 2
   Writing 10 items for partition key: 3
   Writing 10 items for partition key: 4
   Writing 10 items for partition key: 5
   Writing 10 items for partition key: 6
   Writing 10 items for partition key: 7
   Writing 10 items for partition key: 8
   Writing 10 items for partition key: 9
   Writing 10 items for partition key: 10

   Running GetItem, Scan, and Query tests...
   First iteration of each test will result in cache misses
   ```

```
Next iterations are cache hits

GetItem test - partition key 1 and sort keys 1-10
 Total time: 136.681 ms - Avg time: 13.668 ms
 Total time: 122.632 ms - Avg time: 12.263 ms
 Total time: 167.762 ms - Avg time: 16.776 ms
 Total time: 108.130 ms - Avg time: 10.813 ms
 Total time: 137.890 ms - Avg time: 13.789 ms
Query test - partition key 5 and sort keys between 2 and 9
 Total time: 13.560 ms - Avg time: 2.712 ms
 Total time: 11.339 ms - Avg time: 2.268 ms
 Total time: 7.809 ms - Avg time: 1.562 ms
 Total time: 10.736 ms - Avg time: 2.147 ms
 Total time: 12.122 ms - Avg time: 2.424 ms
Scan test - all items in the table
 Total time: 58.952 ms - Avg time: 11.790 ms
 Total time: 25.507 ms - Avg time: 5.101 ms
 Total time: 37.660 ms - Avg time: 7.532 ms
 Total time: 26.781 ms - Avg time: 5.356 ms
 Total time: 46.076 ms - Avg time: 9.215 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

タイミング情報を書き留めます。これは `GetItem`、`Query`、`Scan` テストに必要なミリ秒の数字です。

8. 前のステップで、DynamoDB エンドポイントに対してプログラムを実行しました。ここでプログラムを再度実行しますが、今度は `GetItem`、`Query`、`Scan` オペレーションが DAX クラスターによって処理されます。

   DAX クラスターのエンドポイントを確認するには、次のいずれかを選択します。

   - DynamoDB コンソールの使用 — DAX クラスターを選択します。次の例のように、クラスターエンドポイントがコンソールに表示されます。

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - AWS CLI の使用 — 次のコマンドを入力します。

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

次の例のように、クラスターエンドポイントが出力に表示されます。

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

ここでプログラムを再度実行しますが、今度はクラスターエンドポイントをコマンドラインパラメータとして指定します。

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

出力の残りの部分を見て、タイミング情報を書き留めます。GetItem、Query および Scan の経過時間は、DynamoDB を使用した場合より DAX を使用した方が大幅に低いはずです。

このプログラムの詳細については、以下のセクションを参照してください。

- TryDax.java
- TryDaxHelper.java
- TryDaxTests.java

クライアントを Apache Maven 依存関係として使用する

これらのステップに従って、アプリケーションで DAX SDK for Java のクライアントを依存関係として使用します。

クライアントを Maven 依存関係として使用するには

1. Apache Maven をダウンロードし、インストールします。詳細については、「Apache Maven のダウンロード」および「Apache Maven のインストール」を参照してください。
2. Maven 依存関係クライアントをアプリケーションのプロジェクトオブジェクトモデル (POM) ファイルに追加します。この例では、x.x.x.x をクライアントの実際のバージョン番号に置き換えます (例: 1.0.200704.0)。

```
<!--Dependency:-->
```

```
<dependencies>
    <dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>amazon-dax-client</artifactId>
     <version>x.x.x.x</version>
    </dependency>
</dependencies>
```

TryDax.java

`TryDax.java` ファイルには、`main` メソッドが含まれています。コマンドラインパラメータを指定しないでプログラムを実行すると、Amazon DynamoDB クライアントが作成され、すべての API オペレーションでそのクライアントが使用されます。コマンドラインで DynamoDB Accelerator (DAX) クラスターエンドポイントを指定すると、プログラムは DAX クライアントも作成し、`GetItem`、`Query`、`Scan` オペレーションでこれを使用します。

さまざまな方法でプログラムを変更できます。

- DynamoDB クライアントではなく、DAX クライアントを使用します。詳細については、「Java および DAX」を参照してください。
- テストテーブルに別の名前を付けます。
- `helper.writeData` パラメータを変更して書き込まれる項目の数を変更する。2 番目のパラメータはパーティションキーの数、3 番目のパラメータはソートキーの数です。デフォルトでは、プログラムはパーティションキーの値に 1～10 を使用し、ソートキーの値に 1～10 を使用し、合計で 100 個の項目がテーブルに書き込まれます。詳細については、「TryDaxHelper.java」を参照してください。
- `GetItem`、`Query` および `Scan` テストの数を変更し、そのパラメータを変更する。
- `helper.createTable` および `helper.deleteTable` を含むラインをコメントアウトする (プログラムを実行するたびにテーブルを作成および削除しない場合)。

Note

このプログラムを実行するには、DAX SDK for Java のクライアント、および AWS SDK for Java を依存関係として使用するように Maven を設定できます。詳細については、「クライアントを Apache Maven 依存関係として使用する」を参照してください。

また、DAX Java クライアントと AWS SDK for Java の両方をダウンロードして、クラスパスに含めることもできます。CLASSPATH 変数の設定例については、「Java および DAX」を参照してください。

```
public class TryDax {

    public static void main(String[] args) throws Exception {

        TryDaxHelper helper = new TryDaxHelper();
        TryDaxTests tests = new TryDaxTests();

        DynamoDB ddbClient = helper.getDynamoDBClient();
        DynamoDB daxClient = null;
        if (args.length >= 1) {
            daxClient = helper.getDaxClient(args[0]);
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        helper.createTable(tableName, ddbClient);
        System.out.println("Populating table...");
        helper.writeData(tableName, ddbClient, 10, 10);

        DynamoDB testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem, Scan, and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        tests.getItemTest(tableName, testClient, 1, 10, 5);

        // Query
        tests.queryTest(tableName, testClient, 5, 2, 9, 5);
        // Scan
        tests.scanTest(tableName, testClient, 5);

        helper.deleteTable(tableName, ddbClient);
    }

}
```

TryDaxHelper.java

TryDaxHelper.java ファイルにはユーティリティメソッドが含まれています。

getDynamoDBClient および getDaxClient メソッドは、Amazon DynamoDB および Amazon DynamoDB Accelerator (DAX) クライアントを提供します。コントロールプレーンオペレーション (CreateTable、DeleteTable) および書き込みオペレーションでは、プログラムは DynamoDB クライアントを使用します。DAX クラスターエンドポイントを指定すると、メインプログラムは読み込みオペレーション (GetItem、Query、Scan) 用に DAX クライアントを作成します。

他の TryDaxHelper メソッド (createTable、writeData、deleteTable) は、DynamoDB テーブルとそのデータのセットアップおよび解体用です。

さまざまな方法でプログラムを変更できます。

- テーブルで別のプロビジョニングされたスループット設定を使用する。
- 書き込まれた各項目のサイズを変更する (writeData メソッドの stringSize 変数を参照してください)。
- GetItem、Query、および Scan テストの数とそのパラメータを変更する。
- helper.CreateTable および helper.DeleteTable を含むラインをコメントアウトする (プログラムを実行するたびにテーブルを作成および削除しない場合)。

> Note
>
> このプログラムを実行するには、DAX SDK for Java のクライアント、および AWS SDK for Java を依存関係として使用するように Maven を設定できます。詳細については、「クライアントを Apache Maven 依存関係として使用する」を参照してください。

または、DAX Java クライアントと AWS SDK for Java の両方をダウンロードして、クラスパスに含めることもできます。CLASSPATH 変数の設定例については、「Java および DAX」を参照してください。

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.util.EC2MetadataUtils;

public class TryDaxHelper {

    private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

    DynamoDB getDynamoDBClient() {
        System.out.println("Creating a DynamoDB client");
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
                .withRegion(region)
                .build();
        return new DynamoDB(client);
    }

    DynamoDB getDaxClient(String daxEndpoint) {
        System.out.println("Creating a DAX client with cluster endpoint " +
 daxEndpoint);
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
        AmazonDynamoDB client = daxClientBuilder.build();
        return new DynamoDB(client);
    }

    void createTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("Attempting to create table; please wait...");

            table = client.createTable(tableName,
                    Arrays.asList(
                            new KeySchemaElement("pk", KeyType.HASH), // Partition key
                            new KeySchemaElement("sk", KeyType.RANGE)), // Sort key
                    Arrays.asList(
                            new AttributeDefinition("pk", ScalarAttributeType.N),
                            new AttributeDefinition("sk", ScalarAttributeType.N)),
                    new ProvisionedThroughput(10L, 10L));
            table.waitForActive();
            System.out.println("Successfully created table.  Table status: " +
                    table.getDescription().getTableStatus());

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    void writeData(String tableName, DynamoDB client, int pkmax, int skmax) {
        Table table = client.getTable(tableName);
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (Integer ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
 ipk));
                for (Integer isk = 1; isk <= skmax; isk++) {
                    table.putItem(new Item()
                            .withPrimaryKey("pk", ipk, "sk", isk)
                            .withString("someData", someData));
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    void deleteTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            table.delete();
            table.waitForDelete();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

}
```

TryDaxTests.java

`TryDaxTests.java` ファイルには、Amazon DynamoDB のテストテーブルに対して読み込みオペレーションを実行するメソッドが含まれています。これらのメソッドはデータへのアクセス方法 (DynamoDB クライアントを使用するか DAX クライアントを使用するか) は問わないため、アプリケーションロジックを変更する必要はありません。

さまざまな方法でプログラムを変更できます。

- `queryTest` メソッドを変更して異なる `KeyConditionExpression` を使用する。
- `ScanFilter` を `scanTest` メソッドに追加し、項目の一部のみが返されるようにします。

> **Note**
>
> このプログラムを実行するには、DAX SDK for Java のクライアント、および AWS SDK for Java を依存関係として使用するように Maven を設定できます。詳細については、「クライアントを Apache Maven 依存関係として使用する」を参照してください。

または、DAX Java クライアントと AWS SDK for Java の両方をダウンロードして、クラスパスに含めることもできます。CLASSPATH 変数の設定例については、「Java および DAX」を参照してください。

```
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;

public class TryDaxTests {

    void getItemTest(String tableName, DynamoDB client, int pk, int sk, int iterations) {
        long startTime, endTime;
        System.out.println("GetItem test - partition key " + pk + " and sort keys 1-" + sk);
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (Integer ipk = 1; ipk <= pk; ipk++) {
                    for (Integer isk = 1; isk <= sk; isk++) {
                        table.getItem("pk", ipk, "sk", isk);
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }
    void queryTest(String tableName, DynamoDB client, int pk, int sk1, int sk2, int
 iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key " + pk + " and sort keys between
 " + sk1 + " and " + sk2);
        Table table = client.getTable(tableName);

        HashMap<String, Object> valueMap = new HashMap<String, Object>();
        valueMap.put(":pkval", pk);
        valueMap.put(":skval1", sk1);
        valueMap.put(":skval2", sk2);

        QuerySpec spec = new QuerySpec()
                .withKeyConditionExpression("pk = :pkval and sk between :skval1
 and :skval2")
                .withValueMap(valueMap);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<QueryOutcome> items = table.query(spec);

            try {
                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to query table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    void scanTest(String tableName, DynamoDB client, int iterations) {
        long startTime, endTime;
        System.out.println("Scan test - all items in the table");
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<ScanOutcome> items = table.scan();
            try {
                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to scan table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    public void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

既存の SDK for Java 1.x アプリケーションを DAX を使用するように変更する

Amazon DynamoDB を使用している Java アプリケーションがすでにある場合、DynamoDB Accelerator (DAX) クラスターにアクセスできるように変更する必要があります。DAX Java クライアントは、AWS SDK for Java に含まれる DynamoDB 低レベルクライアントとよく似ているため、アプリケーション全体を書き換える必要はありません。

> **Note**
>
> これらの手順は、AWS SDK for Java 1.x を使用するアプリケーション向けです。AWS SDK for Java 2.x を使用するアプリケーションの場合は、「既存のアプリケーションを DAX を使用するように変更する」を参照してください。

Music という名前の DynamoDB テーブルがあるとします。テーブルのパーティションキーは Artist で、ソートキーは SongTitle です。次のプログラムは、Music テーブルから直接項目を読み込みます。

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

        // Create a DynamoDB client
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

        HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
        key.put("Artist", new AttributeValue().withS("No One You Know"));
        key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

        GetItemRequest request = new GetItemRequest()
            .withTableName("Music").withKey(key);

        try {
            System.out.println("Attempting to read the item...");
            GetItemResult result = client.getItem(request);
            System.out.println("GetItem succeeded: " + result);

        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }
    }
}
```

プログラムを変更するには、DynamoDB クライアントを DAX クライアントに置き換えます。

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;
```

```
public class GetMusicItem {

    public static void main(String[] args) throws Exception {

    //Create a DAX client

    AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
    daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
    AmazonDynamoDB client = daxClientBuilder.build();


       /*
       ** ...
       ** Remaining code omitted (it is identical)
       ** ...
       */

    }
}
```

DynamoDB ドキュメント API を使用する

AWS SDK for Java は DynamoDB 向けのドキュメントインターフェイスを提供します。ドキュメント API は、低レベル DynamoDB クライアントのラッパーとして機能します。詳細については、「ドキュメントインターフェイス」を参照してください。

ドキュメントインターフェイスは、次の例に示すように、低レベル DAX クライアントとともに使用することもできます。

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class GetMusicItemWithDocumentApi {

    public static void main(String[] args) throws Exception {

        //Create a DAX client
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
        AmazonDynamoDB client = daxClientBuilder.build();

        // Document client wrapper
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");

        try {
            System.out.println("Attempting to read the item...");
            GetItemOutcome outcome = table.tgetItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Scared of My Shadow");
            System.out.println(outcome.getItem());
            System.out.println("GetItem succeeded: " + outcome);
        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }

