aws

開発者ガイド

Amazon Quantum Ledger Database (Amazon QLDB)

Amazon Quantum Ledger Database (Amazon QLDB): 開発者ガイド

Table of Contents

Amazon QLDB とは

Amazon Quantum Ledger Database (Amazon QLDB) はフルマネージド型の台帳データベースで、信頼された中央機関が所有する、透過的でイミュータブルであり、暗号的に検証可能なトランザクションログを提供します。Amazon QLDB を使用して、すべてのアプリケーションデータの変更を追跡し、完全で検証可能な変更履歴を長期にわたって保持できます。Amazon Web Services で利用できるさまざまなデータベースオプションの詳細については、「AWS で組織に適したデータベースを選択する」を参照してください。

台帳は、通常、企業の経済活動や財務活動の履歴を記録するために使用されます。多くの組織では、アプリケーションのデータの正確な履歴を維持することを目的に、台帳に似た機能を提供するアプリケーションを構築しています。たとえば、銀行業務トランザクションにおける貸方と借方の履歴を追跡したり、保険金請求書のデータ系列を確認したり、サプライチェーンネットワーク内の商品の動きを追跡したりするなどの目的のためです。多くの場合、台帳アプリケーションは、リレーショナルデータベースに作成されたカスタムの監査テーブルや監査証跡を使用して実装されます。

Amazon QLDB は新しい種類のデータベースで、台帳のようなアプリケーションを自分で構築するという複雑な開発作業を行う必要がありません。QLDB では、データの変更履歴は変更不可能であり、上書きや変更はできません。また、暗号化を使用して、アプリケーションのデータに意図しない変更が行われていないことを確認できます。QLDB では、イミュータブルトランザクションログ (ジャーナル) を使用します。ジャーナルは追加専用であり、コミットされたデータを含む、一連のシーケンスおよびハッシュチェーンされたブロックで構成されます。

Amazon QLDB ビデオ

Amazon QLDB の概要とその利点については、YouTube でこの QLDB の概要ビデオをご覧ください。

Amazon QLDB の料金

Amazon QLDB では使用した分に対してのみ支払いが発生し、最低料金やサービス使用義務はありません。料金は台帳データベースで使用したリソースに対してのみ発生し、事前にプロビジョニングする必要はありません。

詳細については、「Amazon QLDB 料金」を参照してください。

QLDB の開始方法

最初に次のトピックを読むことをお勧めします。

- Amazon QLDB の概要 - QLDB の概要について説明します。
- Amazon QLDB の重要な概念と用語 - QLDB の基本的な概念と用語について説明します。
- Amazon QLDB へのアクセス - AWS Management Console、API、または AWS Command Line Interface (AWS CLI) を使用して QLDB にアクセスする方法について説明します。
- Amazon QLDB で IAM が機能する仕組み - AWS Identity and Access Management (IAM) を使用して、QLDB へのアクセスを制御する方法について説明します。

QLDB コンソールをすぐに使用開始するには、「Amazon QLDB コンソールの使用開始方法」を参照してください。

AWS が提供するドライバーを使用した QLDB での開発については、「Amazon QLDB ドライバーの開始方法」を参照してください。

Amazon QLDB の概要

以下のセクションでは、Amazon QLDB サービスコンポーネントと、コンポーネント間のやり取りの概要について説明します。

トピック

- ジャーナルファースト
- Immutable
- 暗号的に検証可能
- SQL のようにドキュメントを柔軟に操作可能
- オープンソースの開発者ツール
- サーバーレスでかつ高可用性
- エンタープライズグレード

ジャーナルファースト

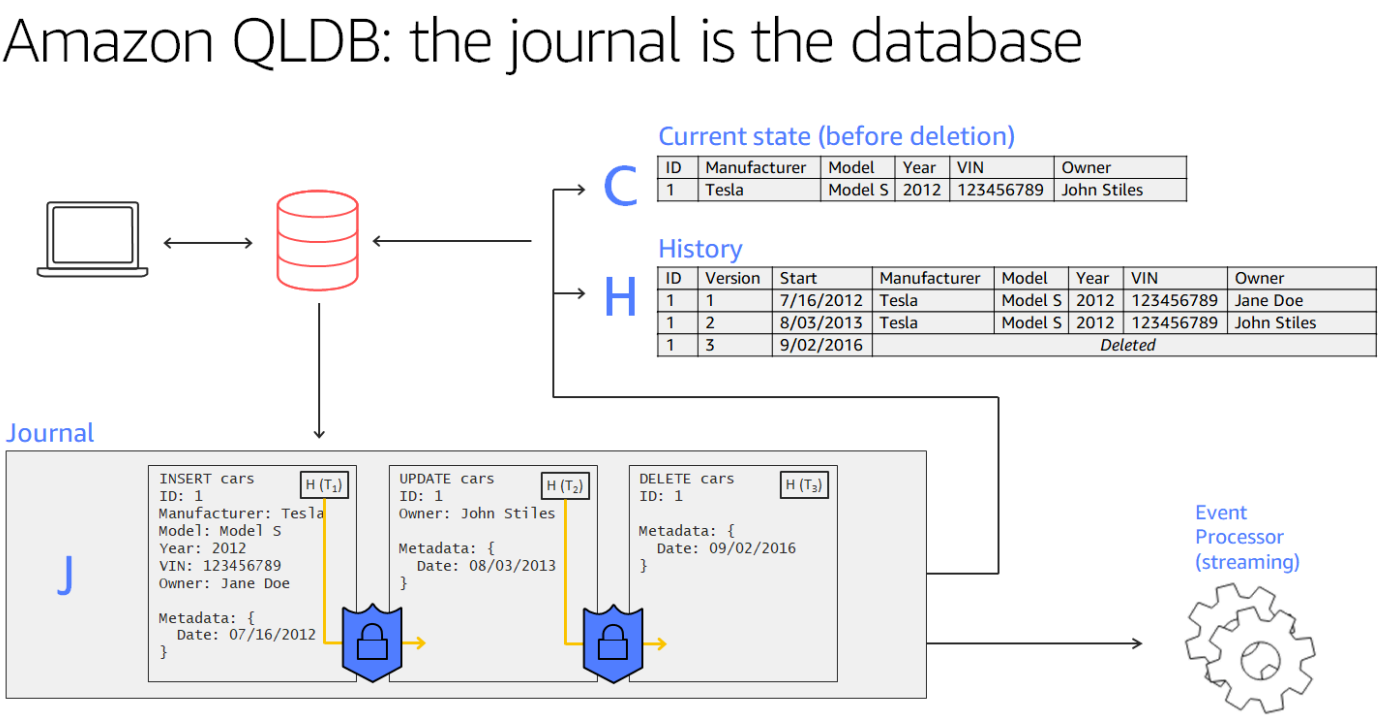
従来のデータベースアーキテクチャでは、通常、トランザクションの一部としてテーブルにデータを書き込みます。トランザクションログ (通常は内部実装) は、トランザクションと、それによるデー

タベースの変更をすべて記録します。トランザクションログは、データベースの重要なコンポーネントです。システム障害、災害復旧、またはデータレプリケーションが発生した場合に、トランザクションを再生するには、ログが必要です。ただし、データベーストランザクションログは変更不可能ではなく、ユーザーが直接簡単にアクセスできるようには設計されていません。

Amazon QLDB では、ジャーナルはデータベースの中核です。トランザクションログと似た構造のジャーナルは、アプリケーションデータと関連するメタデータを保存するイミュータブルな追加専用のデータ構造です。更新や削除を含むすべての書き込みトランザクションは、最初にジャーナルにコミットされます。

QLDB は、ジャーナルを使用して台帳データの現在の状態を確認する際に、データをクエリ可能なユーザー定義テーブルに具体化します。これらのテーブルには、ドキュメントのリビジョンやメタデータなど、すべてのトランザクションデータのアクセス可能な履歴も表示されます。ジャーナルはさらに、同時実行性、順序付け、暗号化検証、および台帳データの可用性を処理します。

QLDB ジャーナルのアーキテクチャを以下に図で示します。



- この例では、アプリケーションが台帳に接続し、cars という名前のテーブルに対しドキュメントを挿入、更新、削除するトランザクションを実行します。
- データは、まず順番にジャーナルに書き込まれます。

- その後、データは組み込みビューを使用してテーブルにマテリアライズされます。これらのビューでは、各リビジョンにバージョン番号が割り当てられた状態で、自動車の現在の状態と完全な履歴の両方にクエリを実行できます。
- ジャーナルから直接データをエクスポートまたはストリーミングすることもできます。

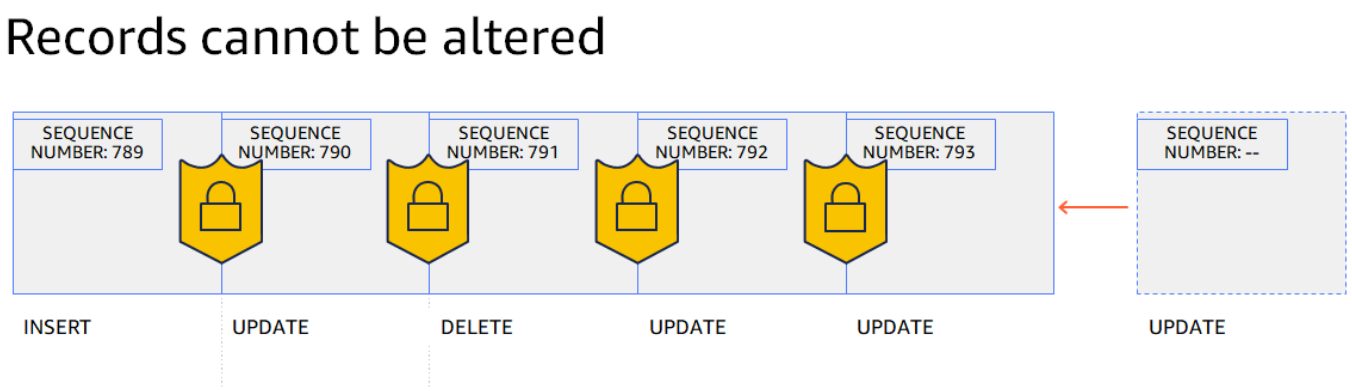
Immutable

QLDB ジャーナルは追加専用であるため、データに対するすべての変更の完全な記録が保持され、変更や上書きはできません。コミットされたデータを直接変更する API やその他の方法はありません。このジャーナルの構造により、台帳の全履歴にアクセスしてクエリを実行できます。

> **Note**
>
> QLDB がサポートする不変性の唯一の例外は、データの秘匿化です。この機能により、欧州連合の一般データ保護規則 (GDPR) やカリフォルニア州消費者プライバシー法 (CCPA) などの規制法令に準拠することができます。
> QLDB にはテーブルの履歴にある使用頻度の低いドキュメントリビジョンを完全に削除することができる秘匿化オペレーションがあります。このオペレーションでは、指定したリビジョンのユーザーデータのみが削除され、ジャーナルシーケンスとドキュメントのメタデータは変更されません。これにより、台帳の全体的なデータの整合性が維持されます。詳細については、「ドキュメントのリビジョンを秘匿化する」を参照してください。

QLDB は、トランザクション内のジャーナルに 1 つのブロックを書き込みます。各ブロックには、挿入、更新、削除するドキュメントを表すエントリオブジェクトと、それらをコミットするために実行したステートメントが含まれます。これらのブロックは、データの整合性を保証するために順序付けおよびハッシュチェーンされています。

以下の図は、このジャーナルの構造を示したものです。

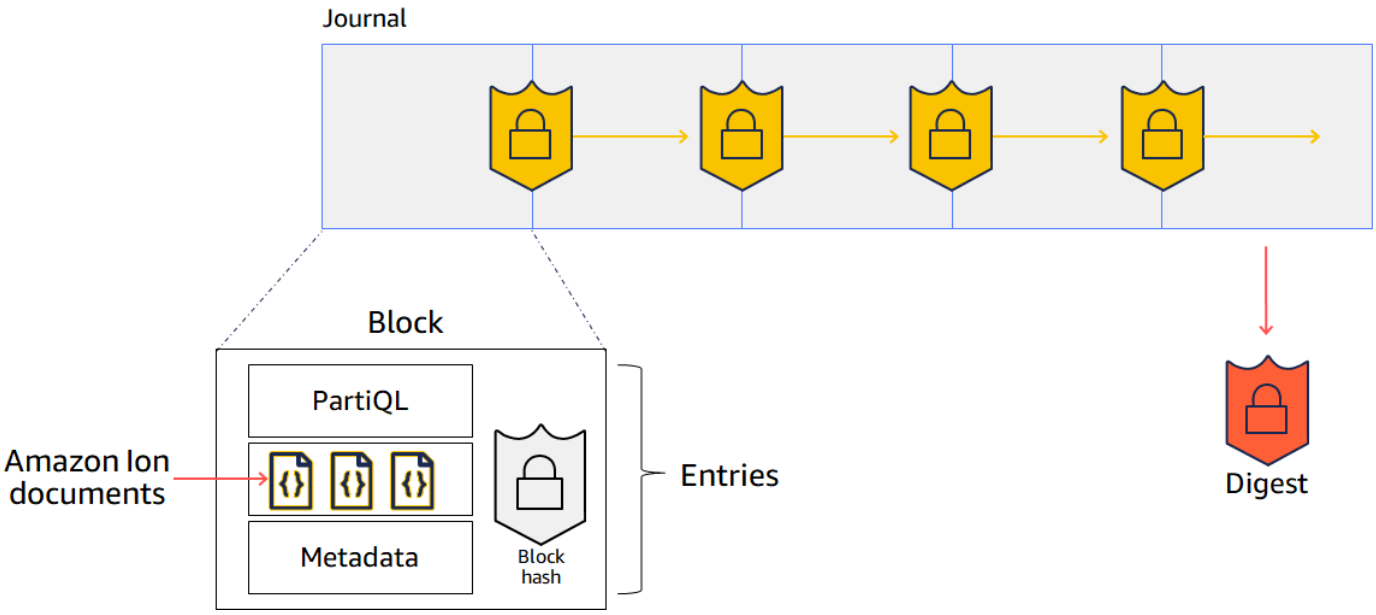


この図は、トランザクションが、検証のためにハッシュチェーンされたブロックとしてジャーナルにコミットされていることを示しています。各ブロックには、そのアドレスを指定するシーケンス番号があります。

暗号的に検証可能

ジャーナルブロックは、ブロックチェーンと同様に、暗号ハッシュ技術とともに順序付けされ、チェーンされます。QLDB はジャーナルのハッシュチェーンを使用し、暗号化検証方法を使用して、トランザクションデータの整合性を提供します。ダイジェスト (ある時点でのジャーナルの完全なハッシュチェーンを表すハッシュ値) と Merkle 監査証明 (バイナリハッシュツリー内のノードの妥当性を証明するメカニズム) を使用すると、データに対する意図しない変更がないことをいつでも確認できます。

次の図は、ある時点でのジャーナルの完全なハッシュチェーンをカバーするダイジェストを示しています。



この図では、ジャーナルブロックは SHA-256 暗号化ハッシュ関数を使用してハッシュ化され、後続のブロックに順次チェーンされています。各ブロックには、データドキュメント、メタデータ、およびトランザクションで実行された PartiQL ステートメントを含むエントリがあります。

詳細については、「Amazon QLDB でのデータ検証」を参照してください。

SQL のようにドキュメントを柔軟に操作可能

QLDB は PartiQL をクエリ言語として使用し、Amazon Ion をドキュメント指向のデータモデルとして使用します。PartiQL は、Ion で動作するように拡張されたオープンソースの SQL 互換のクエリ言語です。PartiQL を使用すると、使い慣れた SQL 演算子を使用してデータを挿入、クエリ、および管理できます。フラットドキュメントにクエリを実行する場合、構文は SQL を使用してリレーショナルテーブルをクエリするのと同じです。PartiQL の QLDB での実装については、「Amazon QLDB の PartiQL リファレンス」を参照してください。

Amazon Ion は JSON のスーパーセットです。Ion はオープンソースのドキュメントベースのデータ形式であり、構造化データ、半構造化データ、およびネストされたデータを柔軟に保存および処理できます。QLDB の詳細については、「Amazon QLDB の Amazon Ion データ形式リファレンス」を参照してください。

従来のリレーショナルデータベースと QLDB についての、コアコンポーネントと機能の概要の比較については、「リレーショナルから台帳へ」を参照してください。

オープンソースの開発者ツール

アプリケーション開発を簡素化するために、QLDB では、さまざまなプログラミング言語でオープンソースドライバーを提供しています。これらのドライバーを使用して、台帳で PartiQL ステートメントを実行し、それらのステートメントの結果を処理することで、トランザクションデータ API を操作できます。現在サポートされているドライバー言語に関する情報とチュートリアルについては、「Amazon QLDB ドライバーの開始方法」を参照してください。

Amazon Ion には、Ion データを処理するクライアントライブラリも用意されています。Ion データの処理に関するデベロッパーガイドおよびコード例については、GitHub の Amazon Ion ドキュメントを参照してください。

サーバーレスでかつ高可用性

QLDB は、フルマネージド、サーバーレスであり、高可用性も備えています。このサービスは、アプリケーションの需要に応じて自動的にスケールされるため、インスタンスや容量をプロビジョニングする必要はありません。データの複数のコピーが 1 つの AWS リージョン の 1 つのアベイラビリティーゾーン内および複数のアベイラビリティーゾーン間でレプリケートされます。

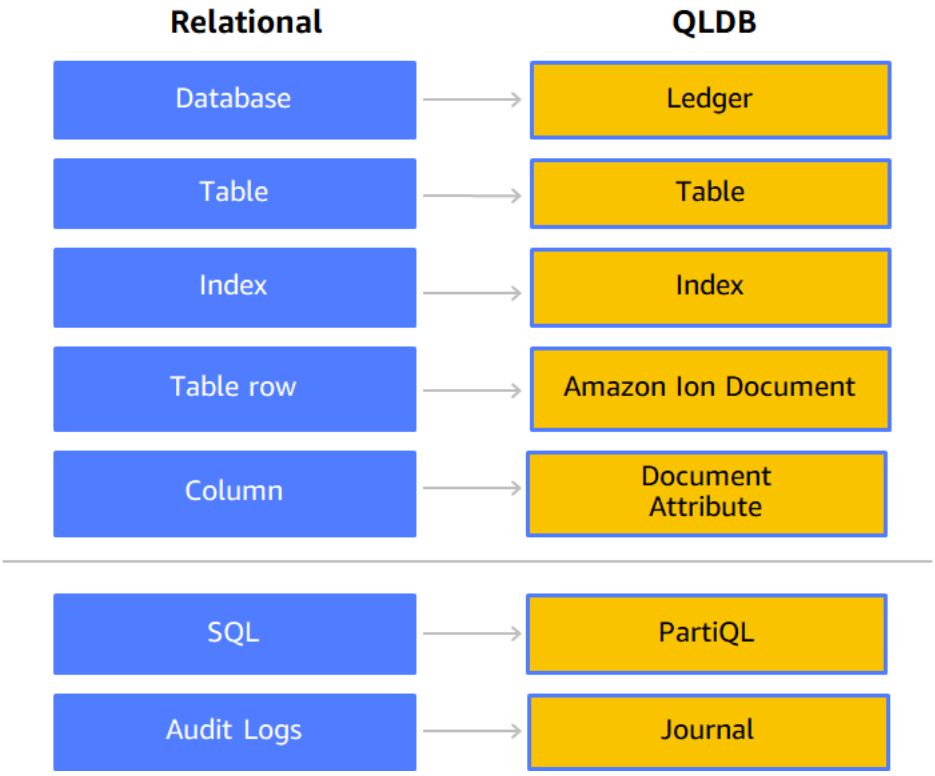
エンタープライズグレード

QLDB トランザクションはアトミック性、整合性、分離、耐久性 (ACID) の各特性に完全に準拠しています。QLDB はオプティミスティック同時実行制御 (OCC、Optimistic Concurrency Control) を使用し、トランザクションは完全な直列化可能性 (最高レベルの分離性) で動作します。つまり、ファントムリード、ダーティリード、書き込みスキューなどの並行性の問題が発生するリスクはありません。詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

リレーショナルから台帳へ

アプリケーションデベロッパーなら、リレーショナルデータベース管理システム (RDBMS) および構造化クエリ言語 (SQL) を使用した経験があるかもしれません。Amazon QLDB の使用を開始すると、多くの類似点があることがわかります。より高度なトピックに進むにつれて、RDBMS 基盤上に構築された QLDB の強力な新機能についても説明します。このセクションでは、一般的なデータベースのコンポーネントとオペレーションについて、QLDB での同等部分と比較、対比しながら説明します。

次の図は、従来の RDBMS と Amazon QLDB との間のコアコンポーネントのマッピング構造を示しています。



次の表に、従来の RDBMS と QLDB 間の組み込みオペレーション機能の主要な類似点と相違点の概要を示します。

操作	RDBMS	QLDB
テーブルの作成	すべての列名とデータ型を定義する CREATE TABLE ステートメント	スキーマレスおよびオープンコンテンツを許可するためにテーブル属性またはデータ型を定義しない CREATE TABLE ステートメント
インデックスの作成	CREATE INDEX ステートメント	CREATE INDEX ステートメント (テーブル上の任意の最上位フィールドが対象)
データの挿入	テーブルで定義されているスキーマに準拠する新しい行またはタプル内の値を指定する INSERT ステートメント	テーブル内の既存のドキュメントに関係なく、有効な Amazon Ion 形式で新しいド

操作	RDBMS	QLDB
		キュメント内の値を指定する INSERT ステートメント
データのクエリ	SELECT-FROM-WHERE ステートメント	フラットドキュメントにクエリを実行する場合は SQL と同じ構文で SELECT-FROM-WHERE ステートメント
データの更新	UPDATE-SET-WHERE ステートメント	フラットドキュメントを更新する場合は SQL と同じ構文で UPDATE-SET-WHERE ステートメント
データの削除	DELETE-FROM-WHERE ステートメント	フラットドキュメントを削除する場合は SQL と同じ構文で DELETE-FROM-WHERE ステートメント
ネストされたデータおよび半構造化されたデータ	フラット行またはタプルのみ	Amazon Ion データ形式と PartiQL クエリ言語でサポートされている構造化、半構造化、またはネストされたデータを持つことができるドキュメント
メタデータのクエリ	組み込みメタデータなし	テーブルのコミットされた組み込みビューからクエリを実行する SELECT ステートメント
リビジョン履歴のクエリの実行	組み込みデータ履歴なし	組み込みの履歴関数からクエリを実行する SELECT ステートメント

操作	RDBMS	QLDB
暗号検証	組み込みの暗号化または不変性なし	ジャーナルのダイジェスト、およびそのダイジェストに関連するドキュメントリビジョンの整合性を検証する証明を返す API

QLDB の主要な概念と用語の概要については、「重要な概念」を参照してください。

台帳でのデータの作成、クエリ、管理のプロセスの詳細については、「データと履歴の使用」を参照してください。

Amazon QLDB の重要な概念と用語

このセクションでは、台帳の構造や台帳によるデータの管理方法など、Amazon QLDB の重要な概念と用語の概要について説明します。台帳データベースの QLDB は、主要なコンセプトに関して、他のドキュメント指向のデータベースとは異なります。

トピック

QLDB データオブジェクトモデル

Amazon QLDB の基本データオブジェクトモデルは次のとおりです。

1. 台帳

   最初のステップは、台帳を作成することです。これは QLDB のプライマリ AWS リソースタイプです。台帳の作成方法については、「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」、または「Amazon QLDB 台帳の基本的なオペレーション」を参照してください。

台帳の ALLOW_ALL と STANDARD のどちらのアクセス許可モードでも、この台帳リソースで API オペレーションを実行するアクセス許可を付与する AWS Identity and Access Management (IAM) ポリシーを作成します。

台帳 ARN 形式:

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}
```

2. ジャーナルとテーブル

   QLDB 台帳でデータの書き込みを開始するには、まずテーブルを 基本 CREATE TABLE ステートメントで作成します。台帳データは、台帳のジャーナルにコミットされたドキュメントのリビジョンで構成されます。ユーザー定義テーブルのコンテキストで、ドキュメントリビジョンを台帳にコミットします。QLDB のテーブルは、ジャーナルからのドキュメントリビジョンのコレクションのマテリアライズドビューです。

   STANDARD アクセス許可モードの台帳では、このテーブルリソースで PartiQL ステートメントを実行するために、アクセス許可を付与する IAM ポリシーを作成する必要があります。テーブルリソースに対するアクセス許可を使用すると、テーブルの現在の状態にアクセスするステートメントを実行できます。組み込みの history() 関数を使用してテーブルのリビジョン履歴のクエリを実行することもできます。

   テーブル ARN 形式:

   ```
   arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/table/${table-id}
   ```

   台帳およびその関連リソースに対するアクセス許可の付与の詳細については、「Amazon QLDB で IAM が機能する仕組み」を参照してください。

3. ドキュメント

   テーブルは QLDB ドキュメント のリビジョンで構成されます。これらは、Amazon Ion struct 形式のデータセットです。ドキュメントリビジョンは、一意のドキュメント ID によって識別される一連のドキュメントの 1 つのバージョンを表します。

   QLDB は、コミットしたドキュメントの完全な変更履歴を保存します。テーブルではドキュメントの現在の状態のクエリを実行できますが、history() 関数を使用するとテーブルのドキュメントのリビジョン履歴全体についてクエリを実行できます。リビジョンのクエリと書き込みの詳細については、「データと履歴の使用」を参照してください。

4. システムカタログ

   各台帳では、システム定義のカタログリソースも提供され、このリソースでクエリを実行して台帳内のすべてのテーブルとインデックスを一覧表示できます。STANDARD アクセス許可モードの台帳では、次の操作を実行するために、このカタログリソースに対する `qldb:PartiQLSelect` アクセス許可が必要です。

   - SELECT ステートメントをシステムカタログテーブル schema.user_tables で実行します。
   - テーブルおよびインデックス情報を QLDB コンソールの台帳詳細ページに表示します。
   - テーブルおよびインデックスのリストを QLDB コンソールの PartiQL エディタに表示します。

   カタログ ARN 形式:

   ```
   arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/information_schema/
   user_tables
   ```

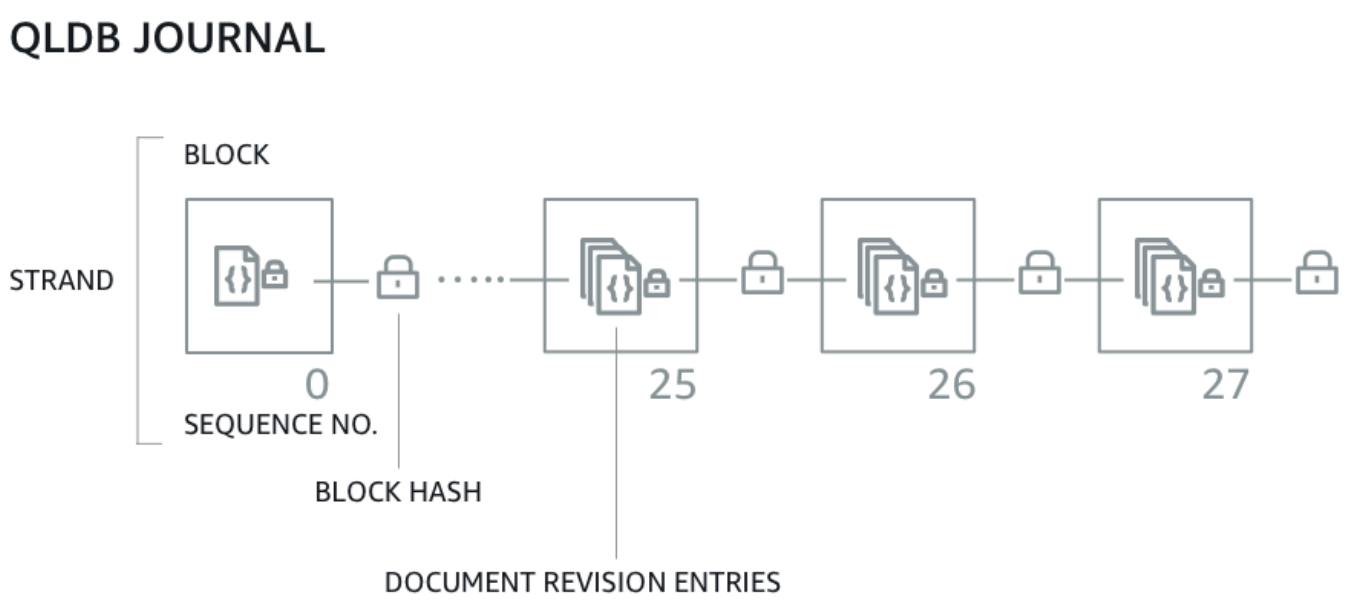
ジャーナルファーストトランザクション

アプリケーションが QLDB 台帳のデータを読み書きするときは、データベーストランザクション内で行われます。すべてのトランザクションは、Amazon QLDB でのクォータと制限 で定義されている制限の対象となります。トランザクション内で、QLDB は次のステップを実行します。

1. 台帳からデータの現在の状態を読み取ります。
2. トランザクションで提供されるステートメントを実行して、完全な直列化可能分離を実現するために、オプティミスティック同時実行制御 (OCC) を使用して競合がないかチェックします。
3. OCC 競合が見つからない場合は、次のようにトランザクション結果を返します。
   - 読み込みの場合は、結果セットを返し、追加専用の方法でジャーナルに SELECT ステートメントをコミットします。
   - 書き込みの場合は、更新、削除、または新しく挿入されたデータを追加専用の方法でジャーナルにコミットします。

ジャーナルは、データに行ったすべてのイミュータブルな変更履歴全体を意味します。QLDB は、トランザクション内のジャーナルに 1 つの連鎖ブロックを書き込みます。各ブロックには、挿入、アップデート、削除するドキュメントリビジョンを表すエントリオブジェクトと、それらをコミットした PartiQL ステートメントが含まれます。

以下の図は、このジャーナルの構造を示したものです。



この図は、トランザクションがドキュメントのリビジョンのエントリを含むブロックとしてジャーナルにコミットされていることを示しています。各ブロックはハッシュされ、検証のために後続のブロックに連鎖されます。各ブロックには、ストランド内のアドレスを指定するシーケンス番号があります。

> **Note**
>
> Amazon QLDB で、ストランドは台帳のジャーナルの仕切りです。QLDB は現在、単一ストランドのジャーナルのみをサポートしています。

ブロック内のデータコンテンツについては、「Amazon QLDB のジャーナルコンテンツ」を参照してください。

データのクエリの実行

QLDB は、高性能なオンライントランザクション処理 (OLTP) ワークロードのニーズに対応することを目的としています。台帳は、ジャーナルにコミットされたトランザクション情報に基づいて、データのクエリ可能なテーブルビューを提供します。QLDB のテーブルビューは、テーブル内のデータのサブセットです。ビューにはリアルタイムのものが反映されますので、常にクエリを実行するアプリケーションで使用できます。

PartiQL SELECT ステートメントを使用して、以下のシステム定義ビューにクエリを実行できます。

- ユーザー - テーブルに書き込んだデータのみの最新のアクティブなリビジョン (つまり、ユーザーデータの現在の状態)。これが QLDB のデフォルトビューです。
- コミット済み - ユーザーデータと、システムによって生成されたメタデータの両方の最新のアクティブなリビジョン。これは完全システム定義テーブルで、ユーザーテーブルに直接対応しています。

これらのクエリ可能なビューに加えて、組み込みの 履歴関数 を使用してデータのリビジョン履歴を照会できます。履歴関数は、コミット済みビューと同じスキーマにあるユーザーデータおよびその関連メタデータの両方を返します。

データストレージ

QLDB には、次の 2 種類のデータストレージがあります。

- ジャーナルストレージ - 台帳のジャーナルで使用されるディスク容量。ジャーナルには、データへのすべての変更に関する完全、不変、検証可能な履歴が含まれています。
- インデックス付きストレージ - 台帳のテーブル、インデックス、およびインデックス付き履歴で使用されるディスク容量。インデックス付きストレージは、高パフォーマンスなクエリ用に最適化された台帳データで構成されています。

データがジャーナルにコミットされると、定義したテーブルにマテリアライズされます。これらのテーブルは、より高速で効率的なクエリを可能にするため最適化されています。アプリケーションがトランザクションデータ API を使用してデータを読み取ると、インデックス付きストレージに格納されているテーブルとインデックスにアクセスします。

QLDB API モデル

QLDB は、アプリケーションコードで操作できる 2 種類の API を提供します。

- Amazon QLDB – QLDB リソース管理 API (コントロールプレーンとも呼ばれます)。この API は、台帳リソースの管理および非トランザクションデータオペレーションにのみ使用されます。これらのオペレーションを使用して、台帳を作成、削除、説明表示、一覧表示、更新できます。また、データを暗号的に検証し、ジャーナルブロックをエクスポートまたはストリーミングすることもできます。
- Amazon QLDB セッション - QLDB トランザクションデータプレーン。この API を PartiQL ステートメントで使用して、台帳のデータトランザクションを実行できます。

> **Important**
>
> QLDB セッション API と直接やり取りする代わりに、QLDB ドライバーまたは QLDB シェルを使用して、台帳のデータトランザクションを実行することをお勧めします。
>
> - AWS SDK を使用している場合は、QLDB ドライバーを使用します。このドライバーは、QLDB セッションデータ API 上に高レベルの抽象化レイヤーを提供し、`SendCommand` オペレーションを管理します。サポートされているプログラミング言語の情報とリストについては、「ドライバーの開始方法」を参照してください。
> - AWS CLI で作業している場合は、QLDB シェルを使用します。シェルは、QLDB ドライバーを使用して台帳と対話するコマンドラインインターフェイスです。詳細については、Amazon QLDB シェルの使用 (データ API のみ)を参照してください。

使用できる API オペレーションの詳細については、「Amazon QLDB API リファレンス」を参照してください。

次のステップ

自分のデータで台帳を使用する方法については、「Amazon QLDB でのデータと履歴の使用」を参照し、テーブルの作成、データの挿入、および基本的なクエリの実行の手順を説明する例に従ってください。このガイドでは、コンテキストのサンプルデータとクエリの例を使用して、これらの概念がどのように機能するかについて詳細を説明します。

QLDB コンソールを使用してサンプルアプリケーションチュートリアルをすばやく使用するには、「Amazon QLDB コンソールの使用開始方法」を参照してください。

このセクションで説明する主要な用語と定義のリストについては、「Amazon QLDB 用語集」を参照してください。

Amazon QLDB のジャーナルコンテンツ

Amazon QLDB の場合、ジャーナルは、データに対するすべての変更の完全かつ検証可能な履歴を保存するイミュータブルトランザクションログです。ジャーナルは追加専用であり、コミットされたデータやその他のシステムメタデータを含む、一連のシーケンスおよびハッシュチェーンされたブロックで構成されます。QLDB は、トランザクション内のジャーナルに 1 つの連鎖ブロックを書き込みます。

このセクションでは、サンプルデータを含むジャーナルブロックの例を示し、ブロックの内容について説明します。

トピック

ブロックの例

ジャーナルブロックには、トランザクションのメタデータ、トランザクションでコミットされたドキュメントリビジョンを表すエントリ、およびそれらをコミットした PartiQL ステートメントが含まれます。

以下に、サンプルデータを含むブロックの例を示します。

> **Note**
>
> このブロック例は、情報提供のみを目的として記載されています。示されているハッシュは、実際に計算されたハッシュ値ではありません。

```
{
  blockAddress:{
    strandId:"4o5UuzWSW5PIoOGm5jPA6J",
    sequenceNo:25
  },
  transactionId:"3gtB8Q8dfIMA8lQ5pzHAMo",
  blockTimestamp:2022-06-08T18:46:46.512Z,
  blockHash:{{QS5lJt8vRxT30L9OGL5oU1pxFTe+UlEwakYBCrvGQ4A=}},
  entriesHash:{{buYYc5kV4rrRtJAsrIQnfnhgkzfQ8BKjI0C2vFnYQEw=}},
  previousBlockHash:{{I1UKRIWUgkM1X6042kcoZ/eN1rn0uxhDTc08zw9kZ5I=}},
  entriesHashList:[
    {{BUCXP6oYgmug2AfPZcAZup2lKolJNTbTuV5RA1VaFpo=}},
    {{cTIRkjuULzp/4KaUEsb/S7+TG8FvpFiZHT4tEJGcANc=}},
    {{3aktJSMyJ3C5StZv4WIJLu/w3D8mGtduZvP0ldKUaUM=}},
    {{GPKIJ1+o8mMZmPj/35ZQXoca2z64MVYMCwqs/g080IM=}}
```

```
  ],
  transactionInfo:{
    statements:[
      {
        statement:"INSERT INTO VehicleRegistration VALUE ?",
        startTime:2022-06-08T18:46:46.063Z,
        statementDigest:{{KY2nL6UGUPs5lXCLVXcUaBxcEIop0Jvk4MEjcFVBfwI=}}
      },
      {
        statement:"SELECT p_id FROM Person p BY p_id WHERE p.FirstName = ? and
p.LastName = ?",
        startTime:2022-06-08T18:46:46.173Z,
        statementDigest:{{QS2nfB8XBf2ozlDx0nvtsliOYDSmNHMYC3IRH4Uh690=}}
      },
      {
        statement:"UPDATE VehicleRegistration r SET r.Owners.PrimaryOwner.PersonId = ?
WHERE r.VIN = ?",
        startTime:2022-06-08T18:46:46.278Z,
        statementDigest:{{nGtIA9Qh0/dwIplOR8J5CTeqyUVtNUQgXfltDUo2Aq4=}}
      },
      {
        statement:"DELETE FROM DriversLicense l WHERE l.LicenseNumber = ?",
        startTime:2022-06-08T18:46:46.385Z,
        statementDigest:{{ka783dcEP58Q9AVQ1m9NOJd3JAmEvXLjzl0OjN1BojQ=}}
      }
    ],
    documents:{
      HwVFkn8IMRa0xjze5xcgga:{
        tableName:"VehicleRegistration",
        tableId:"HQZ6cgIMUi204Lq1tT4oaJ",
        statements:[0,2]
      },
      IiPTRxLGJZa342zHFCFT15:{
        tableName:"DriversLicense",
        tableId:"BvtXEB1JxZg0lJlBAtbtSV",
        statements:[3]
      }
    }
  },
  revisions:[
    {
      hash:{{FR1IWcWew0yw1TnRklo2YMF/qtwb7ohsu5FD8A4DSVg=}}
    },
    {
```

```
      blockAddress:{
        strandId:"4o5UuzWSW5PIoOGm5jPA6J",
        sequenceNo:25
      },
      hash:{{6TTHbcfIVdWoFC/j90BOZi0JdHzhjSXo1tW+uHd6Dj4=}},
      data:{
        VIN:"1N4AL11D75C109151",
        LicensePlateNumber:"LEWISR261LL",
        State:"WA",
        City:"Seattle",
        PendingPenaltyTicketAmount:90.25,
        ValidFromDate:2017-08-21,
        ValidToDate:2020-05-11,
        Owners:{
          PrimaryOwner:{
            PersonId:"3Ax20JIix5J2ulu2rCMvo2"
          },
          SecondaryOwners:[]
        }
      },
      metadata:{
        id:"HwVFkn8IMRa0xjze5xcgga",
        version:0,
        txTime:2022-06-08T18:46:46.492Z,
        txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
      }
    },
    {
      blockAddress:{
        strandId:"4o5UuzWSW5PIoOGm5jPA6J",
        sequenceNo:25
      },
      hash:{{ZVF/f1uSqd5DIMqzI04CCHaCGFK/J0Jf5AFzSEk0l90=}},
      metadata:{
        id:"IiPTRxLGJZa342zHFCFT15",
        version:1,
        txTime:2022-06-08T18:46:46.492Z,
        txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
      }
    }
  ]
}
```

revisions フィールドには、hash 値のみを含み、その他の属性を含まないリビジョンオブジェクトもあります。これらは、ユーザーデータを含まない内部のみのシステムリビジョンです。これらのリビジョンのハッシュは、ジャーナルの完全なハッシュチェーンの一部であり、暗号検証に必要です。

ブロックの内容

ジャーナルブロックには、次のフィールドがあります。

blockAddress

ジャーナル内のブロックの位置。アドレスは、strandId と sequenceNo という 2 つのフィールドを含む Amazon Ion 構造です。

例: {strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}

transactionId

ブロックをコミットしたトランザクションの一意の ID。

blockTimestamp

ブロックがジャーナルにコミットされた時刻を示すタイムスタンプ。

blockHash

ブロックを一意に表す 256 ビットのハッシュ値。これは、entriesHash と previousBlockHash の連結のハッシュです。

entriesHash

ブロック内のすべてのエントリ (内部専用のシステムエントリを含む) を表すハッシュ。これは、Merkle ツリーのルートハッシュです。このツリーのリーフノードは、entriesHashList のすべてのハッシュで構成されます。

previousBlockHash

ジャーナル内の 1 つ前のチェーンされたブロックのハッシュ。

entriesHashList

ブロック内の各エントリを表すハッシュのリスト。このリストには、次のエントリハッシュが含まれる場合があります。

- transactionInfo を表す Ion ハッシュ。この値は、transactionInfo 構造全体の Ion ハッシュを取得することによって計算されます。

- リーフノードが `revisions` のすべてのハッシュで構成される Merkle ツリーのルートハッシュ。
- `redactionInfo` を表す Ion ハッシュ。このハッシュは、秘匿化のトランザクションによってコミットされたブロックにのみ存在します。この値は、`redactionInfo` 構造全体の Ion ハッシュを取得することによって計算されます。
- 内部専用のシステムメタデータを表すハッシュ。これらのハッシュは、すべてのブロックに存在するとは限りません。

`transactionInfo`

ブロックをコミットしたトランザクション内のステートメントに関する情報を含む Amazon Ion 構造。この構造には次のフィールドがあります。

- `statements` - PartiQL ステートメントのリストと、ステートメントの実行が開始したときの `startTime`。各ステートメントには、`transactionInfo` 構造のハッシュを計算するために必要な `statementDigest` ハッシュがあります。
- `documents` - ステートメントによって更新されたドキュメント ID。各ドキュメントには、そのドキュメントが所属する先の `tableName` と `tableId`、およびそのドキュメントを更新した各ステートメントのインデックスが含まれます。

`revisions`

ブロック内でコミットされたドキュメントリビジョンのリスト。各リビジョン構造には、リビジョンのコミットされたビューのすべてのフィールドが含まれます。

これには、ジャーナルのハッシュチェーン全体の一部を構成する内部専用のシステムリビジョンを表すハッシュも含まれます。

秘匿化済みのリビジョン

Amazon QLDB では、`DELETE` ステートメントは、削除済みとしてマークする新しいリビジョンを作成することによってのみドキュメントを論理的に削除します。QLDB テーブルの履歴にある使用頻度の低いドキュメントリビジョンを完全に削除できるデータの秘匿化オペレーションもサポートしています。

秘匿化オペレーションでは、指定されたリビジョンのユーザーデータのみが削除され、ジャーナルシーケンスとドキュメントのメタデータは変更されません。これにより、台帳の全体的なデータの整合性が維持されます。詳細と秘匿化オペレーションの例については、「ドキュメントのリビジョンを秘匿化する」を参照してください。

秘匿化済みのリビジョンの例

先ほどのブロックの例を考えてみましょう。このブロックでは、秘匿化中のリビジョンのドキュメント ID が HwVFkn8IMRa0xjze5xcgga で、バージョン番号が 0 とします。

秘匿化が完了すると、リビジョン内の (data 構造で表される) dataHash ユーザーデータは新しいフィールドに置き換えられます。このフィールドの値は、削除された data 構造の Ion ハッシュです。その結果、台帳は全体的なデータ整合性を維持し、既存の検証 API オペレーションを通じて暗号的に検証できる状態を維持します。

次の改訂例は、この秘匿化の結果を示しています。新しい dataHash フィールドは#####で強調表示されています。

> **Note**
>
> このリビジョンの例は、情報提供のみを目的として記載されています。示されているハッシュは、実際に計算されたハッシュ値ではありません。

```
...
{
  blockAddress:{
    strandId:"4o5UuzWSW5PIoOGm5jPA6J",
    sequenceNo:25
  },
  hash:{{6TTHbcfIVdWoFC/j90BOZi0JdHzhjSXo1tW+uHd6Dj4=}},
  dataHash:{{s83jd7sfhsdfhksj7hskjdfjfpIPP/DP2hvionas2d4=}},
  metadata:{
    id:"HwVFkn8IMRa0xjze5xcgga",
    version:0,
    txTime:2022-06-08T18:46:46.492Z,
    txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
  }
}
...
```

また、QLDB は、完了した秘匿化のリクエストに対応する新しいブロックをジャーナルに追加します。このブロックには、次の例に示すように、トランザクション内で秘匿化済みのリビジョンのリストを含む追加 redactionInfo エントリが含まれます。

```
...
```

```
redactionInfo:{
  revisions:[
    {
      blockAddress:{
        strandId:"4o5UuzWSW5PIoOGm5jPA6J",
        sequenceNo:25
      },
      tableId:"HQZ6cgIMUi204Lq1tT4oaJ",
      documentId:"HwVFkn8IMRa0xjze5xcgga",
      version:0
    }
  ]
}
...
```

サンプルアプリケーション

エクスポートされたデータを使用してジャーナルのハッシュチェーンを検証する Java コード例については、GitHub リポジトリ aws-samples/amazon-qldb-dmv-sample-java を参照してください。このサンプルアプリケーションには、次のクラスファイルが含まれています。

- ValidateQldbHashChain.java - 台帳からジャーナルブロックをエクスポートし、エクスポートされたデータを使用してブロック間のハッシュチェーンを検証するチュートリアルコードが含まれています。
- JournalBlock.java - ブロック内の個々のハッシュコンポーネントを計算する方法を示す `verifyBlockHash()` という名前のメソッドが含まれています。このメソッドは、`ValidateQldbHashChain.java` のチュートリアルコードによって呼び出されます。

この完全なサンプルアプリケーションをダウンロードしてインストールする方法については、「Amazon QLDB Java サンプルアプリケーションのインストール」を参照してください。チュートリアルコードを実行する前に、「Java チュートリアル」のステップ 1 ~ 3 を実行し、サンプル台帳をセットアップして、サンプルデータを使用してロードしてください。

以下も参照してください。

QLDB におけるジャーナルの詳細については、以下のトピックを参照してください。

- 「Amazon QLDB からのジャーナルデータのエクスポート」- ジャーナルデータを Amazon Simple Storage Service (Amazon S3) にエクスポートする方法について説明します。

- 「Amazon QLDB からのジャーナルデータのストリーミング」- ジャーナルデータを Amazon Kinesis Data Streams にストリーミングする方法について説明します。
- 「Amazon QLDB でのデータ検証」- ジャーナルデータの暗号検証について説明します。

Amazon QLDB 用語集

以下は、Amazon QLDB を使用する際の重要な用語の定義です。

ブロック | ダイジェスト | document | ドキュメント ID | ドキュメントリビジョン | エントリ | field | インデックス | インデックス付きストレージ | ジャーナル | ジャーナルブロック | ジャーナルストレージ | ジャーナルストランド | ジャーナルチップ | 台帳 | 証明 | revision | セッション | ストランド | テーブル | テーブルビュー | 表示

ブロック

トランザクション中にジャーナルにコミットされたオブジェクト。1 件のトランザクションは 1 つのブロックに書き込むことができます。そのため、1 つのブロックに関連付けられるトランザクションは 1 件のみです。ブロックには、トランザクションでコミットされたドキュメントリビジョンを表すエントリ、それらをコミットした PartiQL ステートメントが含まれます。

各ブロックに検証用のハッシュ値も含まれています。ブロックハッシュは、そのブロック内のエントリハッシュと、1 つ前のチェーン化されたブロックのハッシュを組み合わせて計算されます。

ダイジェスト

ある時点における台帳のドキュメントリビジョンの履歴全体を一意に表す 256 ビットのハッシュ値。ダイジェストハッシュは、ジャーナル内の最新ブロックがコミットされた時点における、ジャーナルのハッシュチェーン全体から計算されます。

QLDB を使用すると、セキュアな出力ファイルとしてダイジェストを生成できます。次に、その出力ファイルを使用して、そのハッシュに関連するドキュメントリビジョンの整合性を検証できます。

document

テーブルで挿入、更新、削除できる Amazon Ion `struct` 形式のデータセット。QLDB ドキュメントには、構造化データ、半構造化データ、ネストされたデータ、スキーマレスデータを含めることができます。

ドキュメント ID

テーブルに挿入したドキュメントごとに QLDB から割り当てられる共通の一意の識別子 (UUID、Universally Unique Identifier)。この ID は 128 ビットの数値であり、Base62 エンコードの英数字文字列 (22 文字の固定長) で表されます。

ドキュメントリビジョン

一意のドキュメント ID によって識別される一連のドキュメントの 1 つのバージョンを表す Ion 構造体。リビジョンには、ユーザーデータ (ユーザーがテーブルに書き込んだデータ) とシステム生成されたメタデータの両方が含まれます。各リビジョンはテーブルに関連付けられ、ドキュメント ID と 0 から始まるバージョン番号を組み合わせて一意に識別されます。

エントリ

ブロックに含まれるオブジェクト。エントリとは、トランザクションで挿入、更新、削除されたドキュメントリビジョンや、それらをコミットした PartiQL ステートメントを表します。

各エントリに検証用のハッシュ値も含まれます。エントリハッシュは、そのエントリ内のリビジョンハッシュまたはステートメントハッシュから計算されます。

field

QLDB ドキュメントの各属性を構成する名前と値のペア。名前はシンボルトークンであり、値は制限されません。

インデックス

データ取得オペレーションのパフォーマンスを最適化するためにテーブルに作成できるデータ構造。QLDB のインデックスについては、「Amazon QLDB の PartiQL リファレンス」の「CREATE INDEX」を参照してください。

インデックス付きストレージ

台帳のテーブル、インデックス、およびインデックス付き履歴で使用されるディスク容量。インデックス付きストレージは、高パフォーマンスなクエリ用に最適化された台帳データで構成されています。

ジャーナル

台帳でコミットされたすべてのブロックのハッシュチェーンのセット。ジャーナルは、追加専用で、台帳データに行ったすべての変更の完全で不変な履歴全体を表します。

ジャーナルブロック

「ブロック」を参照してください。

ジャーナルストレージ

台帳のジャーナルで使用されるディスク容量。

ジャーナルストランド

「ストランド」を参照してください。

ジャーナルチップ

特定時点におけるジャーナル内のコミットされた最新ブロック。

台帳

Amazon QLDB 台帳データベースリソースのインスタンス。これが、QLDB のプライマリ AWS リソースタイプです。台帳は、ジャーナルストレージとインデックス付きストレージの両方で構成されます。台帳データがジャーナルにコミットされると、Amazon Ion ドキュメントリビジョンのテーブルでクエリできるようになります。

証明

指定のダイジェストおよびドキュメントリビジョンに対して QLDB から返される 256 ビットのハッシュ値の順序付きリスト。これは、指定されたリビジョンハッシュをダイジェストハッシュに連鎖させるためにマークルツリーモデルによって必要とされるハッシュで構成されています。証明を使用することで、ダイジェストに関連するリビジョンの整合性を検証できます。詳細については、「Amazon QLDB でのデータ検証」を参照してください。

revision

「ドキュメントリビジョン」を参照してください。

セッション

台帳とのデータトランザクションのリクエストとレスポンスに関する情報を管理するオブジェクト。アクティブセッション (アクティブにトランザクションを実行するセッション) は、台帳への単一の接続を表します。QLDB ではセッションごとに 1 つのトランザクションがアクティブに実行されます。

ストランド

ジャーナルの仕切り。QLDB は現在、単一ストランドのジャーナルのみをサポートしています。

テーブル

台帳のジャーナルにコミットされたドキュメントリビジョンの順序付けられていないコレクションのマテリアライズドビュー。

テーブルビュー

ジャーナルにコミットされたトランザクションに基づく、テーブルのデータのクエリ可能なサブセット。PartiQL ステートメント内のビューは、テーブル名のプレフィックス修飾子 (_ql_ で始まるもの) で表されます。

SELECT ステートメントを使用して、以下のシステム定義ビューにクエリを実行できます。

- ユーザー - テーブルに書き込んだデータのみの最新のアクティブなリビジョン (つまり、ユーザーデータの現在の状態)。これが QLDB のデフォルトビューです。
- コミット済み - ユーザーデータと、システムによって生成されたメタデータの両方の最新のアクティブなリビジョン。これは完全システム定義テーブルで、ユーザーテーブルに直接対応しています。例: `_ql_committed_`*`TableName`*。

表示

「テーブルビュー」を参照してください。

Amazon QLDB へのアクセス

Amazon QLDB には、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または QLDB API を使用してアクセスできます。次のセクションでは、これらのオプションの使用方法と使用の前提条件について説明します。

前提条件

QLDB にアクセスする前に、まだ設定 AWS アカウント していない場合は、 を設定する必要があります。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- IAM での QLDB 権限の管理
- プログラムによるアクセスを付与する (オプション)

にサインアップする AWS アカウント

がない場合は AWS アカウント、次のステップを実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール) 」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。
2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインインユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

IAM での QLDB 権限の管理

AWS Identity and Access Management (IAM) を使用してユーザーの QLDB アクセス許可を管理する方法については、「」を参照してくださいAmazon QLDB で IAM が機能する仕組み。

プログラムによるアクセスを付与する (オプション)

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の

プログラマチックアクセス権を必要とするユーザー	目的	方法
		AWS CLI「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	(非推奨) 長期認証情報を使用して、、AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

Amazon QLDB にアクセスする方法

を設定するための前提条件を満たしたら AWS アカウント、QLDB へのアクセス方法の詳細については、以下のトピックを参照してください。

- コンソールを使用する場合
- の使用 AWS CLI （ 管理 API のみ ）
- Amazon QLDB シェルの使用 (データ API のみ)
- API を使用する場合

コンソールを使用した Amazon QLDB へのアクセス

Amazon QLDB AWS Management Console の には、https://console.aws.amazon.com/qldb からアクセスできます。

コンソールを使用して、QLDB で以下のことを行うことができます。

- 台帳を作成、削除および列挙したり、台帳に説明を追加したりする。
- PartiQL エディタを使用して PartiQL ステートメントを実行します。
- QLDB リソースのタグを管理する。
- ジャーナルデータを暗号的に検証する。
- ジャーナルブロックをエクスポートまたはストリーミングします。

Amazon QLDB 台帳を作成し、サンプルアプリケーションデータで設定する方法については、「Amazon QLDB コンソールの使用開始方法」を参照してください。

PartiQL エディタのクイックリファレンス

Amazon QLDB では、クエリ言語として PartiQL のサブセットがサポートされ、ドキュメント指向のデータ形式として Amazon Ion がサポートされています。PartiQL の QLDB での実装に関するガイドおよび詳細な情報については、「Amazon QLDB の PartiQL リファレンス」を参照してください。

以下のトピックでは、QLDB での PartiQL の使用方法のクイックリファレンスを示します。

トピック

- QLDB における PartiQL のクイックヒント
- コマンド
- システム定義ビュー
- 基本的な構文ルール
- PartiQL エディタのキーボードショートカット

QLDB における PartiQL のクイックヒント

以下は、QLDB で PartiQL を使用するためのヒントとベストプラクティスの簡単な要約です。

- 同時実行性とトランザクション制限を理解する - SELECT クエリを含むすべてのステートメントはオプティミスティック同時実行制御 (OCC、Optimistic Concurrency Control) 競合およびトランザクション制限 (30 秒のトランザクションタイムアウトなど) の対象になります。
- インデックスの使用 - 高基数インデックスを使用し、ターゲットとなるクエリを実行して、ステートメントを最適化し、すべてのテーブルスキャンを回避します。詳細については、「クエリパフォーマンスの最適化」を参照してください。
- 等価述語の使用 - インデックス付きルックアップには等価演算子 (= または IN) が必要です。不等価演算子 (<、>、LIKE、BETWEEN) はインデックス付きルックアップの対象にならず、すべてのテーブルスキャンが実行されます。
- 内部結合のみを使用する - QLDB は現在、内部結合のみをサポートしています。ベストプラクティスとして、結合するテーブルごとにインデックス付けされたフィールドで結合します。結合基準と等価述語の両方に高基数インデックスを選択します。

コマンド

QLDB は、以下の PartiQL コマンドをサポートしています。

データ定義言語 (DDL)

コマンド	説明
CREATE INDEX	テーブルの最上位のドキュメントフィールドのインデックスを作成します。
CREATE TABLE	テーブルを作成します。
DROP INDEX	テーブルからインデックスを削除します。
DROP TABLE	既存のテーブルを無効にします。
テーブルの削除の取り消し	非アクティブなテーブルを再度有効にします。

データ操作言語 (DML)

コマンド	説明
DELETE	ドキュメントの新しい最終リビジョンを作成して、アクティブなドキュメントを削除済みとしてマークします。
FROM (INSERT、REMOVE、または SET)	セマンティック的には UPDATE と同じです。
INSERT	テーブルに 1 つまたは複数のドキュメントを追加します。
SELECT	1 つまたは複数のテーブルからデータを取得します。
UPDATE	ドキュメント内の特定の要素を更新、挿入、または削除します。

DML ステートメントの例

INSERT