    }
}
```

DAX 非同期クライアント

AmazonDaxClient は同期します。大きいテーブルの Scan のように長期間実行する DAX API オペレーションでは、そのオペレーションが完了するまでプログラムの実行がブロックされることがあります。DAX API オペレーションの処理中にプログラムで他の作業を実行する必要がある場合は、代わりに ClusterDaxAsyncClient を使用できます。

次のプログラムは、ClusterDaxAsyncClient を Java Future と併せて使用し、ノンブロッキングソリューションを実装する方法を示します。

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

import com.amazon.dax.client.dynamodbv2.ClientConfig;
import com.amazon.dax.client.dynamodbv2.ClusterDaxAsyncClient;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.handlers.AsyncHandler;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBAsync;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class DaxAsyncClientDemo {
 public static void main(String[] args) throws Exception {

  ClientConfig daxConfig = new ClientConfig().withCredentialsProvider(new
 ProfileCredentialsProvider())
    .withEndpoints("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111");

  AmazonDynamoDBAsync client = new ClusterDaxAsyncClient(daxConfig);

  HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
  key.put("Artist", new AttributeValue().withS("No One You Know"));
  key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

  GetItemRequest request = new GetItemRequest()
    .withTableName("Music").withKey(key);

  // Java Futures
  Future<GetItemResult> call = client.getItemAsync(request);
  while (!call.isDone()) {
   // Do other processing while you're waiting for the response
   System.out.println("Doing something else for a few seconds...");
   Thread.sleep(3000);
  }
  // The results should be ready by now

  try {
   call.get();

  } catch (ExecutionException ee) {
   // Futures always wrap errors as an ExecutionException.
   // The *real* exception is stored as the cause of the
   // ExecutionException
   Throwable exception = ee.getCause();
   System.out.println("Error getting item: " + exception.getMessage());
  }

  // Async callbacks
  call = client.getItemAsync(request, new AsyncHandler<GetItemRequest, GetItemResult>()
 {
   @Override
   public void onSuccess(GetItemRequest request, GetItemResult getItemResult) {
    System.out.println("Result: " + getItemResult);
   }

   @Override
   public void onError(Exception e) {
    System.out.println("Unable to read item");
    System.err.println(e.getMessage());
    // Callers can also test if exception is an instance of
    // AmazonServiceException or AmazonClientException and cast
    // it to get additional information
   }

  });
  call.get();

 }
}
```

SDK for Java 1.x を使用したグローバルセカンダリインデックスのクエリ

Amazon DynamoDB Accelerator (DAX) を使用して、DynamoDB プログラムインターフェイスを使用してグローバルセカンダリインデックスをクエリできます。

次の例は、DAX を使用して、「例: AWS SDK for Java ドキュメント API を使用したグローバルセカンダリインデックス」で作成した CreateDateIndex グローバルセカンダリインデックスをクエリする方法を示します。

DAXClient クラスは、DynamoDB プログラミングインターフェイスを操作するために必要なクライアントオブジェクトをインスタンス化します。

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;

public class DaxClient {

 private static final String region = EC2MetadataUtils.getEC2InstanceRegion();
 DynamoDB getDaxDocClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDB(client);
 }

 DynamoDBMapper getDaxMapperClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDBMapper(client);
 }
}
```

グローバルセカンダリインデックスは、以下の方法でクエリできます。

- 以下の例で定義された QueryIndexDax クラスの queryIndex メソッドを使用します。QueryIndexDax は、DaxClient クラスの getDaxDocClient メソッドによって返されるクライアントオブジェクトをパラメータとして取ります。
- オブジェクト永続性インターフェイスを使用する場合は、以下の例で定義された queryIndexMapper クラスで QueryIndexDax メソッドを使用します。queryIndexMapper は、DaxClient クラスで定義された getDaxMapperClient メソッドによって返されるクライアントオブジェクトをパラメータとして取ります。

```
import java.util.Iterator;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import java.util.List;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBQueryExpression;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import java.util.HashMap;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;

public class QueryIndexDax {

 //This is used to query Index using the low-level interface.
 public static void queryIndex(DynamoDB client, String tableName, String indexName) {
  Table table = client.getTable(tableName);

  System.out.println("\n***********************************************************
\n");
  System.out.print("Querying index " + indexName + "...");

  Index index = table.getIndex(indexName);

  ItemCollection<QueryOutcome> items = null;

  QuerySpec querySpec = new QuerySpec();

  if (indexName == "CreateDateIndex") {
   System.out.println("Issues filed on 2013-11-01");
   querySpec.withKeyConditionExpression("CreateDate = :v_date and
 begins_with(IssueId, :v_issue)")
     .withValueMap(new ValueMap().withString(":v_date",
 "2013-11-01").withString(":v_issue", "A-"));
   items = index.query(querySpec);
  } else {
   System.out.println("\nNo valid index name provided");
   return;
  }

  Iterator<Item> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
   System.out.println(iterator.next().toJSONPretty());
  }

 }

 //This is used to query Index using the high-level mapper interface.
 public static void queryIndexMapper(DynamoDBMapper mapper, String tableName, String
 indexName) {
  HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
  eav.put(":v_date", new AttributeValue().withS("2013-11-01"));
  eav.put(":v_issue", new AttributeValue().withS("A-"));
  DynamoDBQueryExpression<CreateDate> queryExpression = new
 DynamoDBQueryExpression<CreateDate>()
    .withIndexName("CreateDateIndex").withConsistentRead(false)
    .withKeyConditionExpression("CreateDate = :v_date and
 begins_with(IssueId, :v_issue)")
    .withExpressionAttributeValues(eav);

  List<CreateDate> items = mapper.query(CreateDate.class, queryExpression);
  Iterator<CreateDate> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
   CreateDate iterObj = iterator.next();
   System.out.println(iterObj.getCreateDate());
   System.out.println(iterObj.getIssueId());
  }
 }
}
```

以下の定義は、「Issues」テーブルを表しており、queryIndexMapper メソッドで使用されます。

```
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIndexHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIndexRangeKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;

@DynamoDBTable(tableName = "Issues")
public class CreateDate {
 private String createDate;
 @DynamoDBHashKey(attributeName = "IssueId")
 private String issueId;

 @DynamoDBIndexHashKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "CreateDate")
 public String getCreateDate() {
  return createDate;
 }
 public void setCreateDate(String createDate) {
  this.createDate = createDate;
 }

 @DynamoDBIndexRangeKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "IssueId")
 public String getIssueId() {
  return issueId;
 }