```
INSERT INTO VehicleRegistration VALUE
{
    'VIN' : 'KM8SRDHF6EU074761', --string
    'RegNum' : 1722, --integer
    'PendingPenaltyTicketAmount' : 130.75, --decimal
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' }
        ]
    },
    'ValidToDate' : `2020-06-25T` --Ion timestamp literal with day precision
}
```

UPDATE-INSERT

```
UPDATE Vehicle AS v
```

```
INSERT INTO v VALUE 26500 AT 'Mileage'
WHERE v.VIN = '1N4AL11D75C109151'
```

UPDATE-REMOVE

```
UPDATE Person AS p
REMOVE p.Address
WHERE p.GovId = '111-22-3333'
```

SELECT — 相関サブクエリ

```
SELECT r.VIN, o.SecondaryOwners
FROM VehicleRegistration AS r, @r.Owners AS o
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

SELECT — 内部結合

```
SELECT v.Make, v.Model, r.Owners
FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

SELECT — BY 句を使用してドキュメント ID を取得する

```
SELECT r_id FROM VehicleRegistration AS r BY r_id
WHERE r.VIN = '1HVBBAANXWH544237'
```

システム定義ビュー

QLDB は、テーブルの以下のシステム定義ビューをサポートしています。

ビュー	説明
table_name	ユーザーデータの現在の状態のみが含まれる、テーブルのデフォルトのユーザービュー。
_ql_committed_ *table_name*	ユーザーデータとシステム生成メタデータ (ドキュメント ID など) の両方の現在の状態が含まれる、テーブルの完全システム定義のコミット済みビュー。

ビュー	説明
history(*table_name*)	テーブルのリビジョン履歴全体を返す、組み込みの履歴関数。

基本的な構文ルール

QLDB は、PartiQL の以下の基本的な構文ルールをサポートしています。

文字	説明
'	一重引用符は、Amazon Ion 構造内の文字列値、またはフィールド名を表します。
"	二重引用符は、引用符で囲まれた識別子 (テーブル名として使用される予約語など) を表します。
`	バックティックは Ion リテラル値を表します。
.	ドット表記で、親構造のフィールド名にアクセスします。
[]	角括弧は Ion の `list` を定義します。または、既存のリストのゼロから始まる序数を表します。
{}	中括弧は Ion の `struct` を定義します。
<< >>	二重山括弧は、順序付けられていないコレクションである PartiQL バッグを定義します。バッグを使うと、複数のドキュメントをテーブルに挿入できます。
大文字と小文字の区別	フィールド名やテーブル名など、QLDB システムオブジェクト名はすべて、大文字と小文字が区別されます。

PartiQL エディタのキーボードショートカット

QLDB コンソールの PartiQL エディタは、以下のキーボードショートカットをサポートしています。

アクション	macOS	Windows
実行	Cmd+Return	Ctrl+Enter
コメント	Cmd+/	Ctrl+/
Clear	Cmd+Shift+Delete	Ctrl+Shift+Delete

を使用した Amazon QLDB へのアクセス AWS CLI （ 管理 API のみ ）

AWS Command Line Interface （ AWS CLI) を使用して、コマンドライン AWS のサービス から複数の を制御し、スクリプトを使用して自動化できます。必要に応じて、 を 1 回限りのオペレーション AWS CLI に使用できます。また、ユーティリティスクリプト内に Amazon QLDB オペレーションを埋め込むときにも使用できます。

CLI アクセスには、アクセスキー ID とシークレットアクセスキーが必要です。長期のアクセスキーの代わりに一時的な認証情報をできるだけ使用します。一時的な認証情報には、アクセスキー ID、シークレットアクセスキー、および認証情報の失効を示すセキュリティトークンが含まれています。詳細については、「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」を参照してください。

AWS CLIの QLDB で使用できるすべてのコマンドの一覧と使用例については、「AWS CLI コマンドリファレンス」を参照してください。

> **Note**
>
> は、 にリストされているqldb管理 API オペレーション AWS CLI のみをサポートしますAmazon QLDB API リファレンス。この API は、台帳リソースの管理および非トランザクションデータオペレーションにのみ使用されます。
> コマンドラインインターフェイスで qldb-session API を使用してデータトランザクションを実行するには、「QLDB シェルを使用した Amazon QLDB へのアクセス (データ API のみ)」を参照してください。

トピック

- AWS CLIのインストールおよび設定

- QLDB AWS CLI での の使用

AWS CLIのインストールおよび設定

は Linux、macOSで AWS CLI 実行されます。「AWS Command Line Interface ユーザーガイド」の手順に従ってインストールして設定します。

1. の最新バージョンのインストールまたは更新 AWS CLI
2. Quick Setup

QLDB AWS CLI での の使用

コマンドラインの形式は、Amazon QLDB オペレーション名の後に、そのオペレーション用のパラメータが続く形式です。は、JSON に加えてパラメータ値の短縮構文 AWS CLI をサポートしています。

help を使用すると、QLDB で使用可能なすべてのコマンドを列挙できます。

```
aws qldb help
```

また help を使用して、特定コマンドを記述したり、その用法の詳細を確認したりすることもできます。

```
aws qldb create-ledger help
```

たとえば、台帳を作成するには以下のようにします。

```
aws qldb create-ledger --name my-example-ledger --permissions-mode STANDARD
```

QLDB シェルを使用した Amazon QLDB へのアクセス (データ API のみ)

Amazon QLDB には、トランザクションデータ API とやり取りするためのコマンドラインシェルが用意されています。QLDB シェルを使用すると、台帳データに対して PartiQL ステートメントを実行できます。

このシェルの最新バージョンは Rust で記述されており、GitHub リポジトリのデフォルト `main` ブランチの awslabs/amazon-qldb-shell でオープンソースとして公開されています。Python バージョン (v1) も、`master` ブランチの同じリポジトリに用意されています。

> **Note**
>
> Amazon QLDB シェルは、`qldb-session` トランザクションデータ API のみをサポートしています。この API は、QLDB 台帳に対して PartiQL ステートメントを実行するためにのみ使用します。
> コマンドラインインターフェイスを使用して `qldb` 管理 API オペレーションとやり取りするには、「を使用した Amazon QLDB へのアクセス AWS CLI （管理 API のみ）」を参照してください。

このツールは、アプリケーションに組み込んだり、本番環境で採用したりすることを意図したものではありません。このツールの目的は、QLDB と PartiQL を迅速に実験できるようにすることです。

以下のセクションでは、QLDB シェルの使用を開始する方法について説明します。

トピック

前提条件

QLDB シェルを開始する前に、以下の操作を実行する必要があります。

1. 「Amazon QLDB へのアクセス」にある AWS の設定手順に従います。これには以下が含まれます。

1. AWS にサインアップする。
2. QLDB の適切なアクセス許可を持つユーザーを作成します。
3. 開発に必要なプログラムへのアクセスを提供します。

2. AWS の認証情報とデフォルトの AWS リージョン を設定します。指示については、「AWS Command Line Interface ユーザーガイド」の「設定の基本」を参照してください。

   利用可能なリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

3. STANDARD アクセス許可モードの台帳では、適切なテーブルで PartiQL ステートメントを実行するために、アクセス許可を付与する IAM ポリシーを作成します。これらのポリシーを作成する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

シェルのインストール

QLDB シェルの最新バージョンをインストールするには、GitHub の README.md ファイルを参照してください。QLDB では、GitHub リポジトリの「Releases」セクションにある、Linux、macOS、Windows 向け事前構築済みファイルを利用できます。

macOS の場合、`aws/tap` Homebrew タップを使用して、シェルを統合します。Homebrew を使用してシェルを macOS にインストールするには、次のコマンドを実行します。

```
$ xcode-select --install # Required to use Homebrew
$ brew tap aws/tap # Add AWS as a Homebrew tap
$ brew install qldbshell
```

構成

インストール後、シェルは `$XDG_CONFIG_HOME/qldbshell/config.ion` にあるデフォルト設定ファイルを、初期化中にロードします。Linux と macOS の場合、このファイルは通常 `~/.config/qldbshell/config.ion` にあります。そのようなファイルが存在しない場合、シェルはデフォルト設定で実行されます。

インストール後、`config.ion` ファイルを手動で作成できます。この設定ファイルでは、Amazon Ion データ形式が使用されます。次は、最小の `config.ion` ファイルの例です。

```
{
```

```
  default_ledger: "my-example-ledger"
}
```

設定ファイルに default_ledger が設定されていない場合、シェルを呼び出す際に --ledger パラメータが必要です。設定オプションの詳細なリストについては、GitHub の README.md ファイルを参照してください。

シェルの呼び出し

コマンドラインターミナルで特定の台帳に対して QLDB シェルを呼び出すには、以下のコマンドを実行します。*my-example-ledger* を台帳名に置き換えます。

```
$ qldb --ledger my-example-ledger
```

このコマンドはデフォルトの AWS リージョン に接続します。リージョンを明示的に指定するには、次のセクションで説明するように、--region または --qldb-session-endpoint パラメータを指定してコマンドを実行できます。

qldb シェルセッションを呼び出した後、次のインプットタイプで入力できます。

- シェルコマンド
- 個別のトランザクション内の単一 PartiQL ステートメント
- トランザクション内の複数の PartiQL ステートメント

シェルパラメータ

シェルの呼び出しに使用できるフラグとオプションの詳細なリストを確認するには、次のように、--help フラグを使用して qldb コマンドを実行します。

```
$ qldb --help
```

qldb コマンドのキーフラグとオプションのいくつかを次に示します。これらのオプションパラメータを追加して、AWS リージョン、認証情報プロファイル、エンドポイント、結果の形式、およびその他の設定オプションを上書きできます。

使用方法

```
$ qldb [FLAGS] [OPTIONS]
```

FLAGS

-h, **--help**

ヘルプ情報を表示します。

-v, **--verbose**

ログ記録の詳細を設定します。デフォルトの場合、シェルはエラーのみをログに記録します。詳細レベルを上げるには、この引数を繰り返します (例: -vv)。最高レベルは -vvv であり、これは trace 詳細レベルに相当します。

-V, **--version**

バージョン情報を表示します。

OPTIONS

-l, **--ledger** ***LEDGER_NAME***

接続先の台帳の名前です。config.ion ファイルに default_ledger が設定されていない場合、このシェルパラメータが必要です。このファイルでは、リージョンなどの追加オプションを設定できます。

-c, **--config** ***CONFIG_FILE***

シェル設定オプションを定義できるファイルです。形式の詳細と、設定オプションの詳細なリストについては、GitHub の README.md ファイルを参照してください。

-f, **--format ion|table**

クエリ結果の出力形式です。デフォルトは ion です。

-p, **--profile** ***PROFILE***

認証に使用する AWS 認証情報プロファイルの場所です。

指定しない場合、シェルは、~/.aws/credentials にあるデフォルトの AWS プロファイルを使用します。

-r, **--region** ***REGION_CODE***

接続先の QLDB 台帳の AWS リージョン コードです。例: us-east-1。

指定しない場合、シェルは、AWS プロファイルで指定されたデフォルトの AWS リージョン に接続します。

-s, **--qldb-session-endpoint** ***QLDB_SESSION_ENDPOINT***

接続先の `qldb-session` API エンドポイントです。

QLDB リージョンとエンドポイントの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

コマンドリファレンス

`qldb` セッションを呼び出すと、シェルでは、次のキーとデータベースコマンドがサポートされます。

シェルキー

キー	機能の説明
Enter	ステートメントを実行します。
Escape+Enter (macOS、Linux) Shift+Enter (Windows)	複数の行にまたがるステートメントを入力するために、新しい行を作成します。複数行の入力テキストをコピーして、シェルに貼り付けることもできます。 macOS で、Escape ではなく Option をメタキーとしてセットアップする手順については、OS X Daily サイトを参照してください。
Ctrl+C	現在のコマンドをキャンセルします。
Ctrl+D	ファイルの終了 (EOF) を通知し、シェルの現在のレベルを終了します。アクティブなトランザクションでない場合は、シェルを終了します。アクティブなトランザクションの場合、トランザクションを中断します。

シェルデータベースコマンド

コマンド	機能の説明
`help`	ヘルプ情報を表示します。
`begin`	トランザクションを開始します。
`start transaction`	
`commit`	トランザクションを台帳のジャーナルにコミットします。
`abort`	トランザクションを停止し、行った変更を拒否します。
`exit`	シェルを終了します。
`quit`	

> **Note**
>
> すべての QLDB シェルコマンドで、大文字と小文字の区別がありません。

個別のステートメントの実行

README.md にリストされているデータベースコマンドとシェルメタコマンドを除き、シェルは、入力された各コマンドを個別の PartiQL ステートメントとして解釈します。デフォルトの場合、シェルでは、`auto-commit` モードが有効になります。このモードは設定可能です。

`auto-commit` モードの場合、シェルは、独自のトランザクションで各ステートメントを暗黙的に実行し、エラーがない場合はトランザクションを自動的にコミットします。つまり、ステートメントの実行のたびに、`start transaction` (または `begin`) を実行して、手動で `commit` する必要はありません。

トランザクションの管理

QLDB シェルでは、トランザクションを手動で制御することもできます。トランザクション内の複数のステートメントは、インタラクティブに実行することも、コマンドとステートメントを順番にバッチ処理して非インタラクティブに実行することもできます。

インタラクティブトランザクション

インタラクティブトランザクションを実行するには、次の手順に従います。

1. トランザクションを開始するには、begin コマンドを入力します。

   ```
   qldb> begin
   ```

   トランザクションを開始すると、次のコマンドプロンプトがシェルに表示されます。

   ```
   qldb *>
   ```

2. その後、入力した各ステートメントが、同じトランザクションで実行されます。
   - 例えば、単一のステートメントを次のように実行できます。

     ```
     qldb *> SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'
     ```

     Enter を押すと、ステートメントの結果がシェルに表示されます。
   - 複数のステートメントまたはコマンドを、次のようにセミコロン (;) の記号で区切って入力することもできます。

     ```
     qldb *> SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'; commit
     ```

3. トランザクションを終了するには、次のコマンドのいずれかを入力します。
   - commit コマンドを入力すると、トランザクションが台帳のジャーナルにコミットされます。

     ```
     qldb *> commit
     ```
   - abort コマンドを入力すると、トランザクションが停止され、行った変更が拒否されます。

     ```
     qldb *> abort
     ```

```
transaction was aborted
```

トランザクションタイムアウト制限

QLDB のトランザクションタイムアウト制限に準拠するには、トランザクションをインタラクティブに行います。開始後 30 秒以内にトランザクションをコミットしない場合、QLDB ではトランザクションの有効期限が自動的に切れ、トランザクション中に行った変更がすべて拒否されます。

その場合、ステートメントの結果ではなく、有効期限切れのエラーメッセージが表示され、シェルは通常のコマンドプロンプトに戻ります。再試行するには、begin コマンドを再度実行して、新しいトランザクションを開始します。

```
transaction failed after 1 attempts, last error: communication failure:
 Transaction 2UMpiJ5hh7WLjVgEiMLOoO has expired
```

非インタラクティブトランザクション

次のようにコマンドとステートメントを順番にバッチ処理することで、複数のステートメントを使用する完全なトランザクションを実行できます。

```
qldb> begin; SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'; SELECT * FROM
 Person p, DriversLicense l WHERE p.GovId = l.LicenseNumber; commit
```

各コマンドとステートメントは、セミコロン (;) の記号で区切る必要があります。トランザクション内のいずれかのステートメントが有効でない場合、シェルはトランザクションを自動的に拒否します。入力した後続のステートメントを続行しません。

複数のトランザクションも設定できます。

```
qldb> begin; statement1; commit; begin; statement2; statement3; commit
```

前の例と同様に、トランザクションが失敗すると、シェルは、入力した後続のトランザクションまたはステートメントを処理しません。

トランザクションを終了しない場合、シェルがインタラクティブモードに切り替わり、次のコマンドまたはステートメントの入力が求められます。

```
qldb> begin; statement1; commit; begin
```

```
qldb *>
```

シェルの終了

現在の qldb シェルセッションを終了するには、そのシェルでトランザクションを実行していないときに、exit または quit コマンドを入力するか、キーボードショートカットの Ctrl+D を使用します。

```
qldb> exit
$
```

```
qldb> quit
$
```

例

QLDB での PartiQL ステートメントの記述については、「Amazon QLDB の PartiQL リファレンス」を参照してください。

Example

以下の例では、基本的なコマンドの一般的なシーケンスを示しています。

> **Note**
>
> QLDB シェルは、この例の各 PartiQL ステートメントを独自のトランザクションで実行します。
> この例では、台帳 test-ledger がすでに存在し、アクティブであると想定しています。