 public void setIssueId(String issueId) {
  this.issueId = issueId;
 }
}
```

DynamoDB のドキュメント履歴

2018 年 7 月 3 日以降の DynamoDB デベロッパーガイドの各リリースにおける重要な変更点を以下の表に示します。このドキュメントの更新に関する通知については、RSS フィード (このページの左上) にサブスクライブできます。

変更	説明	日付
DynamoDB のモニタリングとログ記録のドキュメントを再構成、統合	DynamoDB でのモニタリングとログ記録の新しい構造には、メトリクス、ログ記録オペレーション、投稿者のインサイトに関する 3 つの簡潔な章が含まれています。	2024 年 5 月 3 日
DynamoDB キャパシティモードのドキュメントを再構成、統合	DynamoDB ガイドに、DynamoDB キャパシティモード (オンデマンドとプロビジョンド) に関するすべての情報を示す新しい章が追加されました。この更新では、「読み込み/書き込みキャパシティモードを変更する際の考慮事項」トピックが、「ベストプラクティス」章に移動されました。このトピックは、「キャパシティモードを切り替える際の考慮事項」に名前が変更され、キャパシティモードを切り替える際のベストプラクティスに関する詳細な情報が追加されました。さらに、このガイドには、DynamoDB の読み込み/書き込みと、読み込み/書き込みオペレーションのキャパシティユニッ	2024 年 5 月 1 日

	トの消費量に関するすべての情報を示す新しい章が追加されました。詳細については、「DynamoDB のスループットキャパシティ」、「キャパシティモードを切り替える際の考慮事項」、「DynamoDB の読み込みと書き込み」を参照してください。	
オンデマンドリクエストの最大数	個別のテーブル、インデックス、またはその両方で実行できるオンデマンドリクエストの最大数を指定できるようになりました。最大のオンデマンドスループットを指定すると、テーブルレベルの使用量とコストを制限し、リソース消費量の意図しない急増を防止できます。詳細については、「オンデマンドテーブルの最大スループット」を参照してください。	2024 年 5 月 1 日
NoSQL Workbench オペレーションビルダーの改善点	NoSQL Workbench には、ダークモードのネイティブサポートが含まれるようになりました。オペレーションビルダーのテーブルと項目のオペレーションを改善しました。項目の結果とオペレーションビルダーのリクエスト情報は JSON 形式で利用できます。詳細については、「NoSQL Workbench オペレーションビルダー」を参照してください。	2024 年 4 月 24 日

Amazon DynamoDB リソース向けのリソースベースのポリシー	テーブル、インデックス、ストリーム用のリソースベースのポリシーが DynamoDB でサポートされるようになりました。リソースベースのポリシーでは、各リソースにアクセスできるユーザーと、各リソースに対して実行できるアクションを指定することで、アクセス許可を定義できます。詳細については、「DynamoDB 用のリソースベースのポリシーを使用する」を参照してください。	2024 年 3 月 20 日
DynamoDB マネージドポリシーのアップデート	`AmazonDynamoDBReadOnlyAccess` マネージドポリシーに新しいアクセス許可 `dynamodb:GetResourcePolicy` を追加しました。このアクセス許可は、DynamoDB リソースにアタッチされたリソースベースのポリシーの読み取りアクセスを提供します。詳細については、「AWS 管理ポリシー: AmazonDynamoDBReadOnlyAccess」を参照してください。	2024 年 3 月 20 日

Amazon DynamoDB 向けの AWS PrivateLink	AWS PrivateLink が Amazon DynamoDB でサポートされるようになりました。AWS PrivateLink では、インターフェイス VPC エンドポイントおよびプライベート IP アドレスを使用して、仮想プライベートクラウド (VPC)、DynamoDB、およびオンプレミスデータセンター間のプライベートネットワーク接続を簡素化できます。詳細については、「AWS PrivateLink for DynamoDB」を参照してください。	2024 年 3 月 19 日
JavaScript を使ったプログラミングのためのガイド	Amazon DynamoDB は、AWS SDK for JavaScript プログラミングガイドを提供しています。AWS SDK for JavaScript、抽象化レイヤー、接続設定、エラー処理、再試行ポリシーの定義、キープアライブの管理などについて学ぶことができます。詳細については、「JavaScript による Amazon DynamoDB のプログラミング」を参照してください。	2024 年 3 月 6 日

AWS SDK for Java 2.x を使ったプログラミングのためのガイド	高レベル、低レベル、ドキュメントインタフェース、HTTP クライアントとその構成、エラー処理の詳細、SDK for Java 2.x を使用する際に考慮すべき最も一般的な構成設定について説明する新しいプログラミングガイドを作成しました。詳細については、「AWS SDK for Java 2.x を使用した Amazon DynamoDB のプログラミング」を参照してください。	2024 年 3 月 5 日
NoSQL Workbench を使用したテーブルのクローン作成	開発者は NoSQL Workbench を使用して、開発環境とリージョン (DynamoDB ローカルと DynamoDB ウェブ) 間でテーブルのコピーやクローンを作成することができます。詳細については、「NoSQL Workbench を使用したテーブルのクローン作成」を参照してください。	2024 年 2 月 26 日
Python を使ったプログラミングのためのガイド	高レベルライブラリ、低レベルライブラリの詳細、Python SDK を使用する際に考慮すべき最も一般的な設定について説明する新しいガイドを作成しました。詳細については、「Python と Boto3 による Amazon DynamoDB のプログラミング」を参照してください。	2024 年 1 月 5 日

Time to live (TTL) トピックのリライト	ガイド内の TTL セクションをすべてリライトしました。更新されたセクションにはすぐに使用できるコードスニペットが用意されており、TTL の使用を開始する際に役立ちます。現在提供されているコードスニペットは Python と Javascript で記述されています。詳細については、「TTL」を参照してください。	2023 年 12 月 20 日
AWS の請求および使用状況レポートを理解するためのベストプラクティス	DynamoDB のさまざまな使用タイプとそれらの使用タイプの料金を明確にする新しいセクションを追加しました。詳細については、「AWS の請求および使用状況レポートを理解するためのベストプラクティス」を参照してください。	2023 年 12 月 15 日
Amazon OpenSearch Service と Amazon DynamoDB のゼロ ETL 統合	Amazon OpenSearch Service とのゼロ ETL 統合が Amazon DynamoDB でサポートされるようになりました。これにより、カスタムコードやインフラストラクチャを使用せずに、DynamoDB データを自動的に複製および変換することで検索を実行できます。詳細については、「DynamoDB と Amazon OpenSearch Service のゼロ ETL 統合」を参照してください。	2023 年 11 月 28 日

リレーショナルデータベースから DynamoDB への移行	リレーショナルデータベースから DynamoDB への移行方法をユーザーが理解できるように、移行ガイドを作成しました。	2023 年 11 月 27 日
NoSQL Workbench でサンプルデータを生成する	NoSQL Workbench for Amazon DynamoDB で、サンプルデータ型テンプレートから直接データ型を作成できるようになったため、ワークロードのデータスキーマを設計しやすくなりました。この機能を使用して、DynamoDB でアプリケーションを構築するときに NoSQL データモデリングのベストプラクティスに慣れることができます。	2023 年 9 月 28 日
S3 への増分エクスポート	挿入、更新、削除されたデータをわずかな増分ずつエクスポートできるようになりました。増分エクスポートでは、AWS 管理コンソール、API 呼び出し、または AWS コマンドラインインターフェイスで数回クリックするだけで、数メガバイトからテラバイト単位までの変更データをエクスポートできます。	2023 年 9 月 26 日

DynamoDB のデータモデリング	特定のユースケース、そのアクセスパターン、およびそれらのアクセスパターンを実現するためのステップバイステップのガイダンスにに焦点を当てた DynamoDB の例を使用することで、データモデリングについて詳しく学習できるようになりました。	2023 年 7 月 14 日
「トラブルシューティング」セクション	DynamoDB テーブルで発生する可能性のあるレイテンシーとスロットリングの問題について、トラブルシューティングのコンテンツが見つかるようになりました。	2023 年 3 月 13 日
Amazon DynamoDB の削除保護	すべての AWS リージョンの Amazon DynamoDB テーブルで削除保護を利用できるようになりました。DynamoDB では、通常のテーブル管理オペレーションを実行する際に、テーブルが誤って削除されないように保護できるようになりました。	2023 年 3 月 8 日

グローバルテーブルでの KDSD に対する AWS CloudFormation のサポート	Amazon Kinesis Data Streams for DynamoDB で DynamoDB グローバルテーブルに対して AWS CloudFormation がサポートされるようになりました。これにより、CloudFormation テンプレートを使用して DynamoDB グローバルテーブル上の Amazon Kinesis Data Streams へのストリーミングを有効化できるようになりました。	2023 年 2 月 15 日
DynamoDB ローカルは 1 トランザクションあたり 100 アクションをサポート	DynamoDB ローカルでは、1 つのトランザクションで最大 100 のアクションを実行できるようになりました。	2023 年 2 月 9 日
DynamoDB の Well-Architected レンズを使用して DynamoDB ワークロードを最適化する	優れたアーキテクチャの DynamoDB ワークロードを設計するための一連の設計原則とガイダンス、DynamoDB Well-Architected レンズを使用できるようになりました。	2023 年 2 月 3 日
PartiQL GovGloud のサポート	AWS GovCloud (米国東部) および AWS GovCloud (米国西部) で PartiQL (Amazon DynamoDB の SQL 互換のクエリ言語) がサポートされるようになりました。	2022 年 12 月 21 日

NoSQL ワークベンチおよび DynamoDB ローカルのシングルインストールスイート	NoSQL Workbench for DynamoDB に、ガイド付きの DynamoDB ローカルインストールプロセスが含まれるようになり、DynamoDB ローカル開発環境の設定が効率化されました。	2022 年 12 月 6 日
S3 からの一括インポート	Amazon DynamoDB では、Amazon S3 からの一括データインポートをサポートすることで、新しい DynamoDB テーブルへのデータの移行とロードが簡単になりました。	2022 年 8 月 18 日
Service Quotas との高度な統合	Service Quotas では、アカウントとテーブルクォータをプロアクティブに管理できるようになりました。現在の値を表示したり、クォータの使用率が設定可能なしきい値を超えた場合のアラームを設定したりできます。	2022 年 6 月 15 日
NoSQL Workbench がテーブルと GSI のサポートを追加	NoSQL Workbench をテーブルとグローバルセカンダリインデックス (GSI) のコントロールプレーンオペレーション (CreateTable、UpdateTable、DeleteTable など) に使用できるようになりました。	2022 年 6 月 2 日

Standard-Infrequent Access テーブルクラスが中国で利用可能に	Amazon DynamoDB Standard-Infrequent Access テーブルクラスが中国リージョンで利用可能です。この新しいテーブルクラスをアクセス頻度の低いデータを格納するテーブルで使用することで、DynamoDB のコストが最大 60% 削減されます。	2022 年 4 月 18 日
デフォルトの Service Quotas とテーブル管理オペレーションの増加	DynamoDB では、アカウントおよびリージョンあたりのテーブル数のデフォルトクォータが 256 テーブルから 2,500 テーブルに増加し、同時実行テーブルの管理オペレーションの数が 50 から 500 に増加しました。	2022 年 3 月 9 日
DynamoDB 用の PartiQL を使用して項目を制限するオプション	DynamoDB では、各リクエストのオプションパラメータとして、DynamoDB オペレーションの PartiQL で処理するアイテム数を制限することができます。	2022 年 3 月 8 日
AWS Backup 統合が北京および寧夏リージョンで利用可能に	AWS Backup が、中国 (北京および寧夏) リージョンで DynamoDB に統合されるようになりました。アカウント間、リージョン間のバックアップなど、AWS Backup でのバックアップ機能の強化により、コンプライアンスやビジネス継続性の要件をより容易に満たすことができます。	2022 年 1 月 26 日

PartiQL API コールによるスループットキャパシティに関する情報	DynamoDB は、PartiQL API コールによって消費されたスループット容量を返すことができ、クエリとスループットコストの最適化を支援します。	2022 年 1 月 18 日
AWS Backup の統合	DynamoDB では、アカウント間、リージョン間のバックアップなど、AWS Backup でのバックアップ機能の強化により、コンプライアンスやビジネス継続性の要件をより簡単に満たすことができるようになりました。	2021 年 11 月 24 日
CSV での NoSQL Workbench データセットの読み込み/書き出し	Amazon DynamoDB 用 NoSQL Workbench では、データモデルの構築と可視化に役立つサンプルデータのインポートと自動投入が可能になりました。	2021 年 10 月 11 日
Amazon DynamoDB Streams データプレーンアクティビティを AWS CloudTrail でフィルタリングして取得する	Amazon DynamoDB では、Streams のデータプレーン API アクティビティを AWS CloudTrail でフィルタリングできるようになり、監査ログをより詳細に制御できるようになりました。	2021 年 9 月 22 日

コンソールの更新	DynamoDB コンソールがデフォルトのコンソールになり、データ管理を容易にし、一般的なタスクを簡素化し、リソースや機能へより迅速にアクセスできるようになりました。	2021 年 8 月 25 日
DAX SDK For Java 2.x が利用可能に	DynamoDB Accelerator (DAX) SDK for Java 2.x が利用可能になりました。これは、AWS SDK for Java 2.x と互換性があります。ノンブロック I/O を含む最新機能のメリットを得ることができます。	2021 年 7 月 29 日
コントロールプレーンオペレーションを含む NoSQL Workbench 機能の更新	Amazon DynamoDB 用 NoSQL Workbench では、テーブルのデータを変更したりアクセスしたりするための頻繁な操作を、より簡単に実行できるようになりました。	2021 年 7 月 28 日
DynamoDB グローバルテーブルがアジアパシフィックリージョンで利用可能に	DynamoDB グローバルテーブルがアジアパシフィックリージョンで利用可能になりました。選択する 22 の AWS リージョン全体で DynamoDB テーブルを自動的に複製します。	2021 年 7 月 28 日
DAX が中国で利用可能に	DynamoDB Accelerator (DAX) が、Sinnet が運営する中国 (北京) リージョンで利用可能になりました。	2021 年 7 月 28 日

転送時の DAX 暗号化	DynamoDB Accelerator (DAX) では、アプリケーションと DAX クラスター間、および DAX クラスター内のノード間のデータ転送時の暗号化がサポートされるようになりました。	2021 年 7 月 24 日
CloudFormation と CloudTrail の統合	AWS CloudFormation との統合 および CloudFormation データプレーンのログ記録によるセキュリティの機能強化。	2021 年 6 月 18 日
グローバルテーブルが CloudFormation をサポート	Amazon DynamoDB グローバルテーブルで AWS CloudFormation がサポートされるようになり、CloudFormation テンプレートを使用してグローバルテーブルを作成し、その設定を管理できます。	2021 年 5 月 14 日
Amazon DynamoDB Local が Java 2.x をサポート	Amazon DynamoDB のダウンロード版である DynamoDB ローカル で、AWS SDK for Java 2.x を使用できるようになりました。DynamoDB local を使用すると、ローカル開発環境で実行されている DynamoDB のバージョンを使用して、追加コストをかけずにアプリケーションを開発およびテストできます。	2021 年 5 月 3 日

NoSQL Workbench が AWS CloudFormation をサポート	Amazon DynamoDB 用 NoSQL Workbench で AWS CloudFormation がサポートされるようになり、CloudFormation テンプレートで DynamoDB のデータモデルを管理および変更できるようになりました。さらに、NoSQL Workbench でテーブルの容量設定を構成できるようになりました。	2021 年 4 月 22 日
DynamoDB と AWS Amplify の機能が統合	AWS Amplify では、1 つのデプロイで複数の DynamoDB グローバルセカンダリインデックスの更新をオーケストレートできるようになりました。	2021 年 4 月 20 日