```
$ qldb --ledger test-ledger --region us-east-1

qldb> CREATE TABLE TestTable
qldb> INSERT INTO TestTable `{"Name": "John Doe"}`
qldb> SELECT * FROM TestTable
qldb> DROP TABLE TestTable
qldb> exit
```

API を使用した Amazon QLDB へのアクセス

AWS Management Console および AWS Command Line Interface （ AWS CLI) を使用して、Amazon QLDB とインタラクティブに連携できます。ただし、QLDB を最大限に活用するには、QLDB ドライバーまたは AWS SDK を使用してアプリケーションコードを記述し、 APIsを使用して台帳とやり取りできます。

このドライバーを使用すると、アプリケーションはトランザクションデータ API を使用して QLDB とやり取りできるようになります。 AWS SDK は、QLDB リソース管理 API とのやり取りをサポートします。これらの API の詳細については、「Amazon QLDB API リファレンス」を参照してください。

ドライバーは、QLDB のサポートを Java、.NET、Go、Node.js、Python で提供しています。このような言語を使用してすばやく開始する方法については、「Amazon QLDB ドライバーの開始方法」を参照してください。

アプリケーションで QLDB ドライバーまたは AWS SDK を使用する前に、プログラムによるアクセスを許可する必要があります。詳細については、「プログラマチックアクセス権を付与する」を参照してください。

Amazon QLDB コンソールの使用開始方法

このチュートリアルで説明する手順では、最初の Amazon QLDB 台帳を作成し、そこにテーブルとサンプルデータをロードします。このシナリオで作成するサンプルの台帳は、車両登録に関するすべての履歴情報を追跡する車両管理局 (DMV) 用アプリケーションのデータベースです。

資産の履歴管理は QLDB の一般的なユースケースであり、これは Amazon QLDB に台帳データベースの有用性を示すさまざまなシナリオやオペレーションが含まれているためです。QLDB を使用すると、SQL のようなクエリ機能をサポートするドキュメント指向のデータベースで、データへの変更の全履歴に直接アクセスし、クエリを実行し、検証することができます。

このチュートリアルでは、以下のトピックで、車両登録を追加、変更し、登録の変更履歴を確認する方法について説明します。このガイドでは、登録ドキュメントを暗号的に検証する方法も示します。最後に、リソースをクリーンアップしてサンプル台帳を削除します。

チュートリアルの各ステップには、AWS Management Console を使用する方法の説明があります。

トピック

チュートリアルの前提条件と考慮事項

この Amazon QLDB チュートリアルを開始する前に、以下の前提条件を満たしていることを確認してください。

1. まだ行っていない場合は、「Amazon QLDB へのアクセス」の AWS の設定手順に従います。これらのステップには、AWS へのサインアップ手順と、管理ユーザーの作成手順が含まれます。

2. 「アクセス許可のセットアップ」の指示に従って、QLDB リソースの IAM アクセス許可を設定します。このチュートリアルのすべての手順を完了するには、AWS Management Console を介して台帳リソースへのフル管理アクセス権が必要です。

   > **Note**
   >
   > 完全な AWS 管理者権限を持つユーザーとして既にサインインしている場合は、このステップを省略できます。

3. (オプション) QLDB は、AWS Key Management Service (AWS KMS) のキーを使用して保管中のデータを暗号化します。次のいずれかの種類の AWS KMS keys を選択できます。

   - AWS 所有の KMS キー - ユーザーに代わって AWS が所有、管理している KMS キーを使用します。これはデフォルトのオプションであり、追加のセットアップは必要ありません。
   - カスタマーマネージド KMS キー - 作成、所有、管理しているアカウントで、対称暗号化 KMS キーを使用します。QLDB は非対称キーをサポートしていません。

     このオプションを使用するには、KMS キーを作成するか、アカウント内の既存のキーを使用する必要があります。カスタマーマネージドキーの作成方法については、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 KMS キーの作成」を参照してください。

     カスタマーマネージド KMS キーは、ID、エイリアス、または Amazon リソースネーム (ARN) を使用して指定できます。詳細については、「AWS Key Management Service デベロッパーガイド」の「キー識別子 (KeyId)」を参照してください。

     > **Note**
     >
     > クロスリージョンキーはサポートされていません。指定した KMS キーは台帳と同じ AWS リージョン に存在する必要があります。

アクセス許可のセットアップ

このステップでは、AWS アカウント のすべての QLDB リソースに対して、コンソールからフルアクセス許可を設定します。これらのアクセス許可を迅速に付与するには、AWS 管理ポリシー AmazonQLDBConsoleFullAccess を使用してください。

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「シークレットの作成と管理」の手順に従ってください。

- ID プロバイダーを通じて IAM で管理されているユーザー:

  ID フェデレーションのロールを作成する。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが設定できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (非推奨) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス許可の追加」の指示に従います。

> **Important**
>
> このチュートリアルでは、すべての QLDB リソースへのフル管理アクセス権をユーザーに付与します。ただし、本稼働のユースケースについては、最小特権の付与に関するセキュリティのベストプラクティスに従ってください。または、タスクの実行に必要なアクセス許可のみを付与します。例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

`vehicle-registration` という名前の台帳を作成するには、「ステップ 1: 新しい台帳を作成する」に進みます。

ステップ 1: 新しい台帳を作成する

このステップでは、`vehicle-registration` という名前の新しい Amazon QLDB 台帳を作成します。次に、台帳のステータスが [Active (アクティブ)] であることを確認します。台帳に追加したタグを確認することもできます。

台帳を作成すると、デフォルトで削除保護が有効になります。削除保護は、ユーザーによる台帳の削除を防ぐ QLDB の機能です。QLDB API または AWS Command Line Interface (AWS CLI) を使用して台帳を作成するときに、削除保護を無効にできます。

新しい台帳を作成するには

1. AWS Management Console にサインインして、Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペインで、[Getting started (開始方法)] を選択します。
3. [Create your first ledger (最初の台帳の作成)] カードで、[Create Ledger (台帳の作成)] を選択します。
4. [Create Ledger (台帳の作成)] ページで、以下の操作を実行します。

   - [Ledger information] (台帳情報) - [Ledger name] (台帳名) は、**`vehicle-registration`** であらかじめ入力しておく必要があります。
   - [Permissions mode] (アクセス許可モード) – 台帳に割り当てるアクセス許可モード。以下のオプションのいずれかを選択します。
     - [Allow all] (すべて許可) - 台帳の API レベル詳細度でアクセスコントロールを可能にする、レガシーアクセス許可モード。

       このモードは、この台帳の `SendCommand` API アクセス許可を持つユーザーが、指定された台帳の任意のテーブルで、すべての PartiQL コマンド (すなわち `ALLOW_ALL`) を実行することを許可します。このモードでは、台帳用に作成したテーブルレベルまたはコマンドレベルの IAM アクセス許可ポリシーはすべて無視されます。
     - [Standard] (標準) - (推奨) 台帳、テーブル、PartiQL コマンドの、より詳細なレベルのアクセスコントロールを可能にするアクセス許可モード。台帳データのセキュリティを最大化する、このアクセス許可モードの使用を強く推奨します。

       デフォルトでは、このモードは、この台帳内の任意のテーブルで、任意の PartiQL コマンドを実行するすべてのリクエストを拒否します。PartiQL コマンドを許可するには、台帳の `SendCommand` API アクセス許可に加えて、特定のテーブルリソースと PartiQL アクション用の IAM アクセス許可ポリシーを作成する必要があります。詳細については、Amazon QLDB の標準アクセス許可モードの開始方法を参照してください。
   - [Encrypt data at rest] (保管時のデータの暗号化) — 保管中のデータの暗号化に使用する AWS Key Management Service (AWS KMS) のキー。以下のオプションのいずれかを選択します。
     - [AWS 所有の KMS キーを使用する] - ユーザーに代わって AWS が所有、管理している KMS キーを使用します。これはデフォルトのオプションであり、追加のセットアップは必要ありません。
     - [異なる AWS KMS キーを選択] - 作成、所有、管理しているアカウントで、対称暗号化 KMS キーを使用します。

AWS KMS コンソールを使用して新しいキーを作成するには、[AWS KMS キーを作成] を選択します。詳細については、AWS Key Management Service デベロッパーガイドの「非対称 KMS キーを作成する」を参照してください。

既存の KMS キーを使用するには、ドロップダウンリストからキーを選択するか、KMS キー ARN を指定します。

- [Tags] (タグ) - (オプション) タグをキーと値の組み合わせとしてアタッチすることで、メタデータを台帳に追加します。台帳にタグを追加すると、台帳の整理と識別がしやすくなります。詳細については、「Amazon QLDB リソースのタグ付け」を参照してください。

  [Add tag (タグの追加)] を選択し、必要に応じて、任意のキーと値の組み合わせを入力します。

5. すべての設定が正しいことを確認したら、[Create ledger (台帳の作成)] を選択します。

> **Note**
>
> QLDB の台帳には、台帳のステータスが [Active] (アクティブ) になるとアクセスできるようになります。これには数分間かかる場合があります。

6. [Ledgers (台帳)] のリストで、`vehicle-registration` を見つけて、台帳のステータスが [Active (アクティブ)] であることを確認します。
7. (オプション) `vehicle-registration` 台帳名を選択します。[vehicle-registration (車両登録)] 台帳の詳細ページで、台帳に追加したタグが [Tags (タグ)] カードに表示されていることを確認します。このコンソールページを使用して、台帳タグを編集することもできます。

`vehicle-registration` 台帳にテーブルを作成するには、「ステップ 2: 台帳にテーブル、インデックス、サンプルデータを作成する」に進みます。

ステップ 2: 台帳にテーブル、インデックス、サンプルデータを作成する

Amazon QLDB 台帳がアクティブになったら、車両、車両の所有者、登録情報に関するデータのテーブルを作成できるようになります。テーブルとインデックスを作成した後、これらにデータをロードできます。

このステップでは、`vehicle-registration` 台帳に 4 つのテーブルを作成します。

- `VehicleRegistration`
- `Vehicle`
- `Person`
- `DriversLicense`

以下のインデックスも作成します。

テーブル名	フィールド
`VehicleRegistration`	`VIN`
`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`
`Person`	`GovId`
`DriversLicense`	`LicensePlateNumber`
`DriversLicense`	`PersonId`

QLDB コンソールを使用して、インデックス付きのこれらのテーブルを自動的に作成し、そこにサンプルデータをロードできます。または、コンソールの [PartiQL エディタ] を使用して、各 PartiQL ステートメントをステップごとに手動で実行できます。

自動オプション

テーブル、インデックス、およびサンプルデータを作成するには

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペインで、[Getting started (開始方法)] を選択します。
3. [Sample application data] (サンプルアプリケーションデータ) カードの [Automatic option] (自動オプション) で、台帳のリストから `vehicle-registration` を選択します。
4. [Load sample data (サンプルデータのロード)] を選択します。

   オペレーションが正常に終了すると、コンソールに「Sample data loaded (サンプルデータがロードされました)」というメッセージが表示されます。

このスクリプトは、すべてのステートメントを単一のトランザクションで実行します。トランザクションの一部が失敗すると、すべてのステートメントがロールバックされ、該当するエラーメッセージが表示されます。問題に対処した後、オペレーションを再試行できます。

> **Note**
>
> - 考えられる原因の 1 つとして、重複するテーブル複製を作成しようとすると、トランザクションが失敗することがあります。サンプルデータをロードするためのリクエストは、VehicleRegistration、Vehicle、Person、DriversLicense のいずれかのテーブル名が台帳に既に存在する場合、失敗します。
>
>   代わりに、このサンプルデータを空の台帳にロードしてみてください。
> - このスクリプトは、パラメータ化された INSERT ステートメントを実行します。したがって、これらの PartiQL ステートメントは、リテラルデータではなくバインドパラメータを使用してジャーナルブロックに記録されます。たとえば、ジャーナルブロックに次のステートメントがあるとします。ここで、疑問符 (?) はドキュメントコンテンツの変数プレースホルダーです。
>
>   ```
>   INSERT INTO Vehicle ?
>   ```

手動オプション

空の PrimaryOwner フィールドを含む VehicleRegistration と、空の PersonId フィールドを含む DriversLicense にドキュメントを挿入します。後で、Person テーブルからシステムによって割り当てられたドキュメント id をこれらのフィールドに入力します。

> **Tip**
>
> ベストプラクティスとして、このドキュメント id メタデータフィールドを外部キーとして使用します。詳細については、「ドキュメントのメタデータのクエリの実行」を参照してください。

テーブル、インデックス、およびサンプルデータを作成するには

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。

2. ナビゲーションペイン内で [PartiQL エディタ] を選択します。
3. `vehicle-registration` 台帳を選択します。
4. まず、4 つのテーブルを作成します。QLDB ではオープンのコンテンツをサポートしており、スキーマを強制しないため、属性やデータ型は指定しません。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。このステートメントを実行するには、キーボードショートカット Ctrl+Enter (Windows の場合) または Cmd+Return (macOS の場合) を使用することもできます。キーボードショートカットの詳細については、「PartiQL エディタのキーボードショートカット」を参照してください。

   ```
   CREATE TABLE VehicleRegistration
   ```

   以下の各項目について、このステップを繰り返します。

   ```
   CREATE TABLE Vehicle
   ```

   ```
   CREATE TABLE Person
   ```

   ```
   CREATE TABLE DriversLicense
   ```

5. 次に、各テーブルのクエリパフォーマンスを最適化するインデックスを作成します。

   > **Important**
   >
   > ドキュメントを効率的に検索するには、インデックスが必要です。インデックスがないと、QLDB はドキュメントを読み取るときにテーブルスキャンを実行する必要があります。これにより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
   > テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子 (= または IN) を使用する WHERE 述語句でステートメントを実行する必要があります。詳細については、「クエリパフォーマンスの最適化」を参照してください。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

```
CREATE INDEX ON VehicleRegistration (VIN)
```

以下に対してこの手順を繰り返します。

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

```
CREATE INDEX ON Person (GovId)
```

```
CREATE INDEX ON DriversLicense (LicensePlateNumber)
```

```
CREATE INDEX ON DriversLicense (PersonId)
```

6. インデックスを作成したら、テーブルへのデータのロードを開始することができます。このステップでは、台帳が追跡している車両の所有者に関する個人情報を含む Person テーブルにドキュメントを挿入します。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

```
INSERT INTO Person
<< {
    'FirstName' : 'Raul',
    'LastName' : 'Lewis',
    'DOB' : `1963-08-19T`,
    'GovId' : 'LEWISR261LL',
    'GovIdType' : 'Driver License',
    'Address' : '1719 University Street, Seattle, WA, 98109'
},
{
    'FirstName' : 'Brent',
    'LastName' : 'Logan',
    'DOB' : `1967-07-03T`,
    'GovId' : 'LOGANB486CG',
    'GovIdType' : 'Driver License',
    'Address' : '43 Stockert Hollow Road, Everett, WA, 98203'
},
{
```

```
    'FirstName' : 'Alexis',
    'LastName' : 'Pena',
    'DOB' : `1974-02-10T`,
    'GovId' : '744 849 301',
    'GovIdType' : 'SSN',
    'Address' : '4058 Melrose Street, Spokane Valley, WA, 99206'
},
{
    'FirstName' : 'Melvin',
    'LastName' : 'Parker',
    'DOB' : `1976-05-22T`,
    'GovId' : 'P626-168-229-765',
    'GovIdType' : 'Passport',
    'Address' : '4362 Ryder Avenue, Seattle, WA, 98101'
},
{
    'FirstName' : 'Salvatore',
    'LastName' : 'Spencer',
    'DOB' : `1997-11-15T`,
    'GovId' : 'S152-780-97-415-0',
    'GovIdType' : 'Passport',
    'Address' : '4450 Honeysuckle Lane, Seattle, WA, 98101'
} >>
```

7. 次に、各車両の所有者の運転免許証情報を含むドキュメントを DriversLicense テーブルに入力します。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

```
INSERT INTO DriversLicense
<< {
    'LicensePlateNumber' : 'LEWISR261LL',
    'LicenseType' : 'Learner',
    'ValidFromDate' : `2016-12-20T`,
    'ValidToDate' : `2020-11-15T`,
    'PersonId' : ''
},
{
    'LicensePlateNumber' : 'LOGANB486CG',
    'LicenseType' : 'Probationary',
    'ValidFromDate' : `2016-04-06T`,
    'ValidToDate' : `2020-11-15T`,
    'PersonId' : ''
```

```
},
{
    'LicensePlateNumber' : '744 849 301',
    'LicenseType' : 'Full',
    'ValidFromDate' : `2017-12-06T`,
    'ValidToDate' : `2022-10-15T`,
    'PersonId' : ''
},
{
    'LicensePlateNumber' : 'P626-168-229-765',
    'LicenseType' : 'Learner',
    'ValidFromDate' : `2017-08-16T`,
    'ValidToDate' : `2021-11-15T`,
    'PersonId' : ''
},
{
    'LicensePlateNumber' : 'S152-780-97-415-0',
    'LicenseType' : 'Probationary',
    'ValidFromDate' : `2015-08-15T`,
    'ValidToDate' : `2021-08-21T`,
    'PersonId' : ''
} >>
```

8. 次に、車両登録ドキュメントを VehicleRegistration テーブルに入力します。これらのドキュメントには、代表所有者と共有所有者を格納する、ネストされた Owners 構造が含まれています。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

```
INSERT INTO VehicleRegistration
<< {
    'VIN' : '1N4AL11D75C109151',
    'LicensePlateNumber' : 'LEWISR261LL',
    'State' : 'WA',
    'City' : 'Seattle',
    'PendingPenaltyTicketAmount' : 90.25,
    'ValidFromDate' : `2017-08-21T`,
    'ValidToDate' : `2020-05-11T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
```

```
{
    'VIN' : 'KM8SRDHF6EU074761',
    'LicensePlateNumber' : 'CA762X',
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75,
    'ValidFromDate' : `2017-09-14T`,
    'ValidToDate' : `2020-06-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
{
    'VIN' : '3HGGK5G53FM761765',
    'LicensePlateNumber' : 'CD820Z',
    'State' : 'WA',
    'City' : 'Everett',
    'PendingPenaltyTicketAmount' : 442.30,
    'ValidFromDate' : `2011-03-17T`,
    'ValidToDate' : `2021-03-24T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
{
    'VIN' : '1HVBBAANXWH544237',
    'LicensePlateNumber' : 'LS477D',
    'State' : 'WA',
    'City' : 'Tacoma',
    'PendingPenaltyTicketAmount' : 42.20,
    'ValidFromDate' : `2011-10-26T`,
    'ValidToDate' : `2023-09-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
{
    'VIN' : '1C4RJFAG0FC625797',
    'LicensePlateNumber' : 'TH393F',
    'State' : 'WA',
    'City' : 'Olympia',
```

```
    'PendingPenaltyTicketAmount' : 30.45,
    'ValidFromDate' : `2013-09-02T`,
    'ValidToDate' : `2024-03-19T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
} >>
```

9. 最後に、台帳に登録されている車両について記述したドキュメントを Vehicle テーブルに入力します。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

```
INSERT INTO Vehicle
<< {
    'VIN' : '1N4AL11D75C109151',
    'Type' : 'Sedan',
    'Year' : 2011,
    'Make' : 'Audi',
    'Model' : 'A5',
    'Color' : 'Silver'
},
{
    'VIN' : 'KM8SRDHF6EU074761',
    'Type' : 'Sedan',
    'Year' : 2015,
    'Make' : 'Tesla',
    'Model' : 'Model S',
    'Color' : 'Blue'
},
{
    'VIN' : '3HGGK5G53FM761765',
    'Type' : 'Motorcycle',
    'Year' : 2011,
    'Make' : 'Ducati',
    'Model' : 'Monster 1200',
    'Color' : 'Yellow'
},
{
    'VIN' : '1HVBBAANXWH544237',
    'Type' : 'Semi',
    'Year' : 2009,
```

```
    'Make' : 'Ford',
    'Model' : 'F 150',
    'Color' : 'Black'
},
{
    'VIN' : '1C4RJFAG0FC625797',
    'Type' : 'Sedan',
    'Year' : 2019,
    'Make' : 'Mercedes',
    'Model' : 'CLK 350',
    'Color' : 'White'
} >>
```

次に、SELECT ステートメントを使用すると、vehicle-registration 台帳のテーブルからデータを読み取ることができます。ステップ 3: 台帳のテーブルのクエリを実行する に進みます。

ステップ 3: 台帳のテーブルのクエリを実行する

Amazon QLDB 台帳にテーブルを作成し、データをロードした後は、クエリを実行して、挿入した車両登録データを確認できます。QLDB は PartiQL をクエリ言語として使用し、Amazon Ion をドキュメント指向のデータモデルとして使用します。

PartiQL は、Ion で動作するように拡張されたオープンソースの SQL 互換のクエリ言語です。PartiQL を使用すると、使い慣れた SQL 演算子を使用してデータを挿入、クエリ、および管理できます。Amazon Ion は JSON のスーパーセットです。Ion はオープンソースのドキュメントベースのデータ形式であり、構造化データ、半構造化データ、およびネストされたデータを柔軟に保存および処理できます。

このステップでは、SELECT ステートメントを使用して、vehicle-registration 台帳のテーブルからデータを読み取ります。

> **Warning**
>
> インデックス付きルックアップなしで QLDB でクエリを実行すると、完全なテーブルスキャンが呼び出されます。PartiQL は SQL 互換であるため、このようなクエリをサポートしています。ただし、QLDB の本番環境のユースケースではテーブルスキャンを実行しないでください。テーブルスキャンより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。

テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する必要があります (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`)。詳細については、「クエリパフォーマンスの最適化」を参照してください。

テーブルのクエリを実行するには

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [PartiQL エディタ] を選択します。
3. `vehicle-registration` 台帳を選択します。
4. クエリエディタウィンドウで、`Vehicle` テーブルに対して台帳に追加した特定の車両識別番号 (VIN) のクエリを実行する次のステートメントを入力し、[Run (実行)] を選択します。

   このステートメントを実行するには、キーボードショートカット Ctrl+Enter (Windows の場合) または Cmd+Return (macOS の場合) を使用することもできます。キーボードショートカットの詳細については、「PartiQL エディタのキーボードショートカット」を参照してください。

   ```
   SELECT * FROM Vehicle AS v
   WHERE v.VIN = '1N4AL11D75C109151'
   ```

5. 内部結合クエリを記述することができます。次のクエリの例では、`Vehicle` を `VehicleRegistration` と結合して、指定した `VIN` の登録済み車両の登録情報と属性を返します。

   次のステートメントを入力し、[Run (実行)] を選択します。

   ```
   SELECT v.VIN, r.LicensePlateNumber, r.State, r.City, r.Owners
   FROM Vehicle AS v, VehicleRegistration AS r
   WHERE v.VIN = '1N4AL11D75C109151'
   AND v.VIN = r.VIN
   ```

   `DriversLicense` テーブルと `Person` テーブルを結合して、台帳に追加された運転者に関連する属性を表示することもできます。

   以下に対してこの手順を繰り返します。

   ```
   SELECT * FROM Person AS p, DriversLicense AS l
   ```

```
WHERE p.GovId = l.LicensePlateNumber
```

vehicle-registration 台帳のテーブルのドキュメントの変更方法については、「ステップ 4: 台帳のドキュメントを変更する」を参照してください。

ステップ 4: 台帳のドキュメントを変更する

操作するデータを用意できたところで、Amazon QLDB で vehicle-registration 台帳のドキュメントに変更を加えることができます。ここでは、VIN が 1N4AL11D75C109151 の Audi A5 を例にします。この車の最初の所有者は、ワシントン州シアトルの Raul Lewis という名前のドライバーです。

Raul がワシントン州エバレットに住む Brent Logan にこの車を売るとします。その後、Brent と Alexis Pena が結婚することになりました。Brent は共同所有者として車両登録に Alexis を追加したいと考えています。このステップの以下のデータ操作言語 (DML) ステートメントは、台帳に適切な変更を行ってこれらのイベントを反映させる方法を示します。

> **Tip**
>
> ベストプラクティスとして、外部キーには、ドキュメントにシステムによって割り当てられた id を使用します。一意の識別子 (車両の VIN など) 用にフィールドを定義することはできますが、実際のドキュメントの一意の識別子はその id です。このフィールドはドキュメントのメタデータに含まれており、コミット済みビュー (テーブルのシステム定義のビュー) でクエリを実行できます。
>
> QLDB におけるビューの詳細については、「重要な概念」を参照してください。メタデータの詳細については、「ドキュメントのメタデータのクエリの実行」を参照してください。

ドキュメントに変更を加えるには

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [PartiQL エディタ] を選択します。
3. vehicle-registration 台帳を選択します。

> **Note**
>
> コンソールの自動 [Load sample data] (サンプルデータをロード) 機能を使用して台帳を設定する場合は、ステップ 6 に進みます。

4. サンプルデータをロードする INSERT ステートメントを手動で実行した場合は、次のステップに進みます。

   この車両の所有者として最初に Raul を登録するには、まず Person テーブルで、システムによって割り当てられたドキュメント id を検索することから始めます。このフィールドはドキュメントのメタデータに含まれており、テーブルのシステム定義のビュー (コミット済みビュー) でフィールドのクエリを実行できます。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

   ```
   SELECT metadata.id FROM _ql_committed_Person AS p
   WHERE p.data.FirstName = 'Raul' and p.data.LastName = 'Lewis'
   ```

   プレフィックス _ql_committed_ は、Person テーブルのコミット済みビューにクエリを実行することを示す予約済みプレフィックスです。このビューでは、データは data フィールドにネストされており、メタデータは metadata フィールドにネストされています。

5. 次に、この id を UPDATE ステートメントで使用して、VehicleRegistration テーブル内の該当するドキュメントを変更します。次のステートメントを入力し、[Run (実行)] を選択します。

   ```
   UPDATE VehicleRegistration AS r
   SET r.Owners.PrimaryOwner.PersonId = '294jJ3YUoH1IEEm8GSabOs' --replace with your
    id
   WHERE r.VIN = '1N4AL11D75C109151'
   ```

   このステートメントを発行することで、Owners フィールドが変更されていることを確認します。

   ```
   SELECT r.Owners FROM VehicleRegistration AS r
   WHERE r.VIN = '1N4AL11D75C109151'
   ```

6. 車両の所有権を工バレットに住む Brent に譲渡するには、まず次のステートメントを使用して、Brent の id を Person テーブルで検索します。

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Brent' and p.data.LastName = 'Logan'
```

次に、この id を使用して、VehicleRegistration テーブルの PrimaryOwner と City を更新します。

```
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = '7NmE8YLPbXc0IqesJy1rpR', --replace with your
 id
    r.City = 'Everett'
WHERE r.VIN = '1N4AL11D75C109151'
```

このステートメントを発行して、PrimaryOwner フィールドと City フィールドを変更したことを確認します。

```
SELECT r.Owners.PrimaryOwner, r.City
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

7. 車の共同所有者として Alexis を追加するには、Alexis の Person id を検索します。

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Alexis' and p.data.LastName = 'Pena'
```

次に、以下の FROM-INSERT DML ステートメントでこの id を SecondaryOwners リストに挿入します。

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners
    VALUE { 'PersonId' : '5Ufgdlnj06gF5CWcOIu64s' } --replace with your id
```

このステートメントを発行し、SecondaryOwners が変更されていることを確認します。

```
SELECT r.Owners.SecondaryOwners FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

これらの変更を vehicle-registration 台帳で確認するには、「ステップ 5: ドキュメントのリビジョン履歴を表示する」を参照してください。

ステップ 5: ドキュメントのリビジョン履歴を表示する

VIN が　1N4AL11D75C109151 の車両の登録データを変更した後、登録されているすべての所有者やその他の更新されたすべてのフィールドの履歴のクエリを実行できます。組み込みの履歴関数にクエリを実行することによって、挿入、更新、削除したドキュメントのすべてのリビジョンを表示できます。

履歴関数は、テーブルのコミット済みビューからリビジョンを返します。これには、アプリケーションのデータとそれに関連するメタデータの両方が含まれます。このメタデータには、それぞれのリビジョンがいつ、どの順番で作成されたか、またどのトランザクションによってコミットされたかが正確に示されます。

このステップでは、vehicle-registration 台帳の VehicleRegistration テーブルに含まれているドキュメントのリビジョン履歴のクエリを実行します。

リビジョン履歴を表示するには

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [PartiQL エディタ] を選択します。
3. vehicle-registration 台帳を選択します。
4. ドキュメントの履歴にクエリを実行するには、まずその一意の id を検索します。コミット済みビューにクエリを実行する以外に、ドキュメント id を取得する方法として、テーブルのデフォルトのユーザービューで BY キーワードを使用する方法があります。詳細については、「BY 句を使用したドキュメント ID のクエリの実行」を参照してください。

   クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

   ```
   SELECT r_id FROM VehicleRegistration AS r BY r_id
   WHERE r.VIN = '1N4AL11D75C109151'
   ```

5. 次に、この id の値を使用して、履歴関数にクエリを実行します。次のステートメントを入力し、[Run (実行)] を選択します。必要に応じて、id 値を独自のドキュメント ID に置き換えてください。

   ```
   SELECT h.data.VIN, h.data.City, h.data.Owners
   FROM history(VehicleRegistration) AS h
   ```

```
WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
```

> **Note**
>
> このチュートリアルでは、この履歴クエリはドキュメント ID ADR2LQq48kB9neZDupQrMm のすべてのリビジョンを返します。ただし、ベストプラクティスとして、履歴クエリをドキュメント ID と日付範囲 (開始時刻および終了時刻) の両方で修飾します。
>
> QLDB では、すべての SELECT クエリはトランザクションで処理され、トランザクションタイムアウト制限の対象になります。開始時刻と終了時刻を含む履歴クエリでは、日付範囲修飾のメリットが得られます。詳細については、「履歴関数」を参照してください。