AWS CloudTrail で Amazon DynamoDB Streams のデータプレーン API をログに記録する	AWS CloudTrail を使用して Amazon DynamoDB Streams のデータプレーン API アクティビティをログに記録し、DynamoDB テーブルの項目レベルの変更をモニタリングおよび調査することができるようになりました。	2021 年 4 月 20 日

Amazon DynamoDB 用 Amazon Kinesis Data Streams で AWS CloudFormation がサポートされるようになりました	Amazon DynamoDB 用 Amazon Kinesis Data Streams で AWS CloudFormation がサポートされるようになり、CloudFormation テンプレートを使用して DynamoDB テーブル上の Amazon Kinesis Data Streams へのストリーミングを有効化できるようになりました。DynamoDB データの変更を Kinesis Data Streams にストリーミングすることで、Amazon Kinesis のサービスを使用して高度なストリーミングアプリケーションを構築できます。	2021 年 4 月 12 日
Amazon Keyspaces が FIPS 140-2 準拠のエンドポイントを提供	Amazon Keyspaces (Apache Cassandra 用) が、連邦情報処理規格 (FIPS) 140-2 準拠のエンドポイントを提供し、高度に規制されたワークロードをより簡単に実行できるようになりました。FIPS 140-2 は、機密情報を保護する暗号モジュールのセキュリティ要件を規定する米国およびカナダ政府の規格です。	2021 年 4 月 8 日

DAX 用の Amazon EC2 T3 インスタンス	DAX で Amazon EC2 T3 インスタンスタイプがサポートされるようになりました。このインスタンスタイプには、CPU パフォーマンスのベースラインレベルを設定でき、必要な場合にそのレベルを超えてバーストする機能を備えています。	2021 年 2 月 15 日
NoSQL Workbench for Amazon DynamoDB の PartiQL サポート	NoSQL Workbench for DynamoDB を使用して、DynamoDB 用に PartiQL ステートメントを作成できるようになりました。	2020 年 12 月 4 日
DynamoDB 用 PartiQL	DynamoDB 用 PartiQL (SQL 互換クエリ言語) を使用して、DynamoDB テーブルを操作できるようになりました。また、PartiQL 用に AWS Management Console、AWS Command Line Interface、もしくは DynamoDB API を使用して、アドホックのクエリを実行することもできます。	2020 年 11 月 23 日
Amazon DynamoDB 用 Amazon Kinesis Data Streams	Amazon DynamoDB 用 Amazon Kinesis Data Streams と DynamoDB テーブルを使用して、項目レベルの変更をキャプチャし、その結果を Kinesis Data Streams にレプリケートできるようになりました。	2020 年 11 月 23 日

DynamoDB テーブルのエクスポート	DynamoDB テーブルを Amazon S3 にエクスポートできるようになりました。この機能により、Athena、AWS Glue、Lake Formation などのサービスを使用しながら、データに対する分析や複雑なクエリを実行できます。	2020 年 11 月 9 日
空の値のサポート	DynamoDB テーブルにおいて、非キー文字列およびバイナリ属性の空の値を使用できるようになりました。空の値のサポートにより、DynamoDB に送信する前に属性を変換しなくてもよくなり、より広範なユースケースで属性を使用するための柔軟性が高まります。空の文字列とバイナリ値では、リスト作成、マッピング、およびデータ型の設定も行えます。	2020 年 5 月 18 日
NoSQL Workbench for Amazon DynamoDB での Linux サポート	NoSQL Workbench for Amazon DynamoDB で、Linux- Ubuntu、Fedora および Debian のサポートが開始されました。	2020 年 5 月 4 日

CloudWatch Contributor Insights for DynamoDB – 一般提供	DynamoDB 用 CloudWatch Contributor Insights の一般提供が開始されました。DynamoDB 用 CloudWatch Contributor Insights は、DynamoDB テーブルにおけるトラフィックの傾向を一目で把握できる診断ツールです。テーブルで最も頻繁にアクセスされるキー (ホットキーとも呼ばれます) を特定するのに役立ちます。	2020 年 4 月 2 日
グローバルテーブルのアップグレード	グローバルテーブルのバージョン 2017.11.29 からの、最新のバージョン (2019.11.21) への更新が、DynamoDB コンソールで数回クリックするだけで行えるようになりました。グローバルテーブルのバージョンをアップグレードすることで、既存のテーブルを他の AWS リージョンに拡張する際にもテーブルの再構築は必要なくなり、DynamoDB テーブルの可用性を簡単に向上させることができます。	2020 年 3 月 16 日
Amazon DynamoDB 用 NoSQL Workbench – 一般提供	Amazon DynamoDB 用 NoSQL Workbench の一般提供が開始されました。DynamoDB テーブルの設計、作成、クエリ、および管理に、NoSQL Workbench をご利用いただけます。	2020 年 3 月 2 日

DAX キャッシュクラスターのメトリクス	DAX で新たな CloudWatch メトリクスがサポートされ、DAX クラスターのパフォーマンスをよりよく把握できるようになりました。	2020 年 2 月 6 日
DynamoDB 用 CloudWatch Contributor Insights – プレビュー	DynamoDB 用 CloudWatch Contributor Insights は、DynamoDB テーブルにおけるトラフィックの傾向を一目で把握できる診断ツールです。テーブルで最も頻繁にアクセスされるキー (ホットキーとも呼ばれます) を特定するのに役立ちます。	2019 年 11 月 26 日
不均衡なワークロードのための適応型キャパシティのサポート	Amazon DynamoDB の適応型キャパシティーにより、頻繁にアクセスされるアイテムを自動的に分離することで、不均衡なワークロードを処理できるようになりました。アプリケーションが、1 つ以上のアイテムに不均等な高トラフィックを送っている場合、DynamoDB がパーティションのバランスを再調整することで、アクセス頻度の高いアイテムが同一パーティションに偏ることを防ぎます。	2019 年 11 月 26 日

カスタマーマネージドキーのサポート	DynamoDB でカスタマーマネージドキーのサポートが開始されました。つまり、DynamoDB データの暗号化やセキュリティ管理を完全に制御できます。	2019 年 11 月 25 日
NoSQL Workbench による DynamoDB local (ダウンロード可能バージョン) のサポート	NoSQL Workbench では、DynamoDB local (ダウンロード可能バージョン) に接続して、DynamoDB テーブルを設計、作成、クエリ、管理できるようになりました。	2019 年 11 月 8 日
NoSQL Workbench – プレビュー	これは DynamoDB 用 NoSQL Workbench の初版リリースです。NoSQL Workbench を使用して、DynamoDB テーブルを設計、作成、クエリ、および管理します。詳細については、「Amazon DynamoDB 用 NoSQL Workbench (プレビュー)」を参照してください。	2019 年 9 月 16 日
DAX が Python と .NET を使用したトランザクションオペレーションのサポートを追加	DAX は、Go、Java、.NET、Node.js、および Python で記述されたアプリケーション用の `TransactWriteItems` と `TransactGetItems` API をサポートしています。詳細については、「インメモリアクセラレーションと DAX」を参照してください。	2019 年 2 月 14 日

Amazon DynamoDB local (ダウンロード可能バージョン) の更新	DynamoDB local (ダウンロード可能バージョン) がトランザクション API、オンデマンドの読み取り/書き込みキャパシティー、読み取りおよび書き込みオペレーションのキャパシティー報告、20 個のグローバルセカンダリインデックスをサポートするようになりました。詳細については、「ダウンロード可能な DynamoDB と DynamoDB ウェブサービスの違い」を参照してください。	2019 年 2 月 4 日
Amazon DynamoDB オンデマンド	DynamoDB オンデマンドは、容量計画なしで 1 秒あたりに数千ものリクエストを処理できる柔軟な請求オプションです。DynamoDB オンデマンドには、読み取りおよび書き込みリクエストのリクエストごとの支払い料金が用意されているため、使用した分だけ課金されます。詳細については、「DynamoDB スループットキャパシティ」を参照してください。	2018 年 11 月 28 日

Amazon DynamoDB Transactions	DynamoDB のトランザクションは、テーブル内およびテーブル間の複数の項目に調整されたオールオアナッシングの変更を行うことで、DynamoDB にアトミック性、整合性、分離、および堅牢性 (ACID) をもたらします。詳細については、「Amazon DynamoDB Transactions」をご覧ください。	2018 年 11 月 27 日
Amazon DynamoDB により、お客様の保管時のデータをすべて暗号化	DynamoDB の保管時の暗号化は、そのプライマリキー、ローカルおよびグローバルセカンダリインデックス、ストリーム、グローバルテーブル、バックアップ、DAX クラスター (データが堅牢なメディアに保存される場合) を含む、暗号化されたテーブルでデータを保護することにより、追加のデータ保護レイヤーを提供します。詳細については、「保管時の Amazon DynamoDB 暗号化」を参照してください。	2018 年 11 月 15 日

新しいドッカーイメージで Amazon DynamoDB ローカルをさらに容易に	DynamoDB のダウンロード可能なバージョンである、DynamoDB local の使用がさらに容易になりました。新たな DynamoDB local Docker イメージによる、DynamoDB アプリケーションの開発とテストが行えます。詳細およびダウンロード手順については、「DynamoDB (Downloadable Version) and Docker」を参照してください。	2018 年 8 月 22 日
DynamoDB Accelerator (DAX) に保管時の暗号化のサポートが追加されました	DynamoDB Accelerator (DAX) で、新しい DAX クラスターの保管時の暗号化が行えます。セキュリティが重要で、厳格なコンプライアンスと規制要件の対象となるアプリケーションでの、 Amazon DynamoDB テーブルからの読み込みが高速化されます。詳細については、「DAX Encryption at Rest」を参照してください。	2018 年 8 月 9 日
DynamoDB ポイントインタイムリカバリ (PITR) で削除されたテーブル復元のサポートが追加されました	ポイントインタイムリカバリが有効になっているテーブルを削除すると、システムバックアップが自動的に作成され、35 日間 (追加料金なしで) 保持されます。詳細については、「ポイントインタイムリカバリを使用して開始する前に」を参照してください。	2018 年 8 月 7 日

更新を RSS で今すぐ入手可能	RSS フィード (このページの左上) にサブスクライブして、Amazon DynamoDB 開発者ガイドの更新についての通知を受信できるようになりました。	2018 年 7 月 3 日

以前の更新

以下の表に、2018 年 7 月 3 日以前の DynamoDB デベロッパーガイドにおける重要な変更点を示します。

変更	説明	変更日
DAX での Go 言語のサポート	DynamoDB Accelerator (DAX) SDK for Go を使用して Go プログラム言語で記述されたアプリケーションによる、Amazon DynamoDB テーブルのマイクロ秒レベルでの高速読み込みができるようになりました。詳細については、「DAX SDK for Go」を参照してください。	2018 年 26 月 6 日
DynamoDB から SLA がリリース	DynamoDB は、パブリックアベイラビリティの SLA をリリースしました。詳細については、「Amazon DynamoDB サービスレベルアグリーメント」を参照してください。	2018 年 6 月 19 日
DynamoDB での継続的なバックアップとポイントインタイムリカバリ (PITR)	ポイントインタイムリカバリを使用することで、オペレーションによって Amazon DynamoDB テーブルが誤っ	2018 年 4 月 25 日

変更	説明	変更日
	て上書きされたり削除されたりしないようにできます。ポイントインタイムリカバリを有効化すれば、オンデマンドバックアップの作成、維持、スケジュールを心配する必要はありません。たとえば、テストスクリプトで、誤って本稼働環境の DynamoDB テーブルに書き込みを行ったとします。ポイントインタイムリカバリを使用すれば、過去 35 日間の任意の時点にテーブルを復元することができます。DynamoDB では、テーブルの増分バックアップが維持されます。詳細については、「DynamoDB のポイントインタイムリカバリ」を参照してください。	
DynamoDB での保管時の暗号化	新しい DynamoDB テーブルで利用できる DynamoDB の保管時の暗号化は、AWS Key Management Service に保管された AWS マネージド暗号キーを使用して、アプリケーションデータを Amazon DynamoDB テーブルに保護するために便利です。詳細については、「保管時の DynamoDB 暗号化」を参照してください。	2018 年 2 月 8 日

変更	説明	変更日
DynamoDB でのバックアップと復元	オンデマンドバックアップで、DynamoDB テーブルのデータの完全なバックアップを作成してアーカイブできます。これを利用して企業および行政の既成要件を満たすことができます。数メガバイトから数百テラバイトまでのデータを持つテーブルを、本稼働アプリケーションのパフォーマンスや可用性に影響を与えずにバックアップできます。詳細については、「DynamoDB のオンデマンドバックアップおよび復元の使用」を参照してください。	2017 年 11 月 29 日

変更	説明	変更日
DynamoDB グローバルテーブル	DynamoDB のグローバルフットプリント上に構築されたグローバルテーブルにより、完全マネージド型で、マルチリージョン、マルチアクティブなデータベースが提供されます。このデータベースでは、大規模に拡張されたグローバルアプリケーションからの読み込み/書き込みのパフォーマンスが高速に、かつローカルに実現されます。Global Tables は、選択する AWS リージョン全体で Amazon DynamoDB テーブルを自動的に複製します。詳細については、「グローバルテーブル – DynamoDB の複数リージョンレプリケーション」を参照してください。	2017 年 11 月 29 日
DAX の Node.js のサポート	Node.js のデベロッパーは、Node.js 向けの DAX クライアントを使用することで、DynamoDB Accelerator (DAX) を活用できます。詳細については、「DynamoDB Accelerator (DAX) とインメモリアクセラレーション」を参照してください。	2017 年 10 月 5 日

変更	説明	変更日
DynamoDB の VPC エンドポイント	DynamoDB エンドポイントにより、Amazon VPC 内にある Amazon EC2 インスタンスは、パブリックインターネットに公開されることなく DynamoDB にアクセスできるようになります。VPC と DynamoDB 間のネットワークトラフィックは、Amazon ネットワーク内のみに置かれます。詳細については、「Amazon VPC エンドポイントを使用して DynamoDB にアクセスする」を参照してください。	2017 年 8 月 16 日

変更	説明	変更日
DynamoDB での Auto Scaling	DynamoDB での Auto Scaling では、プロビジョニングされたスループットの設定を手動で定義または調整する必要はありません。代わりに、DynamoDB の Auto Scaling では、実際のトラフィックパターンに応じて、読み込み/書き込み容量を動的に調整します。これにより、テーブルまたはグローバルセカンダリインデックスで、プロビジョニングされた読み込み/書き込みキャパシティーが拡張され、トラフィックの急激な増加をスロットリングなしに処理できるようになります。ワークロードが減少した場合には、DynamoDB の Auto Scaling はプロビジョニングされているキャパシティーを削減します。詳細については、「DynamoDB Auto Scaling によるスループットキャパシティの自動管理」を参照してください。	2017 年 6 月 14 日

変更	説明	変更日
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) は、フルマネージド型で可用性の高い、DynamoDB 用のインメモリキャッシュです。1 秒あたりのリクエスト数が数百万におよぶ場合であっても、最大 10 倍のパフォーマンスの (ミリセカンドからマイクロセカンドの範囲への) 向上を実現します。詳細については、「DynamoDB Accelerator (DAX) とインメモリアクセラレーション」を参照してください。	2017 年 4 月 19 日
DynamoDB が項目の有効期限 (TTL) による自動的な停止をサポート	Amazon DynamoDB の Time-to-Live (TTL) では、追加料金なしに、有効期限が切れた項目をテーブルから自動的に削除することができます。詳細については、「Time to Live (TTL)」を参照してください。	2017 年 2 月 27 日
DynamoDB がコスト配分タグをサポート	使用量の類別と細分化したコストレポートの改善に、Amazon DynamoDB にタグを追加できるようになりました。詳細については、「リソースへのタグとラベルの追加」を参照してください。	2017 年 1 月 19 日

変更	説明	変更日
新しい DynamoDB `DescribeLimits` API	`DescribeLimits` API は、リージョンの AWS アカウント用に現在プロビジョンされた容量の上限を返します。この API はリージョン全体と、そこで作成した任意の DynamoDB テーブルの両方について値を返します。これにより、現在のアカウントレベルの制限を判断し、現在使用中のプロビジョニングされた容量と比較できるようになります。また、制限に達する前に引き上げを申請するために十分な時間が得られます。詳細については、Amazon DynamoDB API リファレンスの「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」、および「DescribeLimits」を参照してください。	2016 年 3 月 1 日

変更	説明	変更日
DynamoDB コンソールの更新とプライマリキー属性に関する新しい用語	DynamoDB マネジメントコンソールが、より直感的に、より使いやすくするために設計し直されました。この更新の一部であるプライマリキー属性の新しい用語について説明します。 • [Partition Key (パーティションキー)] – ハッシュ属性とも呼ばれます。 • [Sort Key (ソートキー)] – 範囲属性とも呼ばれます。 名前のみが変更され、機能は同じです。 テーブル、またはセカンダリインデックスを作成する際には、(パーティションキーのみの) シンプルなプライマリキーを選択することも、(パーティションキーおよびソートキーによる) 複合的なプライマリキーを選択することも可能です。これらの変更を反映するために、DynamoDB ドキュメントが更新されています。	2015 年 12 月 11 日

変更	説明	変更日
Titan 向け Amazon DynamoDB Storage Backend	Titan 向け Amazon DynamoDB Storage Backend は、Amazon DynamoDB 上に実装される Titan グラフデータベース用のストレージバックエンドです。Titan 向け Amazon DynamoDB Storage Backend を使用する場合には、Amazon の高い可用性を備えたデータセンター全体で実行される DynamoDB による保護が、データで活用できるようになります。プラグインは、Titan バージョン 0.4.4 (主に既存のアプリケーションに対応) と Titan バージョン 0.5.4 (新しいアプリケーションに推奨) で使用できます。このプラグインは、Titan 用の他のストレージバックエンドのように、Blueprints API および Gremlin シェルを含めて、T inkerpop スタック (バージョン 2.4 および 2.5) をサポートします。詳細については、「Titan 用 Amazon DynamoDB ストレージバックエンド」を参照してください。	2015 年 8 月 20 日

変更	説明	変更日
DynamoDB Streams、クロスリージョンレプリケーション、および強力な整合性のある読み込みによるスキャニング	DynamoDB Streams は、DynamoDB テーブル内の項目レベルの変更に関するシーケンスを時間順にキャプチャし、その情報を最大 24 時間ログに保存します。アプリケーションは、このログにアクセスし、データ項目の変更前および変更後の内容をほぼリアルタイムで参照できます。詳細については、「DynamoDB Streams の変更データキャプチャ」および DynamoDB Streams API リファレンスを参照してください。 DynamoDB クロスリージョンレプリケーションは、異なる AWS リージョン間で、DynamoDB テーブルの完全なコピーをほぼリアルタイムで維持するための、クライアント側ソリューションです。クロスリージョンレプリケーションを使用することで、DynamoDB テーブルをバックアップしたり、ユーザーが地理的に分散されたケースで、データへの低レイテンシーなアクセスを実現したりできます。 DynamoDB Scan オペレーションは、デフォルトで	2015 年 7 月 16 日

変更	説明	変更日
	結果整合性のある読み込みを使用します。代わりに、`ConsistentRead` パラメータを true に設定することによって、強い整合性のある読み込みを使用できます。詳細については、Amazon DynamoDB API リファレンスの「スキャンの読み込み整合性」および「Scan」を参照してください。	
Amazon DynamoDB の AWS CloudTrail サポート	DynamoDB が CloudTrail に統合されました。CloudTrail は、DynamoDB コンソールまたは DynamoDB API で行われた API 呼び出しをキャプチャーし、ログファイルに記録します。詳細については、AWS CloudTrail を使用して DynamoDB オペレーションをログに記録する および AWS CloudTrail ユーザーガイドを参照してください。	2015 年 5 月 28 日

変更	説明	変更日
クエリ式のサポートの向上	このリリースでは、KeyConditionExpression API に新しい Query パラメータが追加されました。Query は、プライマリキー値を使用してテーブルまたはインデックスから項目を読み込みます。KeyConditionExpression パラメータは、プライマリキーの名前、さらにキー値に適用される条件を特定する文字列です。Query は、式を満たすこれらの項目のみを取り出します。KeyConditionExpression の構文は、DynamoDB で使用する他のパラメータ表現に似ています。この構文により、表現内の名前や値に置換する変数を定義できます。詳細については、「DynamoDB のクエリオペレーション」を参照してください。	2015 年 4 月 27 日

変更	説明	変更日
条件付き書き込みの新しい比較関数	DynamoDB では、ConditionExpression パラメータにより PutItem、UpdateItem 、または DeleteItem の成否が判断されます。項目は、条件が true に評価された場合にのみ書き込まれます。このリリースでは、attribute_type で使用するための 2 つの新しい機能、size と ConditionExpression が追加されました。これらの機能により、テーブルの属性のデータの型やサイズに基づいて条件付き書き込みを実行できます。詳細については、「条件式」を参照してください。	2015 年 4 月 27 日

変更	説明	変更日
セカンダリインデックスのスキャン API	DynamoDB では、テーブル内のすべての項目は Scan オペレーションで読み込みまれます。これらの項目には、ユーザー定義のフィルター基準が適用され、そこで選択されたデータ項目がアプリケーションに返されます。この同じ機能は、セカンダリインデックスでも使用できるようになりました。ローカルセカンダリインデックスまたはグローバルセカンダリインデックスをスキャンするには、インデックス名と親テーブルの名前を指定します。デフォルトでは、インデックス Scan はインデックス内のすべてのデータを返します。フィルタ式を使用して、アプリケーションに返される結果を絞り込むことができます。詳細については、「DynamoDB でのスキャンの使用」を参照してください。	2015 年 2 月 10 日

変更	説明	変更日
グローバルセカンダリインデックスのオンラインオペレーション	オンラインインデックスでは、既存のテーブルにグローバルセカンダリインデックスを追加または削除することができます。オンラインインデックスを使用すれば、テーブルの作成時にテーブルのすべてのインデックスを定義する必要はありません。代わりに、いつでも新しいインデックスを追加できます。同様に、インデックスが不要であると判断した場合は、いつでもそのインデックスを削除できます。オンラインインデックスオペレーションはノンブロッキングであるため、インデックスの追加または削除中に、テーブルは読み取りまたは書き込みアクティビティに利用できる状態になります。詳細については、「グローバルセカンダリインデックスの管理」を参照してください。	2015 年 1 月 27 日

変更	説明	変更日
JSON によるドキュメントモデルのサポート	DynamoDB は、すべてのドキュメントモデルをサポートしており、それを保存および利用することができます。新しいデータ型は、JSON 標準と完全互換で、ドキュメント要素を互いに入れ子にすることができます。ドキュメントパスの逆参照演算子を使用して、ドキュメント全体を取得しなくても個々の要素を読み書きすることができます。今回のリリースでは、データ項目の読み書き時に射影、条件、更新アクションを指定する新しい式パラメータも導入されました。JSON によるドキュメントモデルのサポートについて詳しくは、「データ型」と「DynamoDB での式の使用」を参照してください。	2014 年 10 月 7 日

変更	説明	変更日
柔軟なスケーリング	テーブルおよびグローバルセカンダリインデックスについては、テーブルごと、および、アカウントごとの制限内であれば、プロビジョニングされた読み書きスループットキャパシティを任意の量だけ増やすことができます。詳細については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。	2014 年 10 月 7 日
項目サイズの拡大	DynamoDB の最大項目サイズが 64 KB から 400 KB に拡大されました。詳細については、「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」を参照してください。	2014 年 10 月 7 日

変更	説明	変更日
強化された条件式	DynamoDB で条件式に使用できる演算子が強化され、条件付きの入力、更新、削除をより柔軟に行えるようになりました。新たに導入された演算子では、ある属性が存在するかしないかや、特定の値より大きいか小さいかを調べることができます。また、2 つ値の範囲内にあるかや、特定の文字で開始されるものであるかどうかを含め、その他多くの条件を判断することが可能です。DynamoDB では、オプションで OR 演算子も用意されており、多種の条件を評価するために使用できます デフォルトでは、式内の複数の条件は AND 演算が行われるため、それらのすべての条件が true になる場合にのみ、式は true と評価されます。代わりに OR を指定すると、少なくとも 1 つ の条件が true になる場合に、式は true と評価されます。詳細については、「項目と属性の操作」を参照してください。	2014 年 4 月 24 日

変更	説明	変更日
クエリフィルター	DynamoDB の Query API で、新しい QueryFilter オプションがサポートされました。デフォルトでは、Query は、特定のパーティションキー値とオプションのソートキー条件 (省略可能) に一致する項目を検索します。Query フィルターはキー以外の属性 (非キー属性) に条件式を適用します。Query フィルターがある場合は、フィルター条件に一致しない項目が除外されてから、Query の結果がアプリケーションに返されます。詳細については、「DynamoDB のクエリオペレーション」を参照してください。	2014 年 4 月 24 日

変更	説明	変更日
AWS Management Console を使用したデータのエクスポートとインポート	DynamoDB コンソールが改良され、DynamoDB テーブルのデータのエクスポートとインポートが簡略化されました。数回だけのクリックで AWS Data Pipeline を設定することで、ワークフローのオーケストレーションが可能です。また、Amazon Elastic MapReduce クラスターを設定することで、DynamoDB テーブルから Amazon S3 バケットへの (またはその逆方向での) データのコピーも実行できます。エクスポートまたはインポートを 1 回のみ実行するか、日次エクスポートジョブを設定することができます。さらに、クロスリージョンのエクスポートとインポートを実行して、DynamoDB データを 1 つの AWS リージョン内のテーブルから 別の AWS リージョン内のテーブルにコピーすることもできます。詳細については、「AWS Data Pipeline を使用して DynamoDB データをエクスポートおよびインポートする」を参照してください。	2014 年 3 月 6 日

変更	説明	変更日
高レベル API のドキュメントの再構成	次の API に関する情報が見つけやすくなりました。 • Java: DynamoDBMappper • .NET ドキュメントモデルとオブジェクト永続性モデル この高レベル API は、「DynamoDB の上位レベルのプログラミングインターフェイス」に記載されています。	2014 年 1 月 20 日

変更	説明	変更日
グローバルセカンダリインデックス	DynamoDB に、グローバルセカンダリインデックスのサポートが追加されました。ローカルセカンダリインデックスと同様、テーブルからの代替キーを使用して、インデックスに対してクエリリクエストを発行することで、グローバルセカンダリインデックスを定義します。ローカルセカンダリインデックスとは異なり、グローバルセカンダリインデックスのパーティションキーは、テーブルのパーティションキーと同じである必要がありません。テーブルからの任意のスカラー属性にすることができます。ソートキーはオプションです。このキーも任意のスカラーテーブル属性にすることができます。グローバルセカンダリインデックスにも独自のプロビジョニングされたスループット設定があり、親テーブルの設定から独立しています。詳細については、「セカンダリインデックスを使用したデータアクセス性の向上」および「DynamoDB のグローバルセカンダリインデックスの使用」を参照してください。	2013 年 12 月 12 日

変更	説明	変更日
きめ細かなアクセス制御	DynamoDB に、きめ細かなアクセス制御のサポートが追加されました。この機能を使用するお客様は、DynamoDB テーブルまたはセカンダリインデックスに置かれた、個別の項目と属性にアクセスできるプリンシパル (ユーザー、グループ、またはロール) を指定できます。また、アプリケーションでウェブアイデンティティフェデレーションを利用することで、Facebook、Google、Login with Amazon などサードパーティのアイデンティティプロバイダーに、ユーザー認証に関するタスクを任せることができます。こうすることで、アプリケーション (モバイルアプリケーションを含む) は、非常に多くのユーザーを処理できると同時に、権限のないユーザーによる DynamoDB のデータ項目へのアクセスを防ぐことができます。詳細については、「詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用」を参照してください。	2013 年 10 月 29 日

変更	説明	変更日
4 KB の読み込みキャパシティーユニットのサイズ	読み込み用のキャパシティーユニットのサイズが 1 KB から 4 KB に増加しました。この拡張により、多くのアプリケーションで必要な、プロビジョニングされる読み込みキャパシティーユニットの数が少なくなります。たとえば、これより前のリリースでは、10 KB の項目を読み取る場合に 10 個の読み込みキャパシティーユニットが消費されていました。現在のリリースでは、同じ 10 KB の読み込みでも消費されるのは 3 ユニット (10 KB / 4 KB、次の 4 KB 境界に切り上げ) だけです。詳細については、「DynamoDB のスループットキャパシティ」を参照してください。	2013 年 5 月 14 日

変更	説明	変更日
並列スキャン	DynamoDB に、並列スキャンオペレーションのサポートが追加されました。各アプリケーションでは、1 つのテーブルを論理的なセグメントに分割して、すべてのセグメントを同時にスキャンすることができます。この機能によって、スキャンが完了するまでの時間が短縮され、テーブルにプロビジョニングされた読み込みキャパシティーが完全に利用されます。詳細については、「DynamoDB でのスキャンの使用」を参照してください。	2013 年 5 月 14 日
ローカルセカンダリインデックス	DynamoDB に、ローカルセカンダリインデックスのサポートが追加されました。非キー属性にソートキーインデックスを定義し、それらのインデックスをクエリリクエストで使用することができます。アプリケーションがローカルセカンダリインデックスを使って、複数のディメンション全体でデータの項目を効率的に取得することが可能になります。詳細については、「ローカルセカンダリインデックス」を参照してください。	2013 年 4 月 18 日

変更	説明	変更日
API の新バージョン	このリリースでは、新しいバージョンの API (2012-08-10) が DynamoDB に導入されました。以前のバージョンの API (2011-12-05) も、既存のアプリケーションとの下位互換性のために引き続きサポートされています。新しいアプリケーションでは、最新の API バージョン 2012-08-10 を使用してください。DynamoDB の新機能 (ローカルセカンダリインデックスなど) は以前の API バージョンにバックポートされないため、既存のアプリケーションでご使用の API バージョンを、2012-08-10 に移行することをお勧めします。API バージョン 2012-08-10 の詳細については、Amazon DynamoDB API リファレンスを参照してください。	2013 年 4 月 18 日

変更	説明	変更日
IAM ポリシー変数のサポート	IAM アクセスポリシー言語では、変数がサポートされるようになりました。ポリシーが評価されると、認証ユーザーのセッションから得たコンテキストベースの情報に基づく値にポリシー変数が置き換えられます。ポリシー変数を使用すれば、ポリシーのすべての構成要素を明示的にリストしなくても、汎用的なポリシーを定義できます。ポリシー変数の詳細については、AWS Identity and Access Management での IAM の使用ガイドで「ポリシー変数」を参照してください。 DynamoDB のポリシー変数の例については、「Amazon DynamoDB の Identity and Access Management」でご確認ください。	2013 年 4 月 4 日
AWS SDK for PHP バージョン 2 用に更新された PHP コード例	AWS SDK for PHP のバージョン 2 が使用可能となりました。Amazon DynamoDB デベロッパーガイドに示した PHP のコード例も、この新しい SDK に対応するように更新されています。SDK バージョン 2 の詳細については、「AWS SDK for PHP」を参照してください。	2013 年 1 月 23 日

変更	説明	変更日