履歴関数は、コミット済みビューと同じスキーマ内にあるドキュメントを返します。この例では、修正した車両登録データが射影されます。出力は以下の例のようになります。

VIN	市町村	Owners
"1N4AL11D 75C109151"	"Seattle"	{PrimaryOwner:{PersonId:""} ,SecondaryOwners:[]}
"1N4AL11D 75C109151"	"Seattle"	{PrimaryOwner:{PersonId:"29 4jJ3YUoH1IEEm8GSabOs"}, SecondaryOwners:[]}
"1N4AL11D 75C109151"	"Everett"	{PrimaryOwner:{PersonId:"7N mE8YLPbXc0IqesJy1rpR"}, SecondaryOwners:[]}
"1N4AL11D 75C109151"	"Everett"	{PrimaryOwner:{PersonId:"7N mE8YLPbXc0IqesJy1rpR"}, SecondaryOwners:[{PersonId: "5Ufgdlnj06gF5CWcOIu64s"}]}

> **Note**
>
> 履歴クエリでは、ドキュメントのリビジョンが順番に返されない場合があります。

出力を見て、変更が「ステップ 4: 台帳のドキュメントを変更する」で行った内容を反映していることを確認します。

6. 次に、各リビジョンのドキュメントメタデータを調べることができます。次のステートメントを入力し、[Run (実行)] を選択します。ここでも、必要に応じて、id 値を独自のドキュメント ID に置き換えてください。

```
SELECT VALUE h.metadata
FROM history(VehicleRegistration) AS h
WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
```

出力は以下の例のようになります。

versio	id	txTime	txId
0	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T19:20:360d-3Z	"FMoVdWuPxJg3k466Iz4i75"
1	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:40:199d-3Z	"KWByxe842Xw8DNHcvARPOt"
2	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:44:432d-3Z	"EKwDOJRwbHpFvmAyJ2Kdh9"
3	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:49:254d-3Z	"96EiZd7vCmJ6RAvOvTZ4YA"

これらのメタデータフィールドは、各項目がいつ、どのトランザクションによって変更されたかを示す詳細を提供します。このデータからは、以下を推測できます。

- ドキュメントはシステムによって割り当てられた id: ADR2LQq48kB9neZDupQrMm で個々に識別されます。これは Base62 でエンコードされた文字列で表される汎用一意識別子 (UUID) です。
- txTime は、ドキュメントの最初のリビジョン (バージョン 0) が 2019-05-23T19:20:360d-3Z に作成されたことを示しています。
- その後の各トランザクションにより、同じドキュメント id、増分されたバージョン番号、および更新された txId と txTime を含む新しいリビジョンが作成されます。

vehicle-registration 台帳のドキュメントリビジョンを暗号的に検証するには、「ステップ 6: 台帳のドキュメントを検証する」に進みます。

ステップ 6: 台帳のドキュメントを検証する

Amazon QLDB では、SHA-256 の暗号的ハッシュを使用して、台帳のジャーナルのドキュメントの整合性を効率的に検証できます。この例では、Alexis と Brent は VIN が 1N4AL11D75C109151 の車両をディーラーに下取りに出して、新しいモデルに買い替えることにしました。ディーラーは、登記所に問い合わせて車両の所有権を検証することで、このプロセスを開始します。

検証と暗号的ハッシュが QLDB でどのように機能するかについては、「Amazon QLDB でのデータ検証」を参照してください。

このステップでは、vehicle-registration 台帳内のドキュメントリビジョンを検証します。まず、ダイジェストをリクエストします。ダイジェストは出力ファイルとして返され、台帳の変更履歴全体の署名として機能します。次に、そのダイジェストに関連するリビジョンの証明をリクエストします。この証明を使用して、すべての検証チェックに合格すると、リビジョンの整合性が検証されます。

ダイジェストをリクエストするには

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. 台帳のリストで、「vehicle-registration」を選択します。
4. [Get digest (ダイジェストの取得)] を選択します。[Get digest (ダイジェストの取得)] ダイアログボックスに、以下のようなダイジェスト詳細が表示されます。

   - [Digest] (ダイジェスト) - リクエストしたダイジェストの SHA-256 ハッシュ値。

- [Digest tip address] (ダイジェストティップアドレス) - リクエストしたダイジェストの対象となっているジャーナル内の最新のブロック位置。アドレスには以下の 2 つのフィールドがあります。
  - `strandId` - ブロックを含むジャーナルストランドの一意の ID。
  - `sequenceNo` - ストランド内のブロックの位置を指定するインデックス番号。
- [Ledger] (台帳) - ダイジェストをリクエストした台長名。
- [Date] (日付) - ダイジェストをリクエストしたときのタイムスタンプ。

5. ダイジェスト情報を確認します。次に、[Save] (保存) を選択します。デフォルトのファイル名はそのままにしておくことも、新しい名前を入力することもできます。

   このステップでは、Amazon Ion 形式のコンテンツを含むプレーンテキストファイルが保存されます。このファイルには、`.ion.txt` のファイル名拡張子が付いており、前述のダイアログボックスに列挙されたすべてのダイジェスト情報が含まれています。以下は、ダイジェストファイルのコンテンツ例です。フィールドの順序はブラウザにより異なる場合があります。

   ```
   {
     "digest": "42zaJOfV8iGutVGNaIuzQWhD5Xb/5B9lScHnvxPXm9E=",
     "digestTipAddress": "{strandId:\"BlFTjlSXze9BIh1KOszcE3\",sequenceNo:73}",
     "ledger": "vehicle-registration",
     "date": "2019-04-17T16:57:26.749Z"
   }
   ```

6. このファイルを保存して後でアクセスすることができます。以下の手順では、このファイルを使用して、ドキュメントのリビジョンを検証します。

台帳ダイジェストを保存した後は、そのダイジェストと照合してドキュメントのリビジョンを検証するプロセスを開始できます。

> **Note**
>
> 検証の本稼働ユースケースでは、2 つのタスクを同時に行うのではなく、以前に保存したダイジェストを使用します。ベストプラクティスとして、後で検証するリビジョンがジャーナルに書き込まれ次第、ダイジェストをリクエストして保存します。

ドキュメントのリビジョンを検証するには

1. 最初に、検証するドキュメントリビジョンの id と blockAddress に関するクエリを台帳に対し実行します。これらのフィールドはドキュメントのメタデータに含まれており、コミット済みビューでクエリを実行できます。

   ドキュメント id は、システムによって割り当てられた一意の ID 文字列です。blockAddress は、リビジョンがコミットされたブロックの位置を指定する Ion 構造です。

   QLDB コンソールのナビゲーションペインで、[PartiQL エディタ] を選択します。

2. vehicle-registration 台帳を選択します。

3. クエリエディタウィンドウで、以下のステートメントを入力し、[Run (実行)] を選択します。

   ```
   SELECT r.metadata.id, r.blockAddress
   FROM _ql_committed_VehicleRegistration AS r
   WHERE r.data.VIN = '1N4AL11D75C109151'
   ```

4. クエリによって返された id と blockAddress の値をコピーし、保存します。id フィールドの二重引用符は必ず省略してください。Amazon Ion では、文字列データ型は二重引用符で区切られます。

5. これでドキュメントリビジョンが選択されたため、検証プロセスを開始できます。

   ナビゲーションペイン内で [Verification (検証)] を選択します。

6. [Verify document (ドキュメントの検証)] フォームの [Specify the document that you want to verify (検証するドキュメントの指定)] に、次の入力パラメータを入力します。

   - [Ledger] (台帳) — vehicle-registration を選択します。
   - [Block address] (ブロックアドレス) - ステップ 3 のクエリで返された blockAddress 値。
   - [Document ID] (ドキュメント ID) - ステップ 3 のクエリで返された id 値。

7. [Specify the digest to use for verification (検証に使用するダイジェストの指定)] で、[Choose digest (ダイジェストの選択)] を選択することで、以前に保存されたダイジェストを選択します。ファイルが有効であれば、コンソールにあるすべてのダイジェストフィールドにデータが自動入力されます。また、以下の値をダイジェストファイルから手動で直接コピーして貼り付けることもできます。

   - [Digest] (ダイジェスト) - ダイジェストファイルからの digest 値。

- [Digest tip address] (ダイジェストティップアドレス) - ダイジェストファイルからの `digestTipAddress` 値。

8. ドキュメントおよびダイジェストの入力パラメータを再確認し、[Verify (検証)] を選択します。

   コンソールが次の 2 つのステップを自動的に実行します。

   a. 指定されたドキュメントに関するプルーフを QLDB にリクエストします。
   b. QLDB によって返された証拠を使用してクライアント側 API を呼び出すことで、提供されたダイジェストと照合してドキュメントのリビジョンを検証します。

   コンソールの [Verification results (検証結果)] カードに、リクエストの結果が表示されます。詳細については、「検証結果」を参照してください。

9. 検証ロジックをテストするには、「ドキュメントのリビジョンを検証するには」のステップ 6～8 を繰り返しますが、[Digest] (ダイジェスト) 入力文字列の 1 文字を変更します。これにより、[Verify (検証)] リクエストは失敗し、適切なエラーメッセージが表示されます。

`vehicle-registration` 台帳を使用する必要がなくなった場合は、「ステップ 7 (オプション): リソースをクリーンアップする」に進みます。

ステップ 7 (オプション): リソースをクリーンアップする

`vehicle-registration` 台帳は引き続き使用できます。ただし、不要になった場合は、削除することをお勧めします。

台帳に対して削除保護が有効になっている場合は、QLDB API、AWS Command Line Interface (AWS CLI)、または QLDB コンソールを使用して台帳を削除する前に、まず無効にする必要があります。

台帳を削除するには

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. この台帳に対して削除保護が有効になっている場合は、まず無効にする必要があります。

   台帳のリストで、`vehicle-registration` を選択し、[Edit ledger] (台帳を編集) を選択します。

4. [Edit ledger] (台帳を編集) ページで、[Deletion protection] (削除保護) をオフにしてから、[Confirm changes] (変更の確認) を選択します。

5. 台帳のリストで、`vehicle-registration` をもう一度選択し、[Delete] (削除) を選択します。
6. 表示されたフィールドに「**`delete vehicle-registration`**」と入力して、削除を確定します。

QLDB で台帳を使用する方法の詳細については、「Amazon QLDB の開始方法: 次のステップ」を参照してください。

Amazon QLDB の開始方法: 次のステップ

Amazon QLDB の使用に関する詳細については、以下のトピックを参照してください。

- Amazon QLDB でのデータと履歴の使用
- Amazon QLDB ドライバーの開始方法
- Amazon QLDB 同時実行モデル
- Amazon QLDB からのジャーナルデータのエクスポート
- Amazon QLDB からのジャーナルデータのストリーミング
- Amazon QLDB でのデータ検証
- Amazon QLDB の PartiQL リファレンス

Amazon QLDB ドライバーの開始方法

この章には、QLDB ドライバーを使用した Amazon QLDB での開発について学ぶための実践的なチュートリアルが含まれています。ドライバーは AWS SDK 上に構築されており、QLDB API とのインタラクションをサポートしています。

QLDB セッションの抽象化

このドライバーは、トランザクションデータ API (QLDB セッション) 上に高レベルの抽象化レイヤーを提供します。これにより SendCommand API コールを管理して、元帳データに対する PartiQL ステートメントの実行プロセスが合理化されます。これらの API コールにはいくつかのパラメータが必要です。これにより、ドライバーは、セッション、トランザクション、エラー発生時の再試行ポリシーの管理といった処理を行えるようになります。ドライバーはパフォーマンスを最適化し、QLDB とやり取りに関するベストプラクティスを適用しています。

> **Note**
>
> 「Amazon QLDB API リファレンス」に示されているリソース管理 API オペレーションを操作するには、ドライバーの代わりに AWS SDK を直接使用します。管理 API は、台帳リソースの管理および非トランザクションデータオペレーション (エクスポート、ストリーミング、データ検証など) にのみ使用します。

Amazon Ion のサポート

さらに、トランザクションの実行時に Ion データ処理のサポートを提供するために、ドライバーは Amazon Ion ライブラリを使用します。これらのライブラリは Ion 値のハッシュの計算も処理します。QLDB では、データトランザクションリクエストの整合性をチェックするためにこれらの Ion ハッシュが必要です。

ドライバーの用語

このツールは、「ドライバー」と呼ばれます。これは、開発者に優しいインターフェースを提供する他のデータベースドライバーに匹敵するからです。これらのドライバーも、コマンドと関数の標準セットをサービスの低レベル API で必要とされる特定の呼び出しに変換するロジックをカプセル化します。

このドライバーは GitHub 上のオープンソースであり、次のプログラミング言語で使用できます。

- Java ドライバー

- .NET ドライバー
- Go ドライバー
- Node.js ドライバー
- Python ドライバー

サポートされているすべてのプログラミング言語の一般的なドライバー情報、および追加のチュートリアルについては、以下のトピックを参照してください。

- ドライバーによるセッション管理
- ドライバーに関する推奨事項
- ドライバーの再試行ポリシー
- 一般的なエラー
- サンプルアプリケーションチュートリアル
- Amazon Ion の操作
- PartiQL ステートメントの統計の取得

Java 用 Amazon QLDB ドライバー

台帳内のデータを使用するには、AWS が提供するドライバーを使用して Java アプリケーションから Amazon QLDB に接続できます。次のトピックでは、Java 用 QLDB ドライバーの使用を開始する方法について説明します。

トピック

- ドライバーに関するリソース
- 前提条件
- デフォルトの AWS の認証情報とリージョンを設定する
- インストール
- Java 用 Amazon QLDB ドライバー — クイックスタートチュートリアル
- Java 用 Amazon QLDB ドライバー — クックブックリファレンス

ドライバーに関するリソース

Java ドライバーでサポートされる機能の詳細については、以下のリソースを参照してください。

- API リファレンス: 2.x、1.x
- ドライバーのソースコード (GitHub)
- サンプルアプリケーションのソースコード (GitHub)
- レジャーローダーフレームワーク (GitHub)
- Amazon Ion コード例

前提条件

Java 用 QLDB ドライバーを開始する前に、次の操作を行う必要があります。

1. 「Amazon QLDB へのアクセス」にある AWS の設定手順に従います。これには以下が含まれます。
    1. AWS にサインアップする。
    2. QLDB の適切なアクセス許可を持つユーザーを作成します。
    3. 開発に必要なプログラムへのアクセスを提供します。
2. 以下をダウンロードおよびインストールして、Java 開発環境を設定します。
    1. Java SE 開発キット 8 (Amazon Corretto 8 など)。
    2. (オプション) 任意の Java 統合開発環境 (IDE) (Eclipse や IntelliJ など)。
3. 「デフォルトの AWS の認証情報とリージョンを設定する」に従って AWS SDK for Java 用の開発環境を設定します。

次に、すべてのチュートリアル用のサンプルアプリケーションをダウンロードするか、Java プロジェクトにのみドライバーをインストールして短いコード例を実行できます。

- 既存のプロジェクトに QLDB ドライバーと AWS SDK for Java をインストールするには、「インストール」に進みます。
- プロジェクトを設定し、台帳の基本的なデータトランザクションを示す短いコード例については、「クイックスタートチュートリアル」を参照してください。
- チュートリアルのサンプルアプリケーション全体のデータプレーンと管理 API の両方のオペレーションを実行する詳細な例については、「Java チュートリアル」を参照してください。

デフォルトの AWS の認証情報とリージョンを設定する

QLDB ドライバーとその基になる AWS SDK for Java では、アプリケーションのランタイムに AWS 認証情報を提供する必要があります。このガイドのコード例では、AWS 認証情報ファイルを使用していることを前提としています。詳細については、「AWS SDK for Java 2.x デベロッパーガイド」の「デフォルトの認証情報とリージョンを設定する」を参照してください。

これらの手順の一部として、デフォルトの QLDB エンドポイントを決定するためにデフォルトの AWS リージョン も設定する必要があります。コード例では、デフォルトの AWS リージョン で QLDB に接続します。QLDB が利用可能なリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

`~/.aws/credentials` という名前の AWS 認証情報ファイルの例を次に示します。ここで、チルダ文字 (~) はホームディレクトリを表します。

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

your_access_key_id と *your_secret_access_key* の値は、独自の AWS 認証情報の値に置き換えてください。

インストール

QLDB は、以下の Java ドライバーバージョンと AWS SDK の依存関係をサポートしています。

ドライバーのバージョン	AWS SDK	ステータス	最新リリース日
1.x	AWS SDK for Java 1.x	本番リリース	2020 年 3 月 20 日
2.x	AWS SDK for Java 2.x	本番リリース	2021 年 6 月 4 日

QLDB ドライバーをインストールするには、Gradle や Maven などの依存関係管理システムを使用することをお勧めします。例えば、以下のアーティファクトを Java プロジェクトの依存関係として追加します。

2.x

Gradle

build.gradle 設定ファイルにこの依存関係を追加します。

```
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
 '2.3.1'
}
```

Maven

pom.xml 設定ファイルにこの依存関係を追加します。

```
<dependencies>
  <dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>2.3.1</version>
  </dependency>
</dependencies>
```

このアーティファクトには、AWS SDK for Java 2.x コアモジュール、Amazon Ion ライブラリ、およびその他の必要な依存関係が自動的に含まれています。

1.x

Gradle

build.gradle 設定ファイルにこの依存関係を追加します。

```
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
 '1.1.0'
}
```

Maven

pom.xml 設定ファイルにこの依存関係を追加します。

```
<dependencies>
  <dependency>
```

```
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>1.1.0</version>
  </dependency>
</dependencies>
```

このアーティファクトには、AWS SDK for Java コアモジュール、Amazon Ion ライブラリ、およびその他の必要な依存関係が自動的に含まれています。

> **Important**
>
> Amazon Ion 名前空間 — アプリケーションに Amazon Ion クラスをインポートする場合は、名前空間 `com.amazon.ion` の下にあるパッケージを使用する必要があります。AWS SDK for Java は、名前空間 `software.amazon.ion` の別の Ion パッケージに依存しますが、これは QLDB ドライバーと互換性がないレガシーパッケージです。

台帳に対して基本的なデータトランザクションを実行する方法を示す短いコード例については、「クックブックリファレンス」を参照してください。

他のオプションのライブラリ

オプションで、プロジェクトに以下の便利なライブラリを追加することもできます。これらのアーティファクトは、Java チュートリアルサンプルアプリケーションで必須の依存関係にあります。

1. aws-java-sdk-qldb - AWS SDK for Java の QLDB モジュール。QLDB でサポートされる最小のバージョンは、`1.11.785` です。

   このモジュールを使用することで、アプリケーションは Amazon QLDB API リファレンス に表示されている管理 API オペレーションと直接やり取りできます。
2. jackson-dataformat-ion - Ion 用の FasterXML の Jackson データフォーマットモジュール。サンプルアプリケーションにはバージョン `2.10.0` 以降が必要です。

Gradle

`build.gradle` 設定ファイルにこれら依存関係を追加します。

```
dependencies {
    compile group: 'com.amazonaws', name: 'aws-java-sdk-qldb', version: '1.11.785'
```

```
    compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-
ion', version: '2.10.0'
}
```

Maven

pom.xml 設定ファイルにこれら依存関係を追加します。

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-qldb</artifactId>
    <version>1.11.785</version>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-ion</artifactId>
    <version>2.10.0</version>
  </dependency>
</dependencies>
```

Java 用 Amazon QLDB ドライバー — クイックスタートチュートリアル

このチュートリアルでは、Java 用 Amazon QLDB ドライバーの最新バージョンを使用して単純なアプリケーションを設定する方法について説明します。このガイドには、ドライバのインストール手順と、作成、読み取り、更新、削除 (CRUD) の基本的なオペレーションを実行する短いコード例が含まれています。これらのオペレーションを完全なサンプルアプリケーションで実行する詳細な例については、「Java チュートリアル」を参照してください。

トピック

前提条件

作業を始める前に、次の操作を実行してください。

1. Java ドライバー用の「前提条件」を完了します (まだ完了していない場合)。これには、AWS へのサインアップ、開発のためのプログラムによるアクセスの許可、Java 統合開発環境 (IDE) のインストールが含まれます。
2. `quick-start` という名前の台帳を作成します。

   台帳の作成方法については、「Amazon QLDB 台帳の基本的なオペレーション」、または「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」を参照してください。

ステップ 1: プロジェクトを設定する

まず、Java プロジェクトをセットアップします。このチュートリアルでは、Maven 依存関係管理システムを使用することをお勧めします。

> **Note**
>
> これらのセットアップ手順を自動化する機能を備えた IDE を使用する場合は、「ステップ 2: ドライバーを初期化する」までスキップできます。

1. アプリケーション用のフォルダを作成します。

   ```
   $ mkdir myproject
   $ cd myproject
   ```

2. Maven テンプレートからプロジェクトを初期化するには、次のコマンドを入力します。必要に応じて *`project-package`*、*`project-name`*、*`maven-template`* を自分の値で置換します。

   ```
   $ mvn archetype:generate
     -DgroupId=project-package \
     -DartifactId=project-name \
     -DarchetypeArtifactId=maven-template \
     -DinteractiveMode=false
   ```

   `maven-template` では、基本的な Maven テンプレート `maven-archetype-quickstart` を使用できます。

3. Java 用 QLDB ドライバーをプロジェクトの依存関係として追加するには、新しく作成した pom.xml ファイルに移動し、以下のアーティファクトを追加します。

```
<dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>2.3.1</version>
</dependency>
```

このアーティファクトには、AWS SDK for Java 2.x コアモジュール、Amazon Ion ライブラリ、およびその他の必要な依存関係が自動的に含まれています。pom.xml ファイルは以下のようになります。

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>software.amazon.qldb</groupId>
  <artifactId>qldb-quickstart</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>qldb-quickstart</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>software.amazon.qldb</groupId>
        <artifactId>amazon-qldb-driver-java</artifactId>
        <version>2.3.1</version>
    </dependency>
  </dependencies>
</project>
```

4. App.java ファイルを開きます。

次に、以下のステップのコード例を段階的に追加し、いくつかの基本的な CRUD オペレーションを試します。または、ステップバイステップのチュートリアルをスキップして、代わりに完全なアプリケーションを実行することもできます。

ステップ 2: ドライバーを初期化する

quick-start という名前の台帳に接続するドライバーのインスタンスを初期化します。以下のコードを App.java ファイルに追加します。

```
import java.util.*;
import com.amazon.ion.*;
import com.amazon.ion.system.*;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.*;

public final class App {
    public static IonSystem ionSys = IonSystemBuilder.standard().build();
    public static QldbDriver qldbDriver;

    public static void main(final String... args) {
        System.out.println("Initializing the driver");
        qldbDriver = QldbDriver.builder()
            .ledger("quick-start")
            .transactionRetryPolicy(RetryPolicy
                .builder()
                .maxRetries(3)
                .build())
            .sessionClientBuilder(QldbSessionClient.builder())
            .build();
    }
}
```

ステップ 3: テーブルとインデックスを作成する

以下のコード例は、CREATE TABLE および CREATE INDEX ステートメントの実行方法を示しています。

main メソッドで以下のコードを追加して、People という名前のテーブルと、そのテーブルの lastName フィールドのインデックスを作成します。クエリのパフォーマンスを最適化し、オプティミスティック同時実行制御 (OCC) 競合例外を制限するために、インデックスが必要です。

```
// Create a table and an index in the same transaction
qldbDriver.execute(txn -> {
    System.out.println("Creating a table and an index");
    txn.execute("CREATE TABLE People");
    txn.execute("CREATE INDEX ON People(lastName)");
});
```

ステップ 4: ドキュメントを挿入する

次のコード例は、INSERT ステートメントの実行方法を示しています。QLDB は PartiQL クエリ言語 (SQL 互換) と Amazon Ion データ形式 (JSON のスーパーセット) をサポートします。

People テーブルにドキュメントを挿入する以下のコードを追加します。

```
// Insert a document
qldbDriver.execute(txn -> {
    System.out.println("Inserting a document");
    IonStruct person = ionSys.newEmptyStruct();
    person.put("firstName").newString("John");
    person.put("lastName").newString("Doe");
    person.put("age").newInt(32);
    txn.execute("INSERT INTO People ?", person);
});
```

この例では、疑問符 (?) を変数プレースホルダーとして使用して、ドキュメント情報をステートメントに渡します。プレースホルダーを使用する場合、IonValue 型の値を渡す必要があります。

> **Tip**
>
> 1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 IonList のパラメータを (明示的に IonValue にキャストして) ステートメントに渡すことができます。
>
> ```
> // people is an IonList explicitly cast as an IonValue
> txn.execute("INSERT INTO People ?", (IonValue) people);
> ```
>
> IonList を渡すときは、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ステップ 5: ドキュメントにクエリを実行する

次のコード例は、SELECT ステートメントの実行方法を示しています。

People テーブルからドキュメントをクエリする以下のコードを追加します。

```
// Query the document
qldbDriver.execute(txn -> {
    System.out.println("Querying the table");
    Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
            ionSys.newString("Doe"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("firstName")); // prints John
    System.out.println(person.get("lastName")); // prints Doe
    System.out.println(person.get("age")); // prints 32
});
```

ステップ 6: ドキュメントを更新する

次のコード例は、UPDATE ステートメントの実行方法を示しています。

1. age を 42 に更新して People テーブルのドキュメントを更新する以下のコードを追加します。

   ```
   // Update the document
   qldbDriver.execute(txn -> {
       System.out.println("Updating the document");
       final List<IonValue> parameters = new ArrayList<>();
       parameters.add(ionSys.newInt(42));
       parameters.add(ionSys.newString("Doe"));
       txn.execute("UPDATE People SET age = ? WHERE lastName = ?", parameters);
   });
   ```

2. ドキュメントを再度クエリして、更新された値を確認します。

   ```
   // Query the updated document
   qldbDriver.execute(txn -> {
       System.out.println("Querying the table for the updated document");
       Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
               ionSys.newString("Doe"));
       IonStruct person = (IonStruct) result.iterator().next();
       System.out.println(person.get("firstName")); // prints John
       System.out.println(person.get("lastName")); // prints Doe
       System.out.println(person.get("age")); // prints 32
   ```

```
});
```