新しいエンドポイント	DynamoDB が AWS GovCloud (米国西部) リージョンに拡張されました。現在のサービスエンドポイントとプロトコルのリストについては、リージョンとエンドポイントに関する章を参照してください。	2012 年 12 月 3 日
新しいエンドポイント	DynamoDB が南米 (サンパウロ) リージョンに拡張されました。現在サポートされているエンドポイントのリストについては、リージョンとエンドポイントに関する章を参照してください。	2012 年 12 月 3 日
新しいエンドポイント	DynamoDB がアジアパシフィック (シドニー) リージョンに拡張されました。現在サポートされているエンドポイントのリストについては、リージョンとエンドポイントに関する章を参照してください。	2012 年 11 月 13 日

変更	説明	変更日
DynamoDB で CRC32 チェックサムのサポートが実装され、強力な整合性のあるバッチ取得のサポート導入と、テーブルの同時更新に対する制限の廃止が行われました。	• HTTP ペイロードの CRC32 チェックサムが、DynamoDB により計算され、このチェックサムが新しいヘッダー x-amz-crc32 に返されます。詳細については、「DynamoDB 低レベル API」を参照してください。 • デフォルトでは、BatchGetItem API が実行する読み込みオペレーションは、結果整合性がとられます。ConsistentRead の新しい BatchGetItem パラメータを使用すれば、リクエスト内の任意のテーブルに対して、強力な読み込み整合性を選択できます。詳細については、「説明」を参照してください。 • このリリースでは、多数のテーブルを同時に更新する場合の制限がいくつか廃止されています。一度に更新できるテーブルの合計数は 10 のままですが、これらのテーブルのステータスについては、CREATING、UPDATING、または DELETING を任意に組み合わせることが可能になりました。さらに、1 つ	2012 年 11 月 2 日

変更	説明	変更日
	のテーブルの ReadCapacityUnits または WriteCapacityUnits を増減できる最小値の制限もなくなりました。詳細について、Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ を参照してください。	
ベストプラクティスに関するドキュメント	Amazon DynamoDB デベロッパーガイド では、テーブルや項目の使用に関するベストプラクティスを紹介しています。加えて、クエリおよびスキャンオペレーションに関する推奨事項も示しています。	2012 年 9 月 28 日

変更	説明	変更日
バイナリデータ型のサポート	Number および String 型に加えて、DynamoDB で バイナリデータ型がサポートされるようになりました。 これより前のリリースでは、バイナリデータを DynamoDB に格納する際には、そのデータを文字列 (String) 形式に変換していました。クライアント側で変換作業が必要なだけでなく、変換によってデータ項目のサイズが大きくなり、より多くのストレージが必要になり、スループットキャパシティを追加でプロビジョニングしなければならない場合もありました。 バイナリタイプの属性を使用することで、圧縮データ、暗号化データ、イメージなど、任意のバイナリデータを格納できるようになりました。詳細については、「データ型」を参照してください。AWS SDK を使用してバイナリタイプのデータを処理する実例については、以下のセクションを参照してください。 • 例: AWS SDK for Java ドキュメント API を使用したバイナリタイプ属性の処理	2012 年 8 月 21 日

変更	説明	変更日
	- 例: AWS SDK for .NET 低レベル API を使用したバイナリタイプ属性の処理 AWS SDK でサポートされるバイナリデータのタイプを追加するには、最新の SDK をダウンロードする必要があります。既存のアプリケーションの更新が必要になる場合もあります。AWS SDK のダウンロードの詳細については、「.NET コード例」を参照してください。	
DynamoDB テーブルの項目を DynamoDB コンソールを使用して、更新およびコピーできるようになりました。	DynamoDB のユーザーは、テーブル項目の追加と削除に加えて、その更新とコピーも DynamoDB コンソールから行えるようになりました。この新機能によって、コンソールを使用した個々の項目の変更が容易になりました。	2012 年 8 月 14 日

変更	説明	変更日
DynamoDB で最小テーブルスループットの要件が引き下げられました。	DynamoDB の最小テーブルスループットの要件が引き下げられ、書き込みキャパシティーユニットおよび読み込みキャパシティーユニットでは、最低 1 個までがサポートされるようになりました。詳細については、Amazon DynamoDB デベロッパーガイドの「Amazon DynamoDB のサービス、アカウント、およびテーブルのクォータ」トピックを参照してください。	2012 年 8 月 9 日
Signature Version 4 のサポート	DynamoDB で、リクエストの認証用に署名バージョン 4 がサポートされるようになりました。	2012 年 7 月 5 日
DynamoDB コンソールでテーブルエクスプローラーが使用可能になりました。	DynamoDB コンソールで、テーブル内のデータの参照とクエリが可能な、テーブルエクスプローラーがサポートされるようになりました。新しい項目の挿入や既存の項目の削除も可能です。DynamoDB でのコード例用のテーブルの作成とデータのロード および コンソールを使用する場合 セクションは、これらの機能を反映して更新されています。	2012 年 5 月 22 日

変更	説明	変更日
新しいエンドポイント	DynamoDB の可用性が、米国西部 (北カリフォルニア) リージョン、米国西部 (オレゴン) リージョン、アジアパシフィック (シンガポール) リージョンの新しいエンドポイントに拡張されました。 現在サポートされているエンドポイントのリストについては、リージョンとエンドポイントに関する章を参照してください。	2012 年 4 月 24 日
BatchWriteItem API のサポート	DynamoDB でバッチ書き込み API がサポートされ、1 回の API 呼び出しによって、1 つ以上のテーブルでの複数の項目の入力と削除ができるようにました。DynamoDB のバッチ書き込み API の詳細については、「BatchWriteItem」を参照してください。 AWS SDK による項目の操作とバッチ書き込み機能の詳細については、「項目と属性の操作」および「.NET コード例」を参照してください。	2012 年 4 月 19 日
エラーコードの追加	詳細については、「DynamoDB でのエラー処理」を参照してください。	2012 年 4 月 5 日

変更	説明	変更日
新しいエンドポイント	DynamoDB がアジアパシフィック (東京) リージョンに拡張されました。現在サポートされているエンドポイントのリストについては、リージョンとエンドポイントに関する章を参照してください。	2012 年 2 月 29 日
ReturnedItemCount メトリクスの追加	新しいメトリクス ReturnedItemCount により、クエリからいくつかの項目を返せるようになり、同時に、DynamoDB に対するスキャンオペレーションを CloudWatch を通じてモニタリングできるようになりました。	2012 年 2 月 24 日
値を増加させる例を追加	DynamoDB で、既存の数値の増減が可能になりました。以下のドキュメントの「項目の更新」セクションでは、既存の値を増加させる例を示しています。 項目の操作: Java. 項目の操作: .NET.	2012 年 1 月 25 日
初回製品リリース	DynamoDB が、新たにベータリリースのサービスとして導入されました。	2012 年 1 月 18 日

DynamoDB のレガシー機能

以下のトピックは、DynamoDB が現在もサポートしているレガシー機能です。これらの機能については活発な開発は行われていません。

トピック

- グローバルテーブルバージョン 2017.11.29 (レガシー)

グローバルテーブルバージョン 2017.11.29 (レガシー)

> ⚠ Important
>
> このドキュメントはグローバルテーブルのバージョン 2017.11.29 (レガシー) を対象としています。新しいグローバルテーブルでは使用しないでください。可能な限り、グローバルテーブルバージョン 2019.11.21 (現行) を使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティが少ないためです。
>
> ご使用のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。既存のグローバルテーブルを 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) に更新する方法については、「グローバルテーブルのアップグレード」を参照してください。

トピック

- グローバルテーブル: 仕組み
- グローバルテーブルを管理するためのベストプラクティスと要件
- グローバルテーブルの作成
- グローバルテーブルのモニタリング
- グローバルテーブルで IAM を使用します

グローバルテーブル: 仕組み

> ⚠ Important
>
> このドキュメントはグローバルテーブルのバージョン 2017.11.29 (レガシー) を対象としています。新しいグローバルテーブルでは使用しないでください。可能な限り、グローバルテー

ブルバージョン 2019.11.21 (現行) を使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティが少ないためです。
ご使用のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。既存のグローバルテーブルを 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) に更新する方法については、「グローバルテーブルのアップグレード」を参照してください。

以下のセクションでは、Amazon DynamoDB におけるグローバルテーブルの概念および動作を理解することができます。

バージョン 2017.11.29 (レガシー) のグローバルテーブルの概念

グローバルテーブルは 1 つ以上のレプリカテーブルの集合体であり、すべて単一の AWS アカウントが所有します。

レプリカテーブル (または、略してレプリカ) は、グローバルテーブルの一環として機能する単一の DynamoDB テーブルです。各レプリカには、同じデータ項目のセットが保存されます。指定のグローバルテーブルは、AWS リージョンごとに 1 つのレプリカテーブルのみを持つことができます。

以下は、グローバルテーブルの作成方法における概念的な概要です。

1. AWS リージョンで、DynamoDB Streams を有効にしてある通常の DynamoDB テーブルを作成します。
2. データをレプリケートする他のすべてのリージョンに対して、ステップ 1 を繰り返します。
3. 作成したテーブルに基づいて DynamoDB グローバルテーブルを定義します。

AWS Management Console はこれらのタスクを自動化するため、グローバルテーブルをより迅速かつ簡単に作成できます。詳細については、「」を参照してくださいグローバルテーブルの作成

結果の DynamoDB グローバルテーブルは、DynamoDB が単一の単位として扱う複数のレプリカテーブル (リージョンごとに 1 つ) で構成されます。すべてのレプリカは、同じテーブル名と同じプライマリキーのスキーマを持っています。アプリケーションが 1 つのリージョンのレプリカテーブルにデータを書き込むと、DynamoDB はその書き込みを他の AWS リージョンの他のレプリカテーブルに自動的に伝搬します。

⚠ Important

テーブルデータの同期を維持するために、グローバルテーブルでは、すべての項目に対して次の属性を自動的に作成します。

- `aws:rep:deleting`
- `aws:rep:updatetime`
- `aws:rep:updateregion`

これらの属性を変更したり、同じ名前の属性を作成したりしないでください。

レプリカテーブルをグローバルテーブルに追加して、追加のリージョンで使用できるようにすることができます。(これを行うには、グローバルテーブルが空である必要があります。つまり、どのレプリカテーブルにもデータを含めることはできません。)

グローバルテーブルからレプリカテーブルを削除することもできます。これを行うと、テーブルはグローバルテーブルから完全に関連付けが解除されます。この新しく独立したテーブルは、グローバルテーブルと相互作用しなくなり、データはグローバルテーブルとの間で伝播されなくなります。

⚠ Warning

レプリカの削除がアトミックプロセスではないことに注意してください。動作と既知の状態に確実に一貫させるために、削除されるレプリカからアプリケーションの書き込みトラフィックを事前に切り離しておくことをお勧めします。削除後、すべてのレプリカリージョンのエンドポイントでレプリカが関連付け解除されたものとして表示されるのを待ってから、レプリカへのそれ以降の書き込みを独自の分離リージョンテーブルとして実行してください。

一般的なタスク

グローバルテーブルの一般的なタスクは以下のように機能します。

グローバルテーブルのレプリカテーブルは、通常のテーブルと同じように削除できます。これにより、そのリージョンへのレプリケーションが停止し、そのリージョンに保存されているテーブルコピーが削除されます。レプリケーションを切断して、テーブルのコピーを独立したエンティティとすることはできません。

> **Note**
>
> ソーステーブルは、新しいリージョンの作成に使用されてから 24 時間以上経たないと削除できません。削除を試みるのが早すぎると、エラーが発生します。

アプリケーションが異なるリージョンにある同一項目をほぼ同時に更新すると、競合が発生する可能性があります。結果整合性を確保するために、DynamoDB グローバルテーブルでは同時更新間の調整のために、「最新書き込み優先」方法が使用されます。すべてのレプリカが最新の更新に合意し、すべてのレプリカが同一のデータを持つ状態に収束します。

> **Note**
>
> 競合を回避する方法は、以下を含めていくつかあります。
>
> - IAM ポリシーを使用して、1 つのリージョンのテーブルへの書き込みのみを許可する。
> - IAM ポリシーを使用してユーザーを 1 つのリージョンのみにルーティングし、もう 1 つはアイドル状態のスタンバイのままにする。または、奇数のユーザーをあるリージョンに、偶数のユーザーを別のリージョンにルーティングする。
> - Bookmark = Bookmark + 1 のような非冪等の更新は避け、Bookmark=25 のような静的な更新を優先する。

グローバルテーブルのモニタリング

CloudWatch を使用して、`ReplicationLatency` メトリクスをモニタリングできます。このメトリクスは、更新された項目が 1 つのレプリカテーブルの DynamoDB ストリームに表示されてから、その項目がグローバルテーブルの別のレプリカに表示されるまでの経過時間を追跡します。`ReplicationLatency` はミリ秒単位で表され、すべての送信元リージョンと送信先リージョンのペアに対して出力されます。これは、グローバルテーブル v2 が提供する唯一の CloudWatch メトリクスです。

表示されるレイテンシーは、選択したリージョン間の距離やその他の変動要素によって異なります。リージョンのレイテンシーが 0.5～2.5 秒の範囲で発生するのは、同じ地域内でもよくあることです。

有効期限 (TTL)

Time To Live (TTL) を使用して、項目の有効期限を示す値を含む属性名を指定できます。この値は、Unix エポックの開始からの秒数として指定されます。

グローバルテーブルのレガシーバージョンでは、TTL 削除は他のレプリカに自動的に複製されません。TTL ルールによって項目が削除された場合、削除は書き込みユニットを消費することなく実行されます。

ソーステーブルとターゲットテーブルのプロビジョニングされた書き込みキャパシティーが非常に少ない場合、TTL 削除には書き込みキャパシティーが必要となるため、スロットリングが発生する可能性があります。

グローバルテーブルでのストリームとトランザクション

各グローバルテーブルは、書き込みの開始点に関係なく、すべての書き込みに基づいて独立したストリームを生成します。この DynamoDB ストリームを 1 つのリージョンで使用するか、すべてのリージョンで個別に使用するかを選択できます。

レプリケートされた書き込みではなくローカル書き込みを処理する場合は、各項目に独自のリージョン属性を追加できます。次に、Lambda イベントフィルターを使用して、ローカルリージョンへの書き込みに対して Lambda のみを呼び出すことができます。

トランザクションオペレーションは、書き込みが最初に行われたリージョン内でのみ、不可分性、一貫性、分離性、耐久性 (ACID) を保証します。グローバルテーブルのリージョン間では、トランザクションはサポートされていません。

例えば、米国東部 (オハイオ) および米国西部 (オレゴン) リージョンにレプリカを持つグローバルテーブルがあり、米国東部 (オハイオ) リージョンで TransactWriteItems オペレーションを実行する場合、変更がレプリケートされると米国西部 (オレゴン) ではトランザクションが部分的に完了される場合があります。変更は、ソースリージョンでコミットされると、他のリージョンにのみレプリケートされます。

ⓘ Note

- グローバルテーブルは DynamoDB を直接更新することで、DynamoDB Accelerator を「書き込み迂回」します。その結果、DAX は古いデータを保持していることを認識しません。DAX キャッシュは、キャッシュの TTL が期限切れになったときにのみ更新されます。

- グローバルテーブルのタグは自動的には伝播されません。

読み取りおよび書き込みスループット

グローバルテーブルは、次の方法で読み取りと書き込みのスループットを管理します。

- 書き込みキャパシティーは、リージョン間のすべてのテーブルインスタンスで同じでなければなりません。
- バージョン 2019.11.21 (現行) では、テーブルが自動スケーリングをサポートするように設定されているか、オンデマンドモードの場合、書き込みキャパシティーは自動的に同期されます。各リージョンで現在プロビジョニングされている書き込みキャパシティーは、同期された自動スケーリング設定の範囲内で個別に増減します。テーブルをオンデマンドモードにすると、そのモードは他のレプリカと同期します。
- 読み取りキャパシティーはリージョンによって異なる場合があります。これは、読み取りキャパシティーが等しくない場合があるためです。グローバルレプリカをテーブルに追加すると、ソースリージョンのキャパシティーが反映されます。作成後、1 つのレプリカの読み込みキャパシティーを調整できますが、この新しい設定はもう 1 つのレプリカには転送されません。

整合性と競合の解決

レプリカテーブル内の項目に加えられた変更は、同じグローバルテーブル内の他のすべてのレプリカにレプリケートされます。グローバルテーブルでは、新しく書き込まれた項目は通常、数秒以内にすべてのレプリカテーブルに伝播されます。

グローバルテーブルでは、各レプリカテーブルには同じデータ項目のセットが保存されます。DynamoDB では、一部の項目のみの部分的なレプリケーションはサポートされていません。

アプリケーションは、任意のレプリカテーブルとの間でデータを読み込みおよび書き込みできます。DynamoDB は、リージョン間での結果整合性にある読み込みに対応していますが、リージョン間での強力な整合性のある読み込みには対応していません。アプリケーションが結果整合性のある読み込みのみを使用し、1 つの AWS リージョンに対して読み込みのみを発行する場合、変更を加えずに動作します。ただし、アプリケーションで強力な整合性のある読み込みが必要な場合は、同じリージョンですべての強力な整合性のある読み込みと書き込みを実行する必要があります。そうではなく、あるリージョンに書き込んで別のリージョンから読み込む場合、読み込みのレスポンスには、他のリージョンで最近完了した書き込みの結果を反映していない古いデータが含まれる可能性があります。

アプリケーションが異なるリージョンにある同一項目をほぼ同時に更新すると、競合が発生する可能性があります。結果整合性を確保するために、DynamoDB グローバルテーブルでは最終書き込み者優先を使用して、同時更新間の調整を行い、DynamoDB は最終書き込み者を判断するために最善を尽くします。この競合解決メカニズムでは、すべてのレプリカが最新の更新に合意し、すべてのレプリカが同一のデータを持つ状態に収束します。

可用性と耐久性

単一の AWS リージョンが分離または低下した場合、アプリケーションは別のリージョンにリダイレクトし、別のレプリカテーブルに対して読み込みと書き込みを実行できます。カスタムビジネスロジックを適用して、リクエストを他のリージョンにリダイレクトするタイミングを決定できます。

リージョンが分離または縮退した場合、DynamoDB は、実行されたが、まだすべてのレプリカテーブルに反映されていない書き込みを追跡します。リージョンがオンラインに戻ると、DynamoDB はそのリージョンから他のリージョンのレプリカテーブルへの保留中の書き込みの伝播を再開します。また、他のレプリカテーブルから現在オンラインに戻っているリージョンへの書き込みの伝播も再開します。リージョンの分離期間がどれほど長くても、それまでに成功した書き込みはすべて最終的に反映されます。

グローバルテーブルを管理するためのベストプラクティスと要件

> ⚠ Important
>
> このドキュメントはグローバルテーブルのバージョン 2017.11.29 (レガシー) を対象としています。新しいグローバルテーブルでは使用しないでください。可能な限り、グローバルテーブルバージョン 2019.11.21 (現行) を使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティが少ないためです。
> ご使用のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。既存のグローバルテーブルを 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) に更新する方法については、「グローバルテーブルのアップグレード」を参照してください。

Amazon DynamoDB グローバルテーブルを使用すると、テーブルデータを AWS リージョン間でレプリケートできます。データが適切にレプリケートされるように、グローバルテーブル内のレプリカテーブルとセカンダリインデックスにも、書き込みキャパシティーが同じように設定されていることが重要です。

トピック

- グローバルテーブルのバージョン
- 新しいレプリカテーブルを追加するための要件
- キャパシティを管理するためのベストプラクティスと要件

グローバルテーブルのバージョン

DynamoDB グローバルテーブルには、グローバルテーブルバージョン 2019.11.21 (現行) と グローバルテーブルバージョン 2017.11.29 (レガシー) の 2 つのバージョンがあります。可能な限り、グローバルテーブルバージョン 2019.11.21 (現行) を使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティが少ないためです。

ご使用のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。既存のグローバルテーブルを 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) に更新する方法については、「グローバルテーブルのアップグレード」を参照してください。

新しいレプリカテーブルを追加するための要件

新しいレプリカテーブルをグローバルテーブルに追加するには、以下のすべての条件が満たされている必要があります。

- その他すべてのレプリカと同じパーティションキーがテーブルに含まれている。
- テーブルは、同じ指定した書き込みキャパシティー管理設定である必要があります。
- テーブルの名前が、その他すべてのレプリカと同じ名前である。
- 項目の新しいイメージと古いイメージの両方を含むストリーミングで、テーブルの DynamoDB Streams が有効になっている。
- このグローバルテーブルの新規または既存のレプリカテーブルにはデータを含むことはできません。

グローバルセカンダリインデックスが指定されている場合は、次の条件も満たす必要があります。

- グローバルセカンダリインデックスにも同じ名前が設定されている。
- グローバルセカンダリインデックスのパーティションキーとソートキー (存在する場合) にも同じ名前が設定されている。

⚠ Important

書き込みキャパシティーの設定は、すべてのグローバルテーブルのレプリカテーブルおよび一致するセカンダリインデックスに対して一貫性があるように設定する必要があります。グローバルテーブルの書き込み容量を更新するには、DynamoDB コンソールまたは `UpdateGlobalTableSettings` API `UpdateGlobalTableSettings`オペレーションを使用することを強くお勧めします。[585] は、自動的にグローバルテーブル内のすべてのレプリカテーブルおよび一致するセカンダリインデックスに書き込み容量の設定の変更を適用します。`UpdateTable`、`RegisterScalableTarget`、あるいは `PutScalingPolicy` オペレーションを使用する場合には、各レプリカテーブルおよび一致するセカンダリインデックスに個別に変更を適用する必要があります。詳細については、「Amazon DynamoDB API リファレンス」の「UpdateGlobalTableSettings」を参照してください。

プロビジョニングされた書き込みキャパシティー設定を管理するために、Auto Scaling を有効にすることが強く推奨されます。書き込みキャパシティー設定を手動で管理する場合は、すべてのレプリカテーブルと同等のレプリケートされた書き込みキャパシティーユニットをプロビジョニングする必要があります。また、グローバルテーブル全体で一致するセカンダリインデックスには、レプリケートされた同等の書き込みキャパシティーユニットをプロビジョニングします。

また、適切な AWS Identity and Access Management (IAM) 権限も必要です。詳細については、「グローバルテーブルで IAM を使用します」を参照してください。

キャパシティを管理するためのベストプラクティスと要件

DynamoDB でレプリカテーブルの容量設定を管理するときは、以下の点を考慮に入れてください。

DynamoDB Auto Scaling の使用

プロビジョンされたモードを使用するレプリカテーブルのスループット容量設定を管理するには、DynamoDB Auto Scaling の使用をお勧めします。DynamoDB Auto Scaling は各レプリカテーブルの読み込み容量単位 (RCU) および書き込み容量単位 (WCU) を、実際のアプリケーションワークロードに基づいて、自動的に調整します。詳細については、「DynamoDB Auto Scaling によるスループットキャパシティの自動管理」を参照してください。

AWS Management Console を使用してレプリカテーブルを作成すると、Auto Scaling が各レプリカテーブルに対してデフォルトで有効になります。この際、読み込みキャパシティーユニットおよび書き込みキャパシティーユニットを管理するデフォルトの Auto Scaling 設定が使用されます。

DynamoDB コンソールを介して、あるいは UpdateGlobalTableSettings コールを使用して、レプリカテーブルあるいはセカンダリインデックスの Auto Scaling 設定に行う変更は、グローバルテーブルのすべてのレプリカテーブルおよび一致するセカンダリインデックスに対して自動的に適用されます。これらの変更により、すべての既存の Auto Scaling 設定が上書きされます。これによって、プロビジョニングされた書き込みキャパシティーは、グローバルテーブルのすべてのレプリカテーブルおよび一致するセカンダリインデックスを通して一貫性があるようになります。UpdateTable、RegisterScalableTarget、あるいは PutScalingPolicy 呼び出しを使用する場合には、各レプリカテーブルおよび一致するセカンダリインデックスに個別に変更を適用する必要があります。

> Note
>
> Auto Scaling がアプリケーションの容量の変更を満たさない場合 (予測不可能なワークロード) や、その設定を行わない場合 (最小、最大、または使用率しきい値のターゲット設定)、オンデマンドモードを使用してグローバルテーブルの容量を管理できます。詳細については、「オンデマンドモード」を参照してください。
> グローバルテーブルでオンデマンドモードを有効にした場合、レプリケートされた書き込みリクエストユニット (rWCU) の消費は、rWCU のプロビジョニング方法と整合します。たとえば、2 つの追加リージョンにレプリケートされるローカルテーブルに対して書き込みを 10 回実行した場合、書き込みリクエストユニットが 60 個消費されます (10 + 10 + 10 = 30、30 x 2 = 60)。消費される 60 個の書き込みリクエストユニットには、aws:rep:deleting、aws:rep:updatetime、および aws:rep:updateregion 属性を更新するためにグローバルテーブルバージョン 2017.11.29 (レガシー) で消費される追加の書き込みが含まれます。

手動によるキャパシティ管理

DynamoDB Auto Scaling を使用しない場合は、各レプリカテーブルおよびセカンダリインデックスの読み込み容量および書き込み容量を手動で設定する必要があります。

すべてのレプリカテーブルでレプリケートされたプロビジョンド書き込みキャパシティーユニット (rWCU) は、全リージョンにおけるアプリケーションの書き込みに必要な rWCU 合計数の 2 倍に設定する必要があります。これにより、ローカルリージョンで発生するアプリケーションの書き込みと、他のリージョンからのレプリケートされたアプリケーションの書き込みに対応できます。たとえば、オハイオ州のレプリカテーブルに 1 秒間に 5 回の書き込みを、バージニア北部のレプリカテーブルに 1 秒間に 5 回の書き込みを行うとします。この場合、各レプリカテーブルに 20 個の rWCU をプロビジョニングする必要があります (5 + 5 = 10、10×2 = 20)。

グローバルテーブルの書き込み容量を更新するには、DynamoDB コンソールまたは `UpdateGlobalTableSettings` API `UpdateGlobalTableSettings`オペレーションを使用することを強くお勧めします。[585] は、自動的にグローバルテーブル内のすべてのレプリカテーブルおよび一致するセカンダリインデックスに書き込み容量の設定の変更を適用します。`UpdateTable`、`RegisterScalableTarget`、あるいは `PutScalingPolicy` オペレーションを使用する場合には、各レプリカテーブルおよび一致するセカンダリインデックスに個別に変更を適用する必要があります。詳細については、Amazon DynamoDB API リファレンスを参照してください。

> Note
>
> DynamoDB のグローバルテーブルの設定 (`UpdateGlobalTableSettings`) を更新するには、`dynamodb:UpdateGlobalTable`、`dynamodb:DescribeLimits`、`application-autoscaling:DeleteScalingPolicy` および `application-autoscaling:DeregisterScalableTarget` の許可が必要です。詳細については、「グローバルテーブルで IAM を使用します」を参照してください。

グローバルテーブルの作成

> Important
>
> このドキュメントはグローバルテーブルのバージョン 2017.11.29 (レガシー) を対象としています。新しいグローバルテーブルでは使用しないでください。可能な限り、グローバルテーブルバージョン 2019.11.21 (現行) を使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティが少ないためです。
> ご使用のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。既存のグローバルテーブルを 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) に更新する方法については、「グローバルテーブルのアップグレード」を参照してください。

このセクションでは、Amazon DynamoDB コンソールまたは AWS Command Line Interface (AWS CLI) を使用してグローバルテーブルを作成する方法について説明します。

トピック

- グローバルテーブルの作成 (コンソール)

- グローバルテーブル (AWS CLI) の作成

グローバルテーブルの作成 (コンソール)

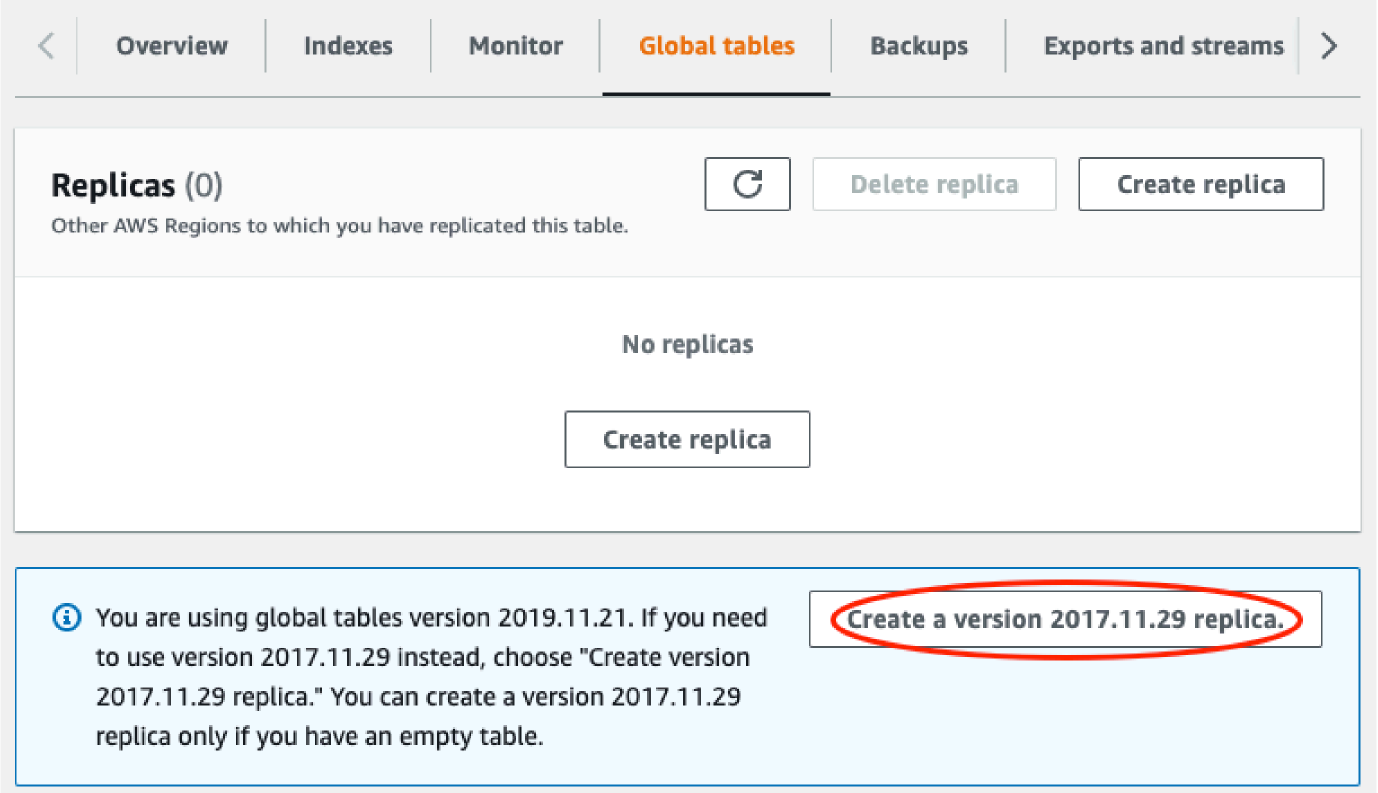
コンソールを使用してグローバルテーブルを作成するには、次の手順に従います。以下の例では、レプリカテーブルを持つグローバルテーブルを米国およびヨーロッパに作成します。

1. DynamoDB コンソール (https://console.aws.amazon.com/dynamodb/home) を開きます。この例では、us-east-2 (米国東部オハイオ) リージョンを選択します。
2. コンソールの左側のナビゲーションペインで、[テーブル] を選択します。
3. [テーブルの作成] を選択します。

   [テーブル名] に「**Music**」と入力します。

   [プライマリキー] に **Artist** と入力します。[Add sort key (ソートキーの追加)] を選択し、**SongTitle** と入力します (**Artist** と **SongTitle** はいずれも文字列で入力します)。

   テーブルを作成するには、[Create (作成)] を選択します。このテーブルは、新しいグローバルテーブルの最初のレプリカテーブルとして機能します。これは、後で追加する他のレプリカテーブルのプロトタイプです。
4. [グローバルテーブル] タブを選択後、[バージョン 2017.11.29 (レガシー) レプリカを作成] を選択します。



You are using global tables version 2019.11.21. If you need to use version 2017.11.29 instead, choose "Create version 2017.11.29 replica." You can create a version 2017.11.29 replica only if you have an empty table.

5. [利用できるレプリケーションリージョン] ドロップダウンで、米国西部 (オレゴン) を選択します。

   コンソールは、選択したリージョンに同一名のテーブルが存在しないことを確認します。同一名のテーブルが存在する場合は、そのリージョンで新しいレプリカテーブルを作成する前に既存のテーブルを削除する必要があります。

6. [レプリカを作成] を選択します。これにより、米国西部 (オレゴン) でテーブル作成プロセスが開始されます。

   選択したテーブル (およびその他すべてのレプリカテーブル) の [グローバルテーブル] タブに、そのテーブルが複数のリージョンでレプリケートされたことが示されます。

7. ここで、別のリージョンを追加して、グローバルテーブルがレプリケートされ、米国およびヨーロッパに同期されるようにします。これを行うには、ステップ 5 を繰り返しますが、今回は米国西部 (オレゴン) の代わりに欧州 (フランクフルト) を指定します。

8. 引き続き、米国東部 (オハイオ)リージョンで AWS Management Console を使用する必要があります。左のナビゲーションメニューで [項目] を選択し、[音楽] テーブルを選択してから、[項目を作成] を選択します。

   a. [Artist] に「**item_1**」と入力します。

b. [SongTitle] に「**Song Value 1**」と入力します。

c. 項目を書き込むには、[項目を作成] を選択します。

9. 間もなくすると、この項目は、グローバルテーブルの 3 つのすべてのリージョンにレプリケートされます。これを確認するには、コンソールの右上隅にあるリージョンセレクターで、[Europe (Frankfurt) (欧州 (フランクフルト))] を選択します。欧州 (フランクフルト) の Music テーブルに新しい項目が追加されます。

10. ステップ 9 を繰り返し、米国西部 (オレゴン) を選択して、そのリージョンでのレプリケーションを確認します。

グローバルテーブル (AWS CLI) の作成

Music を使用してグローバルテーブル AWS CLI を作成するには、次の手順に従います。以下の例では、米国およびヨーロッパのレプリカテーブルを使用して、グローバルテーブルを作成します。

1. DynamoDB Streams を有効にして (Music)、米国東部 (オハイオ) で新規テーブル (NEW_AND_OLD_IMAGES) DynamoDB Streams を作成します。

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. 米国東部 (バージニア北部) に同じ Music テーブルを作成します。

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-1
```