3. Maven または IDE を使用して、App.java ファイルを実行します。

完全なアプリケーションの実行

以下のコード例は、App.java アプリケーションの完全なバージョンです。上のステップを個別に実行する代わりに、このコード例を最初から最後までコピーして実行することもできます。このアプリケーションは、quick-start という名前の台帳に対するいくつかの基本的な CRUD オペレーションを実行します。

> **Note**
>
> このコードを実行する前に、quick-start 台帳に People という名前のアクティブなテーブルが存在しないことを確認してください。
> 最初の行で、*project-package* を ステップ 1: プロジェクトを設定する で Maven コマンドに使用した groupId 値に置換します。

```
package project-package;

import java.util.*;
import com.amazon.ion.*;
import com.amazon.ion.system.*;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.*;

public class App {
    public static IonSystem ionSys = IonSystemBuilder.standard().build();
    public static QldbDriver qldbDriver;

    public static void main(final String... args) {
        System.out.println("Initializing the driver");
        qldbDriver = QldbDriver.builder()
            .ledger("quick-start")
            .transactionRetryPolicy(RetryPolicy
                .builder()
                .maxRetries(3)
                .build())
            .sessionClientBuilder(QldbSessionClient.builder())
            .build();
```

```
        // Create a table and an index in the same transaction
        qldbDriver.execute(txn -> {
            System.out.println("Creating a table and an index");
            txn.execute("CREATE TABLE People");
            txn.execute("CREATE INDEX ON People(lastName)");
        });

        // Insert a document
        qldbDriver.execute(txn -> {
            System.out.println("Inserting a document");
            IonStruct person = ionSys.newEmptyStruct();
            person.put("firstName").newString("John");
            person.put("lastName").newString("Doe");
            person.put("age").newInt(32);
            txn.execute("INSERT INTO People ?", person);
        });

        // Query the document
        qldbDriver.execute(txn -> {
            System.out.println("Querying the table");
            Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
                    ionSys.newString("Doe"));
            IonStruct person = (IonStruct) result.iterator().next();
            System.out.println(person.get("firstName")); // prints John
            System.out.println(person.get("lastName")); // prints Doe
            System.out.println(person.get("age")); // prints 32
        });

        // Update the document
        qldbDriver.execute(txn -> {
            System.out.println("Updating the document");
            final List<IonValue> parameters = new ArrayList<>();
            parameters.add(ionSys.newInt(42));
            parameters.add(ionSys.newString("Doe"));
            txn.execute("UPDATE People SET age = ? WHERE lastName = ?", parameters);
        });

        // Query the updated document
        qldbDriver.execute(txn -> {
            System.out.println("Querying the table for the updated document");
            Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
                    ionSys.newString("Doe"));
            IonStruct person = (IonStruct) result.iterator().next();
```

```
            System.out.println(person.get("firstName")); // prints John
            System.out.println(person.get("lastName")); // prints Doe
            System.out.println(person.get("age")); // prints 42
        });
    }
}
```

Maven または IDE を使用して、`App.java` ファイルを実行します。

Java 用 Amazon QLDB ドライバー — クックブックリファレンス

このリファレンスガイドでは、Java 用 Amazon QLDB ドライバーの一般的なユースケースについて説明します。Java コード例を提供し、ドライバーを使用して作成、読み取り、更新、および削除 (CRUD) の基本的なオペレーションを実行する方法を説明しています。Amazon Ion データを処理するためのコード例も含まれています。さらに、このガイドでは、トランザクションをべき等にし、一意性の制約を実装するためのベストプラクティスを取り上げています。

> **Note**
>
> 該当する場合、一部のユースケースでは、サポートされているメジャーバージョンの Java 用 QLDB ドライバーに対して異なるコード例があります。

目次

ドライバーのインポート

次のコード例では、ドライバー、QLDB セッションクライアント、Amazon Ion パッケージ、およびその他の関連する依存関係をインポートします。

2.x

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;

import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;
```

1.x

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;

import com.amazonaws.services.qldbsession.AmazonQLDBSessionClientBuilder;
import software.amazon.qldb.PooledQldbDriver;
import software.amazon.qldb.Result;
```

ドライバーのインスタンス化

次のコード例では、指定された台帳名に接続するドライバーインスタンスを作成し、指定された再試行ロジックをカスタム再試行制限で使用します。

> **Note**
>
> この例では、Amazon Ion システムオブジェクト (IonSystem) もインスタンス化します。このリファレンスでいくつかのデータオペレーションを実行するときに Ion データを処理するには、このオブジェクトが必要です。詳細については、「Amazon Ion の操作」を参照してください。

2.x

```
QldbDriver qldbDriver = QldbDriver.builder()
    .ledger("vehicle-registration")
    .transactionRetryPolicy(RetryPolicy
        .builder()
        .maxRetries(3)
        .build())
    .sessionClientBuilder(QldbSessionClient.builder())
    .build();

IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

1.x

```
PooledQldbDriver qldbDriver = PooledQldbDriver.builder()
    .withLedger("vehicle-registration")
    .withRetryLimit(3)
    .withSessionClientBuilder(AmazonQLDBSessionClientBuilder.standard())
    .build();

IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

CRUD オペレーション

QLDB は作成、読み取り、更新、および削除 (CRUD) オペレーションをトランザクションの一部として実行します。

> **Warning**
>
> ベストプラクティスとして、書き込みトランザクションを厳密にべき等にしてください。

トランザクションをべき等にする

再試行の場合に予期しない結果を避けるために、書き込みトランザクションをべき等にすることをお勧めします。トランザクションは、複数回実行して毎回同じ結果を生成できる場合、idempotent です。

例えば、Person という名前のテーブルにドキュメントを挿入するトランザクションについて考えてみます。トランザクションは、まずドキュメントがテーブル内に既に存在するかどうかをチェックする必要があります。このチェックを行わないと、テーブルに重複するドキュメントができる可能性があります。

QLDB がサーバー側でトランザクションを正常にコミットできるが、レスポンスを待機中にクライアントはタイムアウトするとします。トランザクションがべき等でない場合、再試行時に同じドキュメントが複数回挿入される可能性があります。

インデックスを使用してテーブル全体のスキャンを回避する

インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`) こともお勧めします。このインデックス付きのルックアップがない場合、QLDB はテーブルスキャンを実行する必要があり、トランザクションタイムアウトやオプティミスティック同時実行制御 (OCC) 競合が発生する可能性があります。

OCC の詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

暗黙的に作成されたトランザクション

QldbDriver.execute メソッドは、Executor のインスタンスを受け取る Lambda 関数を受け入れます。これを使用してステートメントを実行できます。Executor インスタンスは、暗黙的に作成されたトランザクションをラップします。

Executor.execute メソッドを使用して、Lambda 関数内でステートメントを実行できます。ドライバーは、Lambda 関数が戻ったときに暗黙的にトランザクションをコミットします。

次のセクションでは、基本的な CRUD オペレーションの実行、カスタム再試行ロジックの指定、一意性制約の実装方法について説明します。

> **Note**
>
> 該当する場合、これらのセクションでは、組み込みの Ion ライブラリと Jackson Ion マッパーライブラリの両方を使用して Amazon Ion データを処理するコード例を示します。詳細については、「Amazon Ion の操作」を参照してください。

目次

テーブルの作成

```
qldbDriver.execute(txn -> {
    txn.execute("CREATE TABLE Person");
});
```

インデックスの作成

```
qldbDriver.execute(txn -> {
    txn.execute("CREATE INDEX ON Person(GovId)");
});
```

ドキュメントの読み取り

```
// Assumes that Person table has documents as follows:
// { GovId: "TOYENC486FH", FirstName: "Brent" }
```

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

クエリパラメータの使用

次のコード例では、Ion 型のクエリパラメータを使用します。

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

次のコード例では、複数のクエリパラメータを使用します。

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ? AND FirstName
 = ?",
            SYSTEM.newString("TOYENC486FH"),
            SYSTEM.newString("Brent"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

次のコード例では、クエリパラメータのリストを使用します。

```
qldbDriver.execute(txn -> {
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(SYSTEM.newString("TOYENC486FH"));
    parameters.add(SYSTEM.newString("ROEE1"));
    parameters.add(SYSTEM.newString("YH844"));
    Result result = txn.execute("SELECT * FROM Person WHERE GovId IN (?,?,?)",
 parameters);
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
```

```
});
```

Jackson マッパーの使用

```
// Assumes that Person table has documents as follows:
// {GovId: "TOYENC486FH", FirstName: "Brent" }

qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'");
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

クエリパラメータの使用

次のコード例では、Ion 型のクエリパラメータを使用します。

```
qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
                MAPPER.writeValueAsIonValue("TOYENC486FH"));
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

次のコード例では、複数のクエリパラメータを使用します。

```
qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ? AND FirstName
 = ?",
                MAPPER.writeValueAsIonValue("TOYENC486FH"),
                MAPPER.writeValueAsIonValue("Brent"));
```

```
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

次のコード例では、クエリパラメータのリストを使用します。

```
qldbDriver.execute(txn -> {
    try {
        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(MAPPER.writeValueAsIonValue("TOYENC486FH"));
        parameters.add(MAPPER.writeValueAsIonValue("ROEE1"));
        parameters.add(MAPPER.writeValueAsIonValue("YH844"));
        Result result = txn.execute("SELECT * FROM Person WHERE GovId IN (?,?,?)",
 parameters);
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

> **Note**
>
> インデックス付きルックアップなしでクエリを実行すると、完全なテーブルスキャンが呼び出されます。この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、クエリのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの挿入

次のコード例では、Ion データ型を挿入します。

```
qldbDriver.execute(txn -> {
    // Check if a document with GovId:TOYENC486FH exists
```

```
    // This is critical to make this transaction idempotent
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
    // Check if there is a result
    if (!result.iterator().hasNext()) {
        IonStruct person = SYSTEM.newEmptyStruct();
        person.put("GovId").newString("TOYENC486FH");
        person.put("FirstName").newString("Brent");
        // Insert the document
        txn.execute("INSERT INTO Person ?", person);
    }
});
```

Jackson マッパーの使用

次のコード例では、Ion データ型を挿入します。

```
qldbDriver.execute(txn -> {
    try {
        // Check if a document with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
                MAPPER.writeValueAsIonValue("TOYENC486FH"));
        // Check if there is a result
        if (!result.iterator().hasNext()) {
            // Insert the document
            txn.execute("INSERT INTO Person ?",
                    MAPPER.writeValueAsIonValue(new Person("Brent", "TOYENC486FH")));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

このトランザクションは、ドキュメントを Person テーブルに挿入します。挿入する前に、まずドキュメントがテーブル内に既に存在しているかどうかをチェックします。このチェックにより、トランザクションは本質的にべき等になります。このトランザクションを複数回実行しても、意図しない副作用は発生しません。

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

1 つのステートメントで複数のドキュメントの挿入

1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 IonList のパラメータを (明示的に IonValue にキャストして) ステートメントに渡すことができます。

```
// people is an IonList explicitly cast as an IonValue
txn.execute("INSERT INTO People ?", (IonValue) people);
```

IonList を渡すときは、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

明示的なキャストが必要なのはなぜですか？

TransactionExecutor.execute メソッドはオーバーロードされています。これは、可変数の IonValue 引数 (varargs)、または単一の List<IonValue> 引数を受け入れます。ion-java では、IonList が List<IonValue> として実装されます。

Java では、オーバーロードされたメソッドを呼び出すと、デフォルトで最も具体的なメソッド実装が使用されます。この場合、IonList パラメータを渡すと、List<IonValue> を使用するメソッドがデフォルトで使用されます。このメソッド実装は、呼び出されるとリストの IonValue 要素を個別の値として渡します。したがって、代わりに varargs メソッドを呼び出すには、IonList パラメータを IonValue に明示的にキャストする必要があります。

ドキュメントの更新

```
qldbDriver.execute(txn -> {
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(SYSTEM.newString("John"));
    parameters.add(SYSTEM.newString("TOYENC486FH"));
    txn.execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", parameters);
```

```
});
```

Jackson マッパーの使用

```
qldbDriver.execute(txn -> {
    try {
        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(MAPPER.writeValueAsIonValue("John"));
        parameters.add(MAPPER.writeValueAsIonValue("TOYENC486FH"));
        txn.execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", parameters);
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの削除

```
qldbDriver.execute(txn -> {
    txn.execute("DELETE FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
});
```

Jackson マッパーの使用

```
qldbDriver.execute(txn -> {
    try {
        txn.execute("DELETE FROM Person WHERE GovId = ?",
                MAPPER.writeValueAsIonValue("TOYENC486FH"));
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

トランザクションで複数のステートメントの実行

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
 not.
public static boolean InsureCar(QldbDriver qldbDriver, final String vin) {
    final IonSystem ionSystem = IonSystemBuilder.standard().build();
    final IonString ionVin = ionSystem.newString(vin);

    return qldbDriver.execute(txn -> {
        Result result = txn.execute(
                "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
                ionVin);
        if (!result.isEmpty()) {
            txn.execute("UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

再試行ロジック

ドライバーの execute メソッドには、再試行可能な例外 (タイムアウトや OCC 競合など) が発生した場合にトランザクションを再試行する組み込みの再試行メカニズムがあります。

2.x

再試行の最大数とバックオフ戦略を設定できます。

デフォルトの再試行制限は 4 であり、デフォルトのバックオフ戦略は DefaultQldbTransactionBackoffStrategy です。再試行設定は、RetryPolicy のインスタンスを使用して、ドライバーインスタンスごとおよびトランザクションごとに設定できます。

次のコード例では、ドライバーのインスタンスのカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。

```
public void retry() {
    QldbDriver qldbDriver = QldbDriver.builder()
            .ledger("vehicle-registration")
            .transactionRetryPolicy(RetryPolicy.builder()
                    .maxRetries(2)
                    .backoffStrategy(new CustomBackOffStrategy()).build())
            .sessionClientBuilder(QldbSessionClient.builder())
            .build();
}

private class CustomBackOffStrategy implements BackoffStrategy {

    @Override
    public Duration calculateDelay(RetryPolicyContext retryPolicyContext) {
        return Duration.ofMillis(1000 * retryPolicyContext.retriesAttempted());
    }
}
```

次のコード例では、特定のトランザクションのカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。execute のこの設定は、ドライバーインスタンスに設定された再試行ロジックを上書きします。

```
public void retry() {
    Result result = qldbDriver.execute(txn -> { txn.execute("SELECT * FROM Person
 WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH")); },
            RetryPolicy.builder()
                    .maxRetries(2)
                    .backoffStrategy(new CustomBackOffStrategy())
                    .build());
}

private class CustomBackOffStrategy implements BackoffStrategy {

    // Configuring a custom backoff which increases delay by 1s for each attempt.
    @Override
    public Duration calculateDelay(RetryPolicyContext retryPolicyContext) {
        return Duration.ofMillis(1000 * retryPolicyContext.retriesAttempted());
    }
```

```
}
```

1.x

再試行の最大数は設定可能です。再試行制限を設定するには、PooledQldbDriver の初期化時に retryLimit プロパティを設定します。

デフォルトの再試行制限回数は 4 です。

一意性制約の実装

QLDB はユニークインデックスをサポートしていませんが、この動作をアプリケーションに実装できます。

Person テーブル内の GovId フィールドに対して一意性制約を実装するとします。これを行うには、以下を実行するトランザクションを記述します。

1. テーブルに指定された GovId を持つ既存のドキュメントがないことをアサートします。
2. アサーションに合格した場合は、ドキュメントを挿入します。

競合するトランザクションが同時にアサーションに合格すると、一方のトランザクションだけが正常にコミットされます。もう一方のトランザクションは OCC 競合例外で失敗します。

次のコード例は、この一意性制約ロジックを実装する方法を示しています。

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
    // Check if there is a result
    if (!result.iterator().hasNext()) {
        IonStruct person = SYSTEM.newEmptyStruct();
        person.put("GovId").newString("TOYENC486FH");
        person.put("FirstName").newString("Brent");
        // Insert the document
        txn.execute("INSERT INTO Person ?", person);
    }
});
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

Amazon Ion の操作

QLDB で Amazon Ion データを処理するには、複数の方法があります。Ion ライブラリから組み込みメソッドを使用するには、必要に応じて、ドキュメントを柔軟に作成および変更します。または、FasterXML の Ion 用 Jackson データフォーマットモジュールを使用して Ion ドキュメントを Plain Old Java Object (POJO) モデルにマップすることもできます。

次のセクションでは、両方の手法を使用して Ion データを処理するコード例について説明します。

目次

Ion パッケージのインポート

アーティファクト ion-java を Java プロジェクトの依存関係として追加します。

Gradle

```
dependencies {
    compile group: 'com.amazon.ion', name: 'ion-java', version: '1.6.1'
}
```

Maven

```
<dependencies>
  <dependency>
    <groupId>com.amazon.ion</groupId>
```

```
    <artifactId>ion-java</artifactId>
    <version>1.6.1</version>
  </dependency>
</dependencies>
```

次の Ion パッケージをインポートします。

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
```

Jackson マッパーの使用

アーティファクト jackson-dataformat-ion を Java プロジェクトの依存関係として追加します。QLDB にはバージョン `2.10.0` 以降が必要です。

Gradle

```
dependencies {
    compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-
ion', version: '2.10.0'
}
```

Maven

```
<dependencies>
  <dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-ion</artifactId>
    <version>2.10.0</version>
  </dependency>
</dependencies>
```

次の Ion パッケージをインポートします。

```
import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonSystemBuilder;
```

```
import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;
```

Ion の初期化

```
IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

Jackson マッパーの使用

```
IonObjectMapper MAPPER = new IonValueMapper(IonSystemBuilder.standard().build());
```

Ion オブジェクトの作成

次のコード例では、IonStruct インターフェイスとその組み込みメソッドを使用して Ion オブジェクトを作成します。

```
IonStruct ionStruct = SYSTEM.newEmptyStruct();

ionStruct.put("GovId").newString("TOYENC486FH");
ionStruct.put("FirstName").newString("Brent");

System.out.println(ionStruct.toPrettyString()); // prints a nicely formatted copy of
 ionStruct
```

Jackson マッパーの使用

以下のような Person という名前の JSON にマップされたモデルクラスがあるとします。

```
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public static class Person {
    private final String firstName;
    private final String govId;

    @JsonCreator
    public Person(@JsonProperty("FirstName") final String firstName,
                  @JsonProperty("GovId") final String govId) {
        this.firstName = firstName;
        this.govId = govId;
```

```
    }

    @JsonProperty("FirstName")
    public String getFirstName() {
        return firstName;
    }

    @JsonProperty("GovId")
    public String getGovId() {
        return govId;
    }
}
```

次のコード例では、Person のインスタンスから IonStruct オブジェクトを作成します。

```
IonStruct ionStruct = (IonStruct) MAPPER.writeValueAsIonValue(new Person("Brent",
 "TOYENC486FH"));
```

Ion オブジェクトの作成

次のコード例では、ionStruct インスタンスの各フィールドを出力します。

```
// ionStruct is an instance of IonStruct
System.out.println(ionStruct.get("GovId")); // prints TOYENC486FH
System.out.println(ionStruct.get("FirstName")); // prints Brent
```

Jackson マッパーの使用

次のコード例では、IonStruct オブジェクトを読み取り、Person のインスタンスにマップします。

```
// ionStruct is an instance of IonStruct
IonReader reader = IonReaderBuilder.standard().build(ionStruct);
Person person = MAPPER.readValue(reader, Person.class);
System.out.println(person.getFirstName()); // prints Brent
System.out.println(person.getGovId()); // prints TOYENC486FH
```

Ion の操作方法の詳細については、GitHub で Amazon Ion のドキュメントを参照してください。QLDB で Ion を操作するコード例については、「Amazon QLDB で Amazon Ion のデータ型を操作する」を参照してください。

.NET 用 Amazon QLDB ドライバー

台帳内のデータを操作するには、AWS が提供するドライバーを使用して Microsoft .NET アプリケーションから Amazon QLDB に接続します。このドライバーは、.NET Standard 2.0 を対象としています。具体的には、.NET Core (LTS) 2.1+ および .NET Framework 4.5.2+ をサポートしています。互換性の詳細については、Microsoft Docs サイトの「.NET Standard」を参照してください。

Amazon Ion 型とネイティブ C# 型を手動で変換する必要性を完全に回避するために、Ion オブジェクトマッパーを使用することを強くお勧めします。

次のトピックでは、.NET 用 QLDB ドライバーの使用を開始する方法について説明します。

トピック

ドライバーに関するリソース

.NET ドライバーでサポートされている機能の詳細については、以下のリソースを参照してください。

- API リファレンス
- ドライバーのソースコード (GitHub)
- サンプルアプリケーションのソースコード (GitHub)
- Amazon Ion クックブック
- イオンオブジェクトマッパー (GitHub)

前提条件

.NET 用 QLDB ドライバーの使用を開始する前に、次のことを行う必要があります。

1. 「Amazon QLDB へのアクセス」にある AWS の設定手順に従います。これには以下が含まれます。

   1. AWS にサインアップする。
   2. QLDB の適切なアクセス許可を持つユーザーを作成します。
   3. 開発に必要なプログラムへのアクセスを提供します。
2. Microsoft .NET ダウンロードサイトから .NET Core SDK バージョン 2.1 以降をダウンロードしてインストールします。
3. (オプション) Visual Studio、Visual Studio for Mac、Visual Studio Code など、選択した統合開発環境 (IDE) をインストールします。これらの IDE は Microsoft Visual Studio サイトからダウンロードできます。
4. AWS SDK for .NET の開発環境を設定します。
   1. AWS 認証情報をセットアップします。共有認証情報ファイルを作成することをお勧めします。

      手順については、「AWS SDK for .NET デベロッパーガイド」の「Configuring AWS credentials using a credentials file」を参照してください。
   2. デフォルトの AWS リージョン を設定します。方法については、「AWS リージョン selection」を参照してください。

      利用可能なリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

次に、基本的なサンプルアプリケーションを設定し、短いコード例を実行するか、既存の .NET プロジェクトにドライバーをインストールできます。

- 既存のプロジェクトに QLDB ドライバーと AWS SDK for .NET をインストールするには、「インストール」に進みます。
- プロジェクトを設定し、台帳の基本的なデータトランザクションを示す短いコード例については、「クイックスタートチュートリアル」を参照してください。

インストール

NuGet パッケージマネージャーを使用して、.NET 用 QLDB ドライバーをインストールします。プロジェクトの依存関係を追加するには、Visual Studio または任意の IDE を使用することをお勧めします。ドライバーのパッケージ名は、Amazon.QLDB.Driver です。

例えば、Visual Studio では、[Tools] (ツール) メニューの [NuGet Package Manager Console] (NuGet パッケージマネージャーコンソール) を開きます。次に、PM> プロンプトで以下のコマンドを入力します。

```
PM> Install-Package Amazon.QLDB.Driver
```

ドライバーをインストールすると、AWS SDK for .NET や Amazon Ion パッケージなどの依存関係もインストールされます。

Ion オブジェクトマッパーをインストールする

.NET 用 QLDB ドライバーのバージョン 1.3.0 では、Amazon Ion を使用しなくてもネイティブ C# データ型を受け入れたり返したりできるようになりました。この機能を使用するには、次のパッケージをプロジェクトに追加してください。

- Amazon.QLDB.Driver.Serialization – Ion 値を C# のプレーンオールド CLR オブジェクト (POCO) にマッピングしたり、その逆を行ったりできるライブラリです。この Ion オブジェクトマッパーを使用すると、アプリケーションが Ion を操作しなくてもネイティブ C# データ型と直接やり取りできるようになります。このライブラリの使用方法に関する簡単なガイドについては、GitHub リポジトリ `awslabs/amazon-qldb-driver-dotnet` の Serialization.md ファイルを参照してください。

次のコマンドを入力して、このパッケージをインストールします。

```
PM> Install-Package Amazon.QLDB.Driver.Serialization
```

台帳に対して基本的なデータトランザクションを実行する方法を示す短いコード例については、「クックブックリファレンス」を参照してください。

.NET 用 Amazon QLDB ドライバー — クイックスタートチュートリアル

このチュートリアルでは、.NET 用 Amazon QLDB ドライバーを使用して単純なアプリケーションを設定する方法について説明します。このガイドには、ドライバのインストール手順と、作成、読み取り、更新、削除 (CRUD) の基本的なオペレーションを実行する短いコード例が含まれています。

トピック

- ステップ 2: ドライバーを初期化する
- ステップ 3: テーブルとインデックスを作成する
- ステップ 4: ドキュメントを挿入する
- ステップ 5: ドキュメントにクエリを実行する
- ステップ 6: ドキュメントを更新する
- 完全なアプリケーションの実行

前提条件

作業を始める前に、次の操作を実行してください。

1. .NET ドライバー用の「前提条件」を完了します (まだ完了していない場合)。これには、AWS へのサインアップ、開発のためのプログラムによるアクセスの許可、.NET Core SDK のインストールが含まれます。
2. `quick-start` という名前の台帳を作成します。

   台帳の作成方法については、「Amazon QLDB 台帳の基本的なオペレーション」、または「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」を参照してください。

ステップ 1: プロジェクトを設定する

まず、.NET プロジェクトをセットアップします。

1. テンプレートアプリケーションを作成して実行するには、bash、PowerShell、コマンドプロンプトなどのターミナルで、次の `dotnet` コマンドを入力します。

   ```
   $ dotnet new console --output Amazon.QLDB.QuickStartGuide
   $ dotnet run --project Amazon.QLDB.QuickStartGuide
   ```

   このテンプレートによって、`Amazon.QLDB.QuickStartGuide` という名前のフォルダが作成されます。そのフォルダに、同じ名前でプロジェクトが作成され、`Program.cs` という名前でファイルが作成されます。プログラムには、出力 `Hello World!` を表示するコードが含まれています。
2. NuGet パッケージマネージャーを使用して、.NET 用 QLDB ドライバーをインストールします。プロジェクトに依存関係を追加するには、Visual Studio または任意の IDE を使用することをお勧めします。ドライバーのパッケージ名は、Amazon.QLDB.Driver です。

- 例えば、Visual Studio では、[Tools] (ツール) メニューの [NuGet Package Manager Console] (NuGet パッケージマネージャーコンソール) を開きます。次に、PM> プロンプトで以下のコマンドを入力します。

```
PM> Install-Package Amazon.QLDB.Driver
```

- または、ターミナルで以下のコマンドを入力できます。

```
$ cd Amazon.QLDB.QuickStartGuide
$ dotnet add package Amazon.QLDB.Driver
```

ドライバーをインストールすると、AWS SDK for .NET や Amazon Ion ライブラリなどの依存関係もインストールされます。

3. ドライバーのシリアル化ライブラリをインストールします。

```
PM> Install-Package Amazon.QLDB.Driver.Serialization
```

4. Program.cs ファイルを開きます。

次に、以下のステップのコード例を段階的に追加し、いくつかの基本的な CRUD オペレーションを試します。または、ステップバイステップのチュートリアルをスキップして、代わりに完全なアプリケーションを実行することもできます。

> **Note**
>
> - 同期 API と非同期 API の選択 – このドライバーでは、同期 API と非同期 API を利用できます。ブロックせずに複数のリクエストを処理する需要の高いアプリケーションの場合、パフォーマンス向上のために非同期 API の使用をお勧めします。同期 API を利用すると、同期的に記述した既存のコードベースの利便性を高めることができます。
>
>   このチュートリアルでは、同期コードと非同期コードの両方の例を示します。API の詳細については、API ドキュメントに記載された「IQldbDriver」と「IAsyncQldbDriver」の各インターフェイスを参照してください。
>
> - Amazon Ion データの処理 – このチュートリアルでは、デフォルトで Ion オブジェクトマッパーを使用して Amazon Ion データを処理するコード例を紹介します。QLDB は、.NET ドライバーのバージョン 1.3.0 で Ion オブジェクトマッパーを導入しました。こ

のチュートリアルでは、該当する場合、標準の Ion ライブラリを代替として使用するコード例も提供しています。詳細については、「Amazon Ion の操作」を参照してください。

ステップ 2: ドライバーを初期化する

quick-start という名前の台帳に接続するドライバーのインスタンスを初期化します。以下のコードを Program.cs ファイルに追加します。

Async

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();
        }
    }
```

```
}
```

Sync

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();
        }
    }
}
```

Ion ライブラリの使用

Async

```
using System;
```

```
using System.Threading.Tasks;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();
        }
    }
}
```

Sync

```
using System;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();
        }
    }
```

```
}
```

ステップ 3: テーブルとインデックスを作成する

このチュートリアルの残りの部分 (ステップ 6 まで) では、前のコード例に次のコード例を追加する必要があります。

このステップの次のコードは、CREATE TABLE および CREATE INDEX ステートメントの実行方法を示しています。これにより、Person という名前のテーブルと、そのテーブルの firstName フィールドのインデックスが作成されます。クエリのパフォーマンスを最適化し、オプティミスティック同時実行制御 (OCC) 競合例外を制限するために、インデックスが必要です。

Async

```
Console.WriteLine("Creating the table and index");

// Creates the table and the index in the same transaction.
// Note: Any code within the lambda can potentially execute multiple times due to
 retries.
// For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
await driver.Execute(async txn =>
{
    await txn.Execute("CREATE TABLE Person");
    await txn.Execute("CREATE INDEX ON Person(firstName)");
});
```

Sync

```
Console.WriteLine("Creating the tables and index");

// Creates the table and the index in the same transaction.
// Note: Any code within the lambda can potentially execute multiple times due to
 retries.
// For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
driver.Execute(txn =>
{
    txn.Execute("CREATE TABLE Person");
    txn.Execute("CREATE INDEX ON Person(firstName)");
});
```

ステップ 4: ドキュメントを挿入する

次のコード例は、INSERT ステートメントの実行方法を示しています。QLDB は PartiQL クエリ言語 (SQL 互換) と Amazon Ion データ形式 (JSON のスーパーセット) をサポートします。

Person テーブルにドキュメントを挿入する以下のコードを追加します。

Async

```
Console.WriteLine("Inserting a document");

Person myPerson = new Person {
    FirstName = "John",
    LastName = "Doe",
    Age = 32
};

await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?", myPerson);
    await txn.Execute(myQuery);
});
```

Sync

```
Console.WriteLine("Inserting a document");

Person myPerson = new Person {
    FirstName = "John",
    LastName = "Doe",
    Age = 32
};

driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?", myPerson);
    txn.Execute(myQuery);
});
```

Ion ライブラリの使用

Async

```
Console.WriteLine("Inserting a document");

// This is one way of creating Ion values. We can also use an IonLoader.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/
driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
ionPerson.SetField("age", valueFactory.NewInt(32));

await driver.Execute(async txn =>
{
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Sync

```
Console.WriteLine("Inserting a document");

// This is one way of creating Ion values, we can also use an IonLoader.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/
driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
ionPerson.SetField("age", valueFactory.NewInt(32));

driver.Execute(txn =>
{
    txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

> **Tip**
>
> 次のように Ion リスト型パラメータをステートメントに渡すと、1 つの INSERT ステートメントを使用して複数のドキュメントを挿入できます。

```
// people is an Ion list
txn.Execute("INSERT INTO Person ?", people);
```

Ion リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ステップ 5: ドキュメントにクエリを実行する

次のコード例は、SELECT ステートメントの実行方法を示しています。

Person テーブルからドキュメントをクエリする以下のコードを追加します。

Async

```
Console.WriteLine("Querying the table");

// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IAsyncResult<Person> selectResult = await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE FirstName
 = ?", "John");
    return await txn.Execute(myQuery);
});

await foreach (Person person in selectResult)
{
    Console.WriteLine(person);
    // John, Doe, 32
}
```

Sync

```
Console.WriteLine("Querying the table");

// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult<Person> selectResult = driver.Execute(txn =>
```

```
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE FirstName
 = ?", "John");
    return txn.Execute(myQuery);
});

foreach (Person person in selectResult)
{
    Console.WriteLine(person);
    // John, Doe, 32
}
```

Ion ライブラリの使用

Async

```
Console.WriteLine("Querying the table");

IIonValue ionFirstName = valueFactory.NewString("John");

// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IAsyncResult selectResult = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName);
});

await foreach (IIonValue row in selectResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

Sync

```
Console.WriteLine("Querying the table");

IIonValue ionFirstName = valueFactory.NewString("John");

// The result from driver.Execute() is buffered into memory because once the
```

```
// transaction is committed, streaming the result is no longer possible.
IResult selectResult = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE firstName = ?", ionFirstName);
});

foreach (IIonValue row in selectResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

この例では、疑問符 (?) を変数プレースホルダーとして使用して、ドキュメント情報をステートメントに渡します。プレースホルダーを使用する場合、IonValue 型の値を渡す必要があります。

ステップ 6: ドキュメントを更新する

次のコード例は、UPDATE ステートメントの実行方法を示しています。

1. age を 42 に変更して Person テーブルのドキュメントを更新する以下のコードを追加します。

   Async

   ```
   Console.WriteLine("Updating the document");

   await driver.Execute(async txn =>
   {
       IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age = ? WHERE
    FirstName = ?", 42, "John");
       await txn.Execute(myQuery);
   });
   ```

   Sync

   ```
   Console.WriteLine("Updating the document");

   driver.Execute(txn =>
   {
       IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age = ? WHERE
    FirstName = ?", 42, "John");
       txn.Execute(myQuery);
   ```

```
});
```

2. ドキュメントを再度クエリして、更新された値を確認します。

Async

```
Console.WriteLine("Querying the table for the updated document");

IAsyncResult<Person> updateResult = await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE
 FirstName = ?", "John");
    return await txn.Execute(myQuery);
});

await foreach (Person person in updateResult)
{
    Console.WriteLine(person);
    // John, Doe, 42
}
```

Sync

```
Console.WriteLine("Querying the table for the updated document");

IResult<Person> updateResult = driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE
 FirstName = ?", "John");
    return txn.Execute(myQuery);
});

foreach (Person person in updateResult)
{
    Console.WriteLine(person);
    // John, Doe, 42
}
```

3. アプリケーションを実行するには、Amazon.QLDB.QuickStartGuide プロジェクトディレクトリの親ディレクトリから以下のコマンドを入力します。

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

Ion ライブラリの使用

1. age を 42 に変更して Person テーブルのドキュメントを更新する以下のコードを追加します。

   Async

   ```
   Console.WriteLine("Updating the document");

   IIonValue ionIntAge = valueFactory.NewInt(42);
   IIonValue ionFirstName2 = valueFactory.NewString("John");

   await driver.Execute(async txn =>
   {
       await txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
    ionIntAge, ionFirstName2);
   });
   ```

   Sync

   ```
   Console.WriteLine("Updating a document");

   IIonValue ionIntAge = valueFactory.NewInt(42);
   IIonValue ionFirstName2 = valueFactory.NewString("John");

   driver.Execute(txn =>
   {
       txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?", ionIntAge,
    ionFirstName2);
   });
   ```

2. ドキュメントを再度クエリして、更新された値を確認します。

   Async

   ```
   Console.WriteLine("Querying the table for the updated document");

   IIonValue ionFirstName3 = valueFactory.NewString("John");

   IAsyncResult updateResult = await driver.Execute(async txn =>
   {
       return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
    ionFirstName3 );
   });
   ```

```
await foreach (IIonValue row in updateResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

Sync

```
Console.WriteLine("Querying the table for the updated document");

IIonValue ionFirstName3 = valueFactory.NewString("John");

IResult updateResult = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName3);
});

foreach (IIonValue row in updateResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

3. アプリケーションを実行するには、Amazon.QLDB.QuickStartGuide プロジェクトディレクトリの親ディレクトリから以下のコマンドを入力します。

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

完全なアプリケーションの実行

以下のコード例は、Program.cs アプリケーションの完全なバージョンです。上のステップを個別に実行する代わりに、このコード例を最初から最後までコピーして実行することもできます。このアプリケーションは、quick-start という名前の台帳に対するいくつかの基本的な CRUD オペレーションを実行します。

> **Note**
>
> このコードを実行する前に、quick-start 台帳に Person という名前のアクティブなテーブルが存在しないことを確認してください。

Async

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();

            Console.WriteLine("Creating the table and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
```

```
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
            await driver.Execute(async txn =>
            {
                await txn.Execute("CREATE TABLE Person");
                await txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            Person myPerson = new Person {
                FirstName = "John",
                LastName = "Doe",
                Age = 32
            };

            await driver.Execute(async txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?",
 myPerson);
                await txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table");

            // The result from driver.Execute() is buffered into memory because once
 the
            // transaction is committed, streaming the result is no longer possible.
            IAsyncResult<Person> selectResult = await driver.Execute(async txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
 WHERE FirstName = ?", "John");
                return await txn.Execute(myQuery);
            });

            await foreach (Person person in selectResult)
            {
                Console.WriteLine(person);
                // John, Doe, 32
            }

            Console.WriteLine("Updating the document");

            await driver.Execute(async txn =>
```

```
            {
                IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age
 = ? WHERE FirstName = ?", 42, "John");
                await txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table for the updated document");

            IAsyncResult<Person> updateResult = await driver.Execute(async txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
 WHERE FirstName = ?", "John");
                return await txn.Execute(myQuery);
            });

            await foreach (Person person in updateResult)
            {
                Console.WriteLine(person);
                // John, Doe, 42
            }
        }
    }
}
```

Sync

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
```

```
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();

            Console.WriteLine("Creating the table and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
            driver.Execute(txn =>
            {
                txn.Execute("CREATE TABLE Person");
                txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            Person myPerson = new Person {
                FirstName = "John",
                LastName = "Doe",
                Age = 32
            };

            driver.Execute(txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?",
 myPerson);
                txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table");
```

```
            // The result from driver.Execute() is buffered into memory because once
 the
            // transaction is committed, streaming the result is no longer possible.
            IResult<Person> selectResult = driver.Execute(txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
 WHERE FirstName = ?", "John");
                return txn.Execute(myQuery);
            });

            foreach (Person person in selectResult)
            {
                Console.WriteLine(person);
                // John, Doe, 32
            }

            Console.WriteLine("Updating the document");

            driver.Execute(txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age
 = ? WHERE FirstName = ?", 42, "John");
                txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table for the updated document");

            IResult<Person> updateResult = driver.Execute(txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
 WHERE FirstName = ?", "John");
                return txn.Execute(myQuery);
            });

            foreach (Person person in updateResult)
            {
                Console.WriteLine(person);
                // John, Doe, 42
            }

        }
    }
}
```

Ion ライブラリの使用

Async

```
using System;
using System.Threading.Tasks;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();

            Console.WriteLine("Creating the table and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
            await driver.Execute(async txn =>
            {
                await txn.Execute("CREATE TABLE Person");
                await txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            // This is one way of creating Ion values. We can also use an IonLoader.
            // For more details, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
            IIonValue ionPerson = valueFactory.NewEmptyStruct();
            ionPerson.SetField("firstName", valueFactory.NewString("John"));
```

```
            ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
            ionPerson.SetField("age", valueFactory.NewInt(32));

            await driver.Execute(async txn =>
            {
                await txn.Execute("INSERT INTO Person ?", ionPerson);
            });

            Console.WriteLine("Querying the table");

            IIonValue ionFirstName = valueFactory.NewString("John");

            // The result from driver.Execute() is buffered into memory because once
the
            // transaction is committed, streaming the result is no longer possible.
            IAsyncResult selectResult = await driver.Execute(async txn =>
            {
                return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName);
            });

            await foreach (IIonValue row in selectResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }

            Console.WriteLine("Updating the document");

            IIonValue ionIntAge = valueFactory.NewInt(42);
            IIonValue ionFirstName2 = valueFactory.NewString("John");

            await driver.Execute(async txn =>
            {
                await txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
ionIntAge, ionFirstName2);
            });

            Console.WriteLine("Querying the table for the updated document");

            IIonValue ionFirstName3 = valueFactory.NewString("John");

            IAsyncResult updateResult = await driver.Execute(async txn =>
```

```
            {
                return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName3);
            });

            await foreach (IIonValue row in updateResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }
        }
    }
}
```

Sync

```
using System;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();

            Console.WriteLine("Creating the tables and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
            driver.Execute(txn =>
```

```
            {
                txn.Execute("CREATE TABLE Person");
                txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            // This is one way of creating Ion values. We can also use an IonLoader.
            // For more details, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
            IIonValue ionPerson = valueFactory.NewEmptyStruct();
            ionPerson.SetField("firstName", valueFactory.NewString("John"));
            ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
            ionPerson.SetField("age", valueFactory.NewInt(32));

            driver.Execute(txn =>
            {
                txn.Execute("INSERT INTO Person ?", ionPerson);
            });

            Console.WriteLine("Querying the table");

            IIonValue ionFirstName = valueFactory.NewString("John");

            // The result from driver.Execute() is buffered into memory because once
 the
            // transaction is committed, streaming the result is no longer possible.
            IResult selectResult = driver.Execute(txn =>
            {
                return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName);
            });

            foreach (IIonValue row in selectResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }

            Console.WriteLine("Updating a document");

            IIonValue ionIntAge = valueFactory.NewInt(42);
            IIonValue ionFirstName2 = valueFactory.NewString("John");
```

```
            driver.Execute(txn =>
            {
                txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
 ionIntAge, ionFirstName2);
            });

            Console.WriteLine("Querying the table for the updated document");

            IIonValue ionFirstName3 = valueFactory.NewString("John");

            IResult updateResult = driver.Execute(txn =>
            {
                return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName3);
            });

            foreach (IIonValue row in updateResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }
        }
    }
}
```

アプリケーション全体を実行するには、Amazon.QLDB.QuickStartGuide プロジェクトディレクトリの親ディレクトリから以下のコマンドを入力します。

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

.NET 用 Amazon QLDB ドライバー — クックブックリファレンス

このリファレンスガイドでは、.NET 用 Amazon QLDB ドライバーの一般的なユースケースについて説明します。C# コード例を示して、ドライバーを使用した基本的な作成、読み取り、更新、削除 (CRUD) オペレーションの実行方法を解説します。Amazon Ion データを処理するためのコード例も含まれています。さらに、このガイドでは、トランザクションをべき等にし、一意性の制約を実装するためのベストプラクティスを取り上げています。

> **Note**
>
> このトピックでは、デフォルトで Ion オブジェクトマッパーを使用して Amazon Ion データを処理するコード例を紹介します。QLDB は、.NET ドライバーのバージョン 1.3.0 で Ion オブジェクトマッパーを導入しました。このトピックでは、該当する場合、標準の Ion ライブラリを代替として使用するコード例も紹介します。詳細については、「Amazon Ion の操作」を参照してください。

目次

ドライバーのインポート

次のコード例では、ドライバーをインポートします。

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;
```

Ion ライブラリの使用

```
using Amazon.QLDB.Driver;
using Amazon.IonDotnet.Builders;
```

ドライバーのインスタンス化

次のコード例は、デフォルト設定を使用して指定された台帳名に接続するドライバーのインスタンスを作成します。

Async

```
IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
    .WithLedger("vehicle-registration")
    // Add Serialization library
    .WithSerializer(new ObjectSerializer())
    .Build();
```

Sync

```
IQldbDriver driver = QldbDriver.Builder()
    .WithLedger("vehicle-registration")
    // Add Serialization library
    .WithSerializer(new ObjectSerializer())
    .Build();
```

Ion ライブラリの使用

Async

```
IAsyncQldbDriver driver = AsyncQldbDriver.Builder().WithLedger("vehicle-
registration").Build();
```

Sync

```
IQldbDriver driver = QldbDriver.Builder().WithLedger("vehicle-
registration").Build();
```

CRUD オペレーション

QLDB は作成、読み取り、更新、および削除 (CRUD) オペレーションをトランザクションの一部として実行します。

> ⚠ Warning
>
> ベストプラクティスとして、書き込みトランザクションを厳密にべき等にしてください。

トランザクションをべき等にする

再試行の場合に予期しない結果を避けるために、書き込みトランザクションをべき等にすることをお勧めします。トランザクションは、複数回実行して毎回同じ結果を生成できる場合、idempotent です。

例えば、`Person` という名前のテーブルにドキュメントを挿入するトランザクションについて考えてみます。トランザクションは、まずドキュメントがテーブル内に既に存在するかどうかをチェックする必要があります。このチェックを行わないと、テーブルに重複するドキュメントができる可能性があります。

QLDB がサーバー側でトランザクションを正常にコミットできるが、レスポンスを待機中にクライアントはタイムアウトするとします。トランザクションがべき等でない場合、再試行時に同じドキュメントが複数回挿入される可能性があります。

インデックスを使用してテーブル全体のスキャンを回避する

インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する `WHERE` 述語句でステートメントを実行する (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`) こともお勧めします。このインデックス付きのルックアップがない場合、QLDB はテーブルスキャンを実行する必要があり、トランザクションタイムアウトやオプティミスティック同時実行制御 (OCC) 競合が発生する可能性があります。

OCC の詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

暗黙的に作成されたトランザクション

Amazon.QLDB.Driver.IQldbDriver.Execute メソッドは Amazon.QLDB.Driver.TransactionExecutor のインスタンス (これを使用してステートメントを実行可能) を受け取る Lambda 関数を受け入れます。TransactionExecutor のインスタンスは、暗黙的に作成されたトランザクションをラップします。

Lambda 関数内でステートメントを実行するには、トランザクションエグゼキューターの Execute メソッドを使用します。ドライバーは、Lambda 関数が戻ったときに暗黙的にトランザクションをコミットします。

次のセクションでは、基本的な CRUD オペレーションの実行、カスタム再試行ロジックの指定、一意性制約の実装方法について説明します。

目次

テーブルの作成

Async

```
IAsyncResult<Table> createResult = await driver.Execute(async txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE TABLE Person");
    return await txn.Execute(query);
});
```

```
await foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the created table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Sync

```
IResult<Table> createResult = driver.Execute( txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE TABLE Person");
    return txn.Execute(query);
});

foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the created table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Ion ライブラリの使用

Async

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("CREATE TABLE Person");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

Sync

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult result = driver.Execute(txn =>
{
    return txn.Execute("CREATE TABLE Person");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

インデックスの作成

Async

```
IAsyncResult<Table> createResult = await driver.Execute(async txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE INDEX ON Person(firstName)");
    return await txn.Execute(query);
});

await foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the updated table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Sync

```
IResult<Table> createResult = driver.Execute(txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE INDEX ON Person(firstName)");
    return txn.Execute(query);
});
```

```
foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the updated table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Ion ライブラリの使用

Async

```
IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("CREATE INDEX ON Person(GovId)");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

Sync

```
IResult result = driver.Execute(txn =>
{
    return txn.Execute("CREATE INDEX ON Person(GovId)");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

ドキュメントの読み取り

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// Person class is defined as follows:
// public class Person
// {
//     public string GovId { get; set; }
//     public string FirstName { get; set; }
//  }

IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'"));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

> **Note**
>
> インデックス付きルックアップなしでクエリを実行すると、完全なテーブルスキャンが呼び出されます。この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、クエリのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

クエリパラメータの使用

次のコード例では、C# 型のクエリパラメータを使用します。

```
IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE FirstName
 = ?", "Brent"));
});
```

```
await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

次のコード例では、複数の C# 型のクエリパラメータを使用します。

```
IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId = ?
 AND FirstName = ?", "TOYENC486FH", "Brent"));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

次のコード例では、C# 型のクエリパラメータの配列を使用します。

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }

string[] ids = {
    "TOYENC486FH",
    "ROEE1C1AABH",
    "YH844DA7LDB"
};

IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId IN
 (?,?,?)", ids));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.FirstName); // Prints Brent on first iteration.
```

```
    // Prints Jim on second iteration.
    // Prints Mary on third iteration.
}
```

次のコード例では、C# リストを値に使用しています。

```
// Assumes that Person table has document as follows:
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//      { "Make": "Volkswagen",
//        "Model": "Golf"},
//      { "Make": "Honda",
//        "Model": "Civic"}
//   ]
// }
// Person class is defined as follows:
// public class Person
// {
//     public string GovId { get; set; }
//     public string FirstName { get; set; }
//     public List<Vehicle> Vehicles { get; set; }
// }
// Vehicle class is defined as follows:
// public class Vehicle
// {
//     public string Make { get; set; }
//     public string Model { get; set; }
// }

List<Vehicle> vehicles = new List<Vehicle>
{
    new Vehicle
    {
        Make = "Volkswagen",
        Model = "Golf"
    },
    new Vehicle
    {
        Make = "Honda",
        Model = "Civic"
    }
};
```

```
IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE Vehicles
 = ?", vehicles));
});

await foreach (Person person in result)
{
    Console.WriteLine("{");
    Console.WriteLine($"  GovId: {person.GovId},");
    Console.WriteLine($"  FirstName: {person.FirstName},");
    Console.WriteLine("  Vehicles: [");
    foreach (Vehicle vehicle in person.Vehicles)
    {
        Console.WriteLine("  {");
        Console.WriteLine($"    Make: {vehicle.Make},");
        Console.WriteLine($"    Model: {vehicle.Model},");
        Console.WriteLine("  },");
    }
    Console.WriteLine("  ]");
    Console.WriteLine("}");
    // Prints:
    // {
    //     GovId: TOYENC486FH,
    //     FirstName: Brent,
    //     Vehicles: [
    //     {
    //         Make: Volkswagen,
    //         Model: Golf
    //     },
    //     {
    //         Make: Honda,
    //         Model: Civic
    //     },
    //     ]
    // }
}
```

Ion ライブラリの使用

Async

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

> **Note**
>
> インデックス付きルックアップなしでクエリを実行すると、完全なテーブルスキャンが呼び出されます。この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、クエリのレイテ

ンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

次のコード例では、Ion 型のクエリパラメータを使用します。

Async

```
IValueFactory valueFactory = new ValueFactory();
IIonValue ionFirstName = valueFactory.NewString("Brent");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE FirstName = ?",
 ionFirstName);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
IValueFactory valueFactory = new ValueFactory();
IIonValue ionFirstName = valueFactory.NewString("Brent");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE FirstName = ?", ionFirstName);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

次のコード例では、複数のクエリパラメータを使用します。

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("Brent");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName
 = ?", ionGovId, ionFirstName);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("Brent");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?",
 ionGovId, ionFirstName);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

次のコード例では、クエリパラメータのリストを使用します。

Async

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }
```

```
IIonValue[] ionIds = {
    valueFactory.NewString("TOYENC486FH"),
    valueFactory.NewString("ROEE1C1AABH"),
    valueFactory.NewString("YH844DA7LDB")
};

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId IN (?,?,?)", ionIds);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent on
 first iteration.
                                                              // Prints Jim on
 second iteration.
                                                              // Prints Mary on
 third iteration.
}
```

Sync

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }

IIonValue[] ionIds = {
    valueFactory.NewString("TOYENC486FH"),
    valueFactory.NewString("ROEE1C1AABH"),
    valueFactory.NewString("YH844DA7LDB")
};

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId IN (?,?,?)", ionIds);
});

foreach (IIonValue row in result)
{
```

```
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent on
 first iteration.
                                                              // Prints Jim on
 second iteration.
                                                              // Prints Mary on
 third iteration.
}
```

次のコード例では、Ion リストを値に使用しています。Ion のさまざまなデータ型の詳細については、「Amazon QLDB で Amazon Ion のデータ型を操作する」を参照してください。

Async

```
// Assumes that Person table has document as follows:
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//      { "Make": "Volkswagen",
//        "Model": "Golf"},
//      { "Make": "Honda",
//        "Model": "Civic"}
//   ]
// }

IIonValue ionVehicle1 = valueFactory.NewEmptyStruct();
ionVehicle1.SetField("Make", valueFactory.NewString("Volkswagen"));
ionVehicle1.SetField("Model", valueFactory.NewString("Golf"));

IIonValue ionVehicle2 = valueFactory.NewEmptyStruct();
ionVehicle2.SetField("Make", valueFactory.NewString("Honda"));
ionVehicle2.SetField("Model", valueFactory.NewString("Civic"));

IIonValue ionVehicles =  valueFactory.NewEmptyList();
ionVehicles.Add(ionVehicle1);
ionVehicles.Add(ionVehicle2);

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE Vehicles = ?",
 ionVehicles);
});
```

```
await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // Prints:
    // {
    //     GovId: "TOYENC486FN",
    //     FirstName: "Brent",
    //     Vehicles: [
    //     {
    //         Make: "Volkswagen",
    //         Model: "Golf"
    //     },
    //     {
    //         Make: "Honda",
    //         Model: "Civic"
    //     }
    //     ]
    // }
}
```