3. us-east-2 リージョンと us-east-1 リージョンのレプリカテーブルで構成されるグローバルテーブル (Music) を作成します。

```
aws dynamodb create-global-table \
    --global-table-name Music \
    --replication-group RegionName=us-east-2 RegionName=us-east-1 \
    --region us-east-2
```

> **Note**
>
> グローバルテーブル名 (Music) は、各レプリカテーブルの名前 (Music) と一致する必要があります。詳細については、「グローバルテーブルを管理するためのベストプラクティスと要件」を参照してください。

4. ステップ 1 とステップ 2 で作成した設定と同じ設定で、欧州 (アイルランド) に別のテーブルを作成します。

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region eu-west-1
```

このステップを実行した後、新しいテーブルを Music グローバルテーブルに追加します。

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
    --region us-east-2
```

5. レプリケーションが機能していることを確認するには、新しい項目を米国東部 (オハイオ) の Music テーブルに追加します。

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. 数秒間待ってから、項目が米国東部 (バージニア北部) および欧州 (アイルランド) に正常にレプリケートされたかどうかを確認します。

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1
```

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

グローバルテーブルのモニタリング

⚠ Important

このドキュメントはグローバルテーブルのバージョン 2017.11.29 (レガシー) を対象としています。新しいグローバルテーブルでは使用しないでください。可能な限り、グローバルテーブルバージョン 2019.11.21 (現行) を使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティが少ないためです。

ご使用のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。既存のグローバルテーブルを 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) に更新する方法については、「グローバルテーブルのアップグレード」を参照してください。

Amazon CloudWatch を使用して、グローバルテーブルの動作とパフォーマンスをモニタリングできます。Amazon DynamoDB は、グローバルテーブルの各レプリカの ReplicationLatency および PendingReplicationCount メトリクスを公開します。

- **ReplicationLatency** - 更新された項目が 1 つのレプリカテーブルの DynamoDB ストリームに表示されてから、その項目がグローバルテーブルの別のレプリカに表示されるまでの経過時間。ReplicationLatency はミリ秒単位で表し、すべての送信元リージョンと送信先リージョンのペアに対して出力されます。

  通常オペレーション中、ReplicationLatency は完全に一定にする必要があります。ReplicationLatency で評価された値は、1 つのレプリカを更新しても、他のレプリカテーブルにタイムリーに伝達されません。これが原因で、アップデートは一貫して受け取ることができないため、時間の経過とともに、他のレプリカテーブルは遅延します。この場合、各レプリカテーブルの読み込みキャパシティーユニット (RCU) と書き込みキャパシティーユニット (WCU) が同一であることを確認する必要があります。また、WCU 設定を選択する場合は、「グローバルテーブルのバージョン」の推奨事項に従います。

  ReplicationLatency は、AWS リージョンのパフォーマンスが低下し、そのリージョンにレプリカテーブルが含まれる場合は増加することがあります。この場合、アプリケーションの読み込みおよび書き込みアクティビティを別の AWS リージョンに一時的にリダイレクトすることができます。

- **PendingReplicationCount** — 1 つのレプリカテーブルに書き込まれているが、グローバルテーブル内の別のレプリカにはまだ書き込まれていない項目の更新数。PendingReplicationCount は項目数で表され、すべての送信元と送信先のリージョンのペアに対して発行されます。

  通常のオペレーションでは、PendingReplicationCount は非常に低いです。PendingReplicationCount が長期間増加する場合は、レプリカテーブルのプロビジョンされた書き込み容量設定が現在のワークロードに対して十分であるかどうかを調査します。

  PendingReplicationCount は、AWS リージョンのパフォーマンスが低下し、そのリージョンにレプリカテーブルが含まれる場合は増加することがあります。この場合、アプリケーションの読み込みおよび書き込みアクティビティを別の AWS リージョンに一時的にリダイレクトすることができます。

詳細については、「DynamoDB のメトリクスとディメンション」を参照してください。

グローバルテーブルで IAM を使用します

> ⚠ Important
>
> このドキュメントはグローバルテーブルのバージョン 2017.11.29 (レガシー) を対象としています。新しいグローバルテーブルでは使用しないでください。可能な限り、グローバルテーブルバージョン 2019.11.21 (現行) を使用してください。2017.11.29 (レガシー) よりも柔軟性と効率が高く、消費する書き込みキャパシティが少ないためです。
>
> ご使用のバージョンを確認するには、「使用しているグローバルテーブルのバージョンを確認する」を参照してください。既存のグローバルテーブルを 2017.11.29 (レガシー) からバージョン 2019.11.21 (現行) に更新する方法については、「グローバルテーブルのアップグレード」を参照してください。

グローバルテーブルを初めて作成する場合は、Amazon DynamoDB によって、AWS Identity and Access Management (IAM) サービスにリンクされたロールが自動的に作成されます。このロールの名前は `AWSServiceRoleForDynamoDBReplication` です。このロールを使用して、お客様に代わって DynamoDB でグローバルテーブルのクロスリージョンレプリケーションを管理できます。このサービスにリンクされたロールは削除しないでください。削除すると、グローバルテーブルはすべて機能しなくなります。

サービスリンクロールの詳細については、「IAM ユーザーガイド」の「サービスリンクロールの使用」を参照してください。

DynamoDB でグローバルテーブルを作成および管理するには、次のそれぞれにアクセスするための `dynamodb:CreateGlobalTable` 許可が必要です。

- 追加するレプリカテーブル。
- すでにグローバルテーブルの一部になっている既存の各レプリカ。
- グローバルテーブル自体。

DynamoDB のグローバルテーブルの設定 (`UpdateGlobalTableSettings`) を更新するには、`dynamodb:UpdateGlobalTable`、`dynamodb:DescribeLimits`、`application-autoscaling:DeleteScalingPolicy` および `application-autoscaling:DeregisterScalableTarget` の許可が必要です。

`application-autoscaling:DeleteScalingPolicy` および `application-autoscaling:DeregisterScalableTarget` 許可は、既存のスケーリングポリシーを更新するときに必要です。これは、新しいポリシーをテーブルまたはセカンダリインデックスに添付する前に、グローバルテーブルサービスが古いスケーリングポリシーを削除できるようにするためです。

IAM ポリシーを使用して 1 つのレプリカテーブルへのアクセスを管理する場合は、そのグローバルテーブル内の他のすべてのレプリカに同じポリシーを適用する必要があります。この方法により、すべてのレプリカテーブルで一貫した許可モデルを維持できます。

グローバルテーブル内のすべてのレプリカで同じ IAM ポリシーを使用することにより、グローバルテーブルデータに対する意図しない読み込みおよび書き込みアクセスを許可しないようにすることもできます。たとえば、グローバルテーブル内の 1 つのレプリカにのみアクセスできるユーザーについて考えてみます。そのユーザーがこのレプリカに書き込むことができる場合、DynamoDB はその書き込みを他のすべてのレプリカテーブルに伝播します。実際には、ユーザーは、グローバルテーブル内の他のすべてのレプリカに (間接的に) 書き込むことができます。このシナリオは、すべてのレプリカテーブルで一貫した IAM ポリシーを使用することで回避できます。

例: CreateGlobalTable アクションを許可します

グローバルテーブルにレプリカを追加するには、その前にグローバルテーブルとその各レプリカテーブルに対して `dynamodb:CreateGlobalTable` 許可を付与します。

以下の IAM ポリシーは、`CreateGlobalTable` アクションをすべてのテーブルに適用する許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
        }
    ]
}
```

例: UpdateGlobalTable、DescribeLimits、application-autoscaling:DeleteScalingPolicy、および application-autoscaling:DeregisterScalableTarget アクションを許可します

DynamoDB のグローバルテーブルの設定 (`UpdateGlobalTableSettings`) を更新するには、`dynamodb:UpdateGlobalTable`、`dynamodb:DescribeLimits`、`application-autoscaling:DeleteScalingPolicy` および `application-autoscaling:DeregisterScalableTarget` の許可が必要です。

以下の IAM ポリシーは、`UpdateGlobalTableSettings` アクションをすべてのテーブルに適用する許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

例: 特定のリージョンでのみレプリカが許可されている特定のグローバルテーブル名に対して CreateGlobalTable アクションを許可します

以下の IAM ポリシーは、`CreateGlobalTable` アクションが 2 つのリージョンにレプリカを持つ `Customers` という名前のグローバルテーブルを作成できるようにする許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:CreateGlobalTable",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Customers",
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers"
            ]
        }
    ]
}
```