Sync

```
// Assumes that Person table has document as follows:
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//      { "Make": "Volkswagen",
//        "Model": "Golf"},
//      { "Make": "Honda",
//        "Model": "Civic"}
//   ]
// }

IIonValue ionVehicle1 = valueFactory.NewEmptyStruct();
ionVehicle1.SetField("Make", valueFactory.NewString("Volkswagen"));
ionVehicle1.SetField("Model", valueFactory.NewString("Golf"));

IIonValue ionVehicle2 = valueFactory.NewEmptyStruct();
ionVehicle2.SetField("Make", valueFactory.NewString("Honda"));
ionVehicle2.SetField("Model", valueFactory.NewString("Civic"));

IIonValue ionVehicles =  valueFactory.NewEmptyList();
ionVehicles.Add(ionVehicle1);
```

```
ionVehicles.Add(ionVehicle2);

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE Vehicles = ?", ionVehicles);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // Prints:
    // {
    //     GovId: "TOYENC486FN",
    //     FirstName: "Brent",
    //     Vehicles: [
    //     {
    //         Make: "Volkswagen",
    //         Model: "Golf"
    //     },
    //     {
    //         Make: "Honda",
    //         Model: "Civic"
    //     }
    //     ]
    // }
}
```

ドキュメントの挿入

次のコード例では、Ion データ型を挿入します。

```
string govId = "TOYENC486FH";

Person person = new Person
{
    GovId = "TOYENC486FH",
    FirstName = "Brent"
};

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
```

```
    IAsyncResult<Person> result = await txn.Execute(txn.Query<Person>("SELECT * FROM
 Person WHERE GovId = ?", govId));

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", person));
});
```

Ion ライブラリの使用

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult result = await txn.Execute("SELECT * FROM Person WHERE GovId = ?",
 ionGovId);

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

driver.Execute(txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IResult result = txn.Execute("SELECT * FROM Person WHERE GovId = ?", ionGovId);

    // Check if there is a record in the cursor.
    int count = result.Count();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

このトランザクションは、ドキュメントを Person テーブルに挿入します。挿入する前に、まずドキュメントがテーブル内に既に存在しているかどうかをチェックします。このチェックにより、トランザクションは本質的にべき等になります。このトランザクションを複数回実行しても、意図しない副作用は発生しません。

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

1 つのステートメントで複数のドキュメントの挿入

次のように C# List パラメータをステートメントに渡すと、1 つの INSERT ステートメントを使用して複数のドキュメントを挿入できます。

```
Person person1 = new Person
{
    FirstName = "Brent",
    GovId = "TOYENC486FH"
};

Person person2 = new Person
{
    FirstName = "Jim",
    GovId = "ROEE1C1AABH"
};

List<Person> people = new List<Person>();
people.Add(person1);
people.Add(person2);

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", people));
});

await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the created documents' ID:
    // { documentId: 6BFt5eJQDFLBW2aR8LPw42 }
    // { documentId: K5Zrcb6N3gmIEHgGhwoyKF }
}
```

Ion ライブラリの使用

次のように Ion リスト型パラメータをステートメントに渡すと、1 つの INSERT ステートメントを使用して複数のドキュメントを挿入できます。

Async

```
IIonValue ionPerson1 = valueFactory.NewEmptyStruct();
ionPerson1.SetField("FirstName", valueFactory.NewString("Brent"));
```

```
ionPerson1.SetField("GovId", valueFactory.NewString("TOYENC486FH"));

IIonValue ionPerson2 = valueFactory.NewEmptyStruct();
ionPerson2.SetField("FirstName", valueFactory.NewString("Jim"));
ionPerson2.SetField("GovId", valueFactory.NewString("ROEE1C1AABH"));

IIonValue ionPeople = valueFactory.NewEmptyList();
ionPeople.Add(ionPerson1);
ionPeople.Add(ionPerson2);

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("INSERT INTO Person ?", ionPeople);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created documents' ID:
    // {
    //     documentId: "6BFt5eJQDFLBW2aR8LPw42"
    // }
    //
    // {
    //     documentId: "K5Zrcb6N3gmIEHgGhwoyKF"
    // }
}
```

Sync

```
IIonValue ionPerson1 = valueFactory.NewEmptyStruct();
ionPerson1.SetField("FirstName", valueFactory.NewString("Brent"));
ionPerson1.SetField("GovId", valueFactory.NewString("TOYENC486FH"));

IIonValue ionPerson2 = valueFactory.NewEmptyStruct();
ionPerson2.SetField("FirstName", valueFactory.NewString("Jim"));
ionPerson2.SetField("GovId", valueFactory.NewString("ROEE1C1AABH"));

IIonValue ionPeople = valueFactory.NewEmptyList();
ionPeople.Add(ionPerson1);
ionPeople.Add(ionPerson2);

IResult result = driver.Execute(txn =>
```

```
{
    return txn.Execute("INSERT INTO Person ?", ionPeople);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created documents' ID:
    // {
    //     documentId: "6BFt5eJQDFLBW2aR8LPw42"
    // }
    //
    // {
    //     documentId: "K5Zrcb6N3gmIEHgGhwoyKF"
    // }
}
```

Ion リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ドキュメントの更新

```
string govId = "TOYENC486FH";
string firstName = "John";

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("UPDATE Person SET FirstName = ? WHERE
 GovId = ?", firstName , govId));
});

await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the updated document ID:
    // { documentId: Djg30Zoltqy5M4BFsA2jSJ }
}
```

Ion ライブラリの使用

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("John");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?",
 ionFirstName , ionGovId);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("John");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?",
 ionFirstName , ionGovId);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの削除

```
string govId = "TOYENC486FH";

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("DELETE FROM Person WHERE GovId = ?",
 govId));
});

await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the updated document ID:
    // { documentId: Djg30Zoltqy5M4BFsA2jSJ }
}
```

Ion ライブラリの使用

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("DELETE FROM Person WHERE GovId = ?", ionGovId);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the deleted document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
```

```
    // }
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("DELETE FROM Person WHERE GovId = ?", ionGovId);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the deleted document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

トランザクションで複数のステートメントの実行

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
 not.
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.Generic.IAsyncResult<Vehicle> result = await txn.Execute(
```

```
            txn.Query<Vehicle>("SELECT insured FROM Vehicles WHERE vin = ? AND insured
 = FALSE", vin));

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                txn.Query<Document>("UPDATE Vehicles SET insured = TRUE WHERE vin = ?",
 vin));
            return true;
        }
        return false;
    });
}
```

Ion ライブラリの使用

Async

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    ValueFactory valueFactory = new ValueFactory();
    IIonValue ionVin = valueFactory.NewString(vin);

    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.IAsyncResult result = await txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 ionVin);

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
```

```
    });
}
```

再試行ロジック

ドライバーに組み込まれた再試行ロジックの詳細については、「Amazon QLDB でドライバーが使用する再試行ポリシーを理解する」を参照してください。

一意性制約の実装

QLDB はユニークインデックスをサポートしていませんが、この動作をアプリケーションに実装できます。

Person テーブル内の GovId フィールドに対して一意性制約を実装するとします。これを行うには、以下を実行するトランザクションを記述します。

1. テーブルに指定された GovId を持つ既存のドキュメントがないことをアサートします。
2. アサーションに合格した場合は、ドキュメントを挿入します。

競合するトランザクションが同時にアサーションに合格すると、一方のトランザクションだけが正常にコミットされます。もう一方のトランザクションは OCC 競合例外で失敗します。

次のコード例は、この一意性制約ロジックを実装する方法を示しています。

```
string govId = "TOYENC486FH";

Person person = new Person
{
    GovId = "TOYENC486FH",
    FirstName = "Brent"
};

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult<Person> result = await txn.Execute(txn.Query<Person>("SELECT * FROM
 Person WHERE GovId = ?", govId));

    // Check if there is a record in the cursor.
```

```
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", person));
});
```

Ion ライブラリの使用

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult result = await txn.Execute("SELECT * FROM Person WHERE GovId = ?",
 ionGovId);

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
```

```
IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

driver.Execute(txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IResult result = txn.Execute("SELECT * FROM Person WHERE GovId = ?", ionGovId);

    // Check if there is a record in the cursor.
    int count = result.Count();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

Amazon Ion の操作

QLDB で Amazon Ion データを処理するには、複数の方法があります。Ion ライブラリを使用して Ion 値を作成および変更できます。または、Ion オブジェクトマッパーを使用して C# の Plain Old CLR Objects (POCO) を Ion 値にマッピングしたり、Ion 値からマップしたりすることもできます。.NET 用 QLDB ドライバーのバージョン 1.3.0 では、Ion オブジェクトマッパーのサポートが導入されています。

次のセクションでは、両方の手法を使用して Ion データを処理するコード例について説明します。

目次

Ion モジュールのインポート

```
using Amazon.IonObjectMapper;
```

Ion ライブラリの使用

```
using Amazon.IonDotnet.Builders;
```

Ion 型の作成

次のコード例は、Ion オブジェクトマッパーを使用して C# オブジェクトから Ion 値を作成する方法を示しています。

```
// Assumes that Person class is defined as follows:
// public class Person
// {
//     public string FirstName { get; set; }
//     public int Age { get; set; }
// }

// Initialize the Ion Object Mapper
IonSerializer ionSerializer = new IonSerializer();

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
    Age = 13
};

// Serialize the C# object into stream using the Ion Object Mapper
Stream stream = ionSerializer.Serialize(person);
```

```
// Load will take in stream and return a datagram; a top level container of Ion values.
IIonValue ionDatagram = IonLoader.Default.Load(stream);

// To get the Ion value within the datagram, we call GetElementAt(0).
IIonValue ionPerson = ionDatagram.GetElementAt(0);

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

Ion ライブラリの使用

次のコード例は、Ion ライブラリを使用して Ion 値を作成する 2 つの方法を示しています。

`ValueFactory` を使用する

```
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;

IValueFactory valueFactory = new ValueFactory();

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("age", valueFactory.NewInt(13));

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

`IonLoader` を使用する

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;

// Load will take in Ion text and return a datagram; a top level container of Ion
 values.
IIonValue ionDatagram = IonLoader.Default.Load("{firstName: \"John\", age: 13}");

// To get the Ion value within the datagram, we call GetElementAt(0).
IIonValue ionPerson = ionDatagram.GetElementAt(0);

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

Ion バイナリダンプの取得

```
// Initialize the Ion Object Mapper with Ion binary serialization format
IonSerializer ionSerializer = new IonSerializer(new IonSerializationOptions
{
    Format = IonSerializationFormat.BINARY
});

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
    Age = 13
};

MemoryStream stream = (MemoryStream) ionSerializer.Serialize(person);
Console.WriteLine(BitConverter.ToString(stream.ToArray()));
```

Ion ライブラリの使用

```
// ionObject is an Ion struct
MemoryStream stream = new MemoryStream();
using (var writer = IonBinaryWriterBuilder.Build(stream))
{
    ionObject.WriteTo(writer);
    writer.Finish();
}

Console.WriteLine(BitConverter.ToString(stream.ToArray()));
```

Ion テキストダンプの取得

```
// Initialize the Ion Object Mapper
IonSerializer ionSerializer = new IonSerializer(new IonSerializationOptions
{
    Format = IonSerializationFormat.TEXT
});

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
```

```
    Age = 13
};

MemoryStream stream = (MemoryStream) ionSerializer.Serialize(person);
Console.WriteLine(System.Text.Encoding.UTF8.GetString(stream.ToArray()));
```

Ion ライブラリの使用

```
// ionObject is an Ion struct
StringWriter sw = new StringWriter();
using (var writer = IonTextWriterBuilder.Build(sw))
{
    ionObject.WriteTo(writer);
    writer.Finish();
}

Console.WriteLine(sw.ToString());
```

Ion の操作方法の詳細については、GitHub で Amazon Ion のドキュメントを参照してください。QLDB で Ion を操作するコード例については、「Amazon QLDB で Amazon Ion のデータ型を操作する」を参照してください。

Go 用 Amazon QLDB ドライバー

台帳内のデータを使用するには、AWS が提供するドライバーを使用して Go アプリケーションから Amazon QLDB に接続できます。次のトピックでは、Go 用 QLDB ドライバーの使用を開始する方法について説明します。

トピック

- ドライバーに関するリソース
- 前提条件
- インストール
- Go 用 Amazon QLDB ドライバー — クイックスタートチュートリアル
- Go 用 Amazon QLDB ドライバー — クックブックリファレンス

ドライバーに関するリソース

Go ドライバーでサポートされている機能の詳細については、以下のリソースを参照してください。

- API リファレンス: 3.x、2.x、1.x
- ドライバーのソースコード (GitHub)
- Amazon Ion クックブック

前提条件

Go 用 QLDB ドライバーの使用を開始する前に、次のことを行う必要があります。

1. 「Amazon QLDB へのアクセス」にある AWS の設定手順に従います。これには以下が含まれます。

   1. AWS にサインアップする。
   2. QLDB の適切なアクセス許可を持つユーザーを作成します。
   3. 開発に必要なプログラムへのアクセスを提供します。

2. (オプション) 任意の統合開発環境 (IDE) をインストールします。一般的に使用されている Go IDE の一覧については、Go ウェブサイトの「Editor plugins and IDEs」を参照してください。
3. Go ダウンロードサイトから、Go の以下のバージョニングのいずれかをダウンロードしてインストールしてください。
   - 1.15 またはそれ以降 – Go バージョン 3 用の QLDB ドライバー
   - 1.14 – Go バージョン 1 または 2 用の QLDB ドライバー
4. AWS SDK for Go の開発環境を設定します。

   1. AWS 認証情報をセットアップします。共有認証情報ファイルを作成することをお勧めします。

      手順については、「AWS SDK for Go デベロッパーガイド」の「Specifying Credentials」(認証情報の指定) を参照してください。

   2. デフォルトの AWS リージョン を設定します。この方法については、「Specifying the AWS リージョン」を参照してください。

      利用可能なリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

次に、基本的なサンプルアプリケーションを設定し、短いコード例を実行するか、既存の Go プロジェクトにドライバーをインストールできます。

- 既存のプロジェクトに QLDB ドライバーと AWS SDK for Go をインストールするには、「インストール」に進みます。
- プロジェクトを設定し、台帳の基本的なデータトランザクションを示す短いコード例については、「クイックスタートチュートリアル」を参照してください。

インストール

Go 用 QLDB ドライバーは GitHub リポジトリ awslabs/amazon-qldb-driver-go でオープンソース化されています。QLDB は、以下のドライバーバージョンと Go の依存関係をサポートしています。

ドライバーのバージョン	Go バージョニング	ステータス	最新リリース日
1.x	1.14 以降	本番リリース	2021年6月16日
2.x	1.14 以降	本番リリース	2021 年 7 月 21 日
3.x	1.15 以降	本番リリース	2022 年 11 月 10 日

ドライバーをインストールするには

1. プロジェクトの依存関係をインストールする Go モジュールをプロジェクトで使用していることを確認します。
2. プロジェクトディレクトリで次の `go get` コマンドを入力します。

   3.x

   ```
   $ go get -u github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver
   ```

   2.x

   ```
   $ go get -u github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver
   ```

ドライバーをインストールすると、AWS SDK for Go や AWS SDK for Go バージョン 2、Amazon Ion パッケージなどの依存関係もインストールされます。

台帳に対して基本的なデータトランザクションを実行する方法を示す短いコード例については、「クックブックリファレンス」を参照してください。

Go 用 Amazon QLDB ドライバー — クイックスタートチュートリアル

このチュートリアルでは、Go 用 Amazon QLDB ドライバーの最新バージョンを使用して単純なアプリケーションを設定する方法について説明します。このガイドには、ドライバのインストール手順と、作成、読み取り、更新、削除 (CRUD) の基本的なオペレーションを実行する短いコード例が含まれています。

トピック

前提条件

作業を始める前に、次の操作を実行してください。

1. Go ドライバー用の「前提条件」を完了します (まだ完了していない場合)。これには、AWS へのサインアップ、開発のためのプログラムによるアクセスの許可、Go のインストールが含まれます。
2. `quick-start` という名前の台帳を作成します。

   台帳の作成方法については、「Amazon QLDB 台帳の基本的なオペレーション」、または「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」を参照してください。

ステップ 1: ドライバーをインストールする

プロジェクトの依存関係をインストールする Go モジュールをプロジェクトで使用していることを確認します。

プロジェクトディレクトリで次の go get コマンドを入力します。

```
$ go get -u github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver
```

ドライバーをインストールすると、AWS SDK for Go v2 パッケージや Amazon Ion パッケージなどの依存関係もインストールされます。

ステップ 2: パッケージをインポートする

次の AWS パッケージをインポートします。

```
import (
    "context"
    "fmt"

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/qldbSession"
    "github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver"
)
```

ステップ 3: ドライバーを初期化する

quick-start という名前の台帳に接続するドライバーのインスタンスを初期化します。

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
  panic(err)
}

qldbSession := qldbsession.NewFromConfig(cfg, func(options *qldbsession.Options) {
    options.Region = "us-east-1"
})
driver, err := qldbdriver.New(
    "quick-start",
    qldbSession,
```

```
    func(options *qldbdriver.DriverOptions) {
        options.LoggerVerbosity = qldbdriver.LogInfo
    })
if err != nil {
    panic(err)
}

defer driver.Shutdown(context.Background())
```

> **Note**
>
> このコード例では、*us-east-1* を、台帳を作成した AWS リージョン に置き換えます。

ステップ 4: テーブルとインデックスを作成する

以下のコード例は、CREATE TABLE および CREATE INDEX ステートメントの実行方法を示しています。

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    _, err := txn.Execute("CREATE TABLE People")
    if err != nil {
        return nil, err
    }

    // When working with QLDB, it's recommended to create an index on fields we're
 filtering on.
    // This reduces the chance of OCC conflict exceptions with large datasets.
    _, err = txn.Execute("CREATE INDEX ON People (firstName)")
    if err != nil {
        return nil, err
    }

    _, err = txn.Execute("CREATE INDEX ON People (age)")
    if err != nil {
        return nil, err
    }

    return nil, nil
})
if err != nil {
```

```
    panic(err)
}
```

このコードにより、People という名前のテーブルと、そのテーブルにある firstName フィールドと age フィールドのインデックスが作成されます。クエリのパフォーマンスを最適化し、オプティミスティック同時実行制御 (OCC) 競合例外を制限するために、インデックスが必要です。

ステップ 5: ドキュメントを挿入する

次のコード例は、INSERT ステートメントの実行方法を示しています。QLDB は PartiQL クエリ言語 (SQL 互換) と Amazon Ion データ形式 (JSON のスーパーセット) をサポートします。

リテラル PartiQL の使用

次のコードでは、文字列のリテラル PartiQL ステートメントを使用して、People テーブルにドキュメントを挿入しています。

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("INSERT INTO People {'firstName': 'Jane', 'lastName': 'Doe',
 'age': 77}")
})
if err != nil {
    panic(err)
}
```

Ion データ型の使用

Go の組み込み JSON パッケージと同様に、Ion との間で Go データ型のマーシャルとアンマーシャルを行えます。

1. Person という名前を持つ次のような Go 構造体があるとします。

   ```
   type Person struct {
       FirstName string `ion:"firstName"`
       LastName  string `ion:"lastName"`
       Age       int    `ion:"age"`
   }
   ```

2. Person のインスタンスを作成します。

   ```
   person := Person{"John", "Doe", 54}
   ```

ドライバーによって、Ion エンコードされた、person のテキスト表現がマーシャルされます。

> **Important**
>
> マーシャルとアンマーシャルを適切に機能させるには、Go データ構造体のフィールド名をエクスポートする必要があります (先頭文字は大文字で表記)。

3. その person インスタンスをトランザクションの Execute メソッドに渡します。

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("INSERT INTO People ?", person)
})
if err != nil {
    panic(err)
}
```

この例では、疑問符 (?) を変数プレースホルダーとして使用して、ドキュメント情報をステートメントに渡します。プレースホルダーを使用する場合、Ion エンコードされたテキスト値を渡す必要があります。

> **Tip**
>
> 1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 list のパラメータをステートメントに渡すことができます。
>
> ```
> // people is a list
> txn.Execute("INSERT INTO People ?", people)
> ```
>
> リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ステップ 6: クエリを実行してドキュメントを取得する

次のコード例は、SELECT ステートメントの実行方法を示しています。

```
p, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE age =
 54")
    if err != nil {
        return nil, err
    }

    // Assume the result is not empty
    hasNext := result.Next(txn)
    if !hasNext && result.Err() != nil {
        return nil, result.Err()
    }

    ionBinary := result.GetCurrentData()

    temp := new(Person)
    err = ion.Unmarshal(ionBinary, temp)
    if err != nil {
        return nil, err
    }

    return *temp, nil
})
if err != nil {
    panic(err)
}

var returnedPerson Person
returnedPerson = p.(Person)

if returnedPerson != person {
    fmt.Print("Queried result does not match inserted struct")
}
```

この例では、People テーブルからドキュメントを取得するクエリが実行 (結果セットは空でないと仮定) され、その結果からドキュメントが返ります。

ステップ 7: ドキュメントを更新する

次のコード例は、UPDATE ステートメントの実行方法を示しています。

```
person.Age += 10
```

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("UPDATE People SET age = ? WHERE firstName = ?", person.Age,
 person.FirstName)
})
if err != nil {
    panic(err)
}
```

ステップ 8: クエリを実行して更新済みドキュメントを取得する

次のコード例では、People テーブルに、firstName を使用してクエリが実行され、結果セット内のすべてのドキュメントが返ります。

```
p, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
 firstName = ?", person.FirstName)
    if err != nil {
        return nil, err
    }

    var people []Person
    for result.Next(txn) {
        ionBinary := result.GetCurrentData()

        temp := new(Person)
        err = ion.Unmarshal(ionBinary, temp)
        if err != nil {
            return nil, err
        }

        people = append(people, *temp)
    }
    if result.Err() != nil {
        return nil, result.Err()
    }

    return people, nil
})
if err != nil {
    panic(err)
```

```
}

var people []Person
people = p.([]Person)

updatedPerson := Person{"John", "Doe", 64}
if people[0] != updatedPerson {
    fmt.Print("Queried result does not match updated struct")
}
```

ステップ 9: テーブルを削除する

次のコード例は、DROP TABLE ステートメントの実行方法を示しています。

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("DROP TABLE People")
})
if err != nil {
    panic(err)
}
```

完全なアプリケーションの実行

以下のコード例は、 アプリケーションの完全なバージョンです。上のステップを個別に実行する代わりに、このコード例を最初から最後までコピーして実行することもできます。このアプリケーションは、quick-start という名前の台帳に対するいくつかの基本的な CRUD オペレーションを実行します。

> **Note**
>
> このコードを実行する前に、quick-start 台帳に People という名前のアクティブなテーブルが存在しないことを確認してください。

```
package main

import (
    "context"
    "fmt"
```

```
    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

func main() {
    awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
    qldbSession := qldbsession.New(awsSession)

    driver, err := qldbdriver.New(
        "quick-start",
        qldbSession,
        func(options *qldbdriver.DriverOptions) {
            options.LoggerVerbosity = qldbdriver.LogInfo
        })
    if err != nil {
        panic(err)
    }
    defer driver.Shutdown(context.Background())

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
        _, err := txn.Execute("CREATE TABLE People")
        if err != nil {
            return nil, err
        }

        // When working with QLDB, it's recommended to create an index on fields we're
 filtering on.
        // This reduces the chance of OCC conflict exceptions with large datasets.
        _, err = txn.Execute("CREATE INDEX ON People (firstName)")
        if err != nil {
            return nil, err
        }

        _, err = txn.Execute("CREATE INDEX ON People (age)")
        if err != nil {
            return nil, err
        }

        return nil, nil
```

```
    })
    if err != nil {
        panic(err)
    }

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        return txn.Execute("INSERT INTO People {'firstName': 'Jane', 'lastName': 'Doe',
'age': 77}")
    })
    if err != nil {
        panic(err)
    }

    type Person struct {
        FirstName string `ion:"firstName"`
        LastName  string `ion:"lastName"`
        Age       int    `ion:"age"`
    }

    person := Person{"John", "Doe", 54}

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        return txn.Execute("INSERT INTO People ?", person)
    })
    if err != nil {
        panic(err)
    }

    p, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
age = 54")
        if err != nil {
            return nil, err
        }

        // Assume the result is not empty
        hasNext := result.Next(txn)
        if !hasNext && result.Err() != nil {
            return nil, result.Err()
        }
```

```
        ionBinary := result.GetCurrentData()

        temp := new(Person)
        err = ion.Unmarshal(ionBinary, temp)
        if err != nil {
            return nil, err
        }

        return *temp, nil
    })
    if err != nil {
        panic(err)
    }

    var returnedPerson Person
    returnedPerson = p.(Person)

    if returnedPerson != person {
        fmt.Print("Queried result does not match inserted struct")
    }

    person.Age += 10

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        return txn.Execute("UPDATE People SET age = ? WHERE firstName = ?", person.Age,
person.FirstName)
    })
    if err != nil {
        panic(err)
    }

    p, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
firstName = ?", person.FirstName)
        if err != nil {
            return nil, err
        }

        var people []Person
        for result.Next(txn) {
            ionBinary := result.GetCurrentData()
```

```
            temp := new(Person)
            err = ion.Unmarshal(ionBinary, temp)
            if err != nil {
                return nil, err
            }

            people = append(people, *temp)
        }
        if result.Err() != nil {
            return nil, result.Err()
        }

        return people, nil
    })
    if err != nil {
        panic(err)
    }

    var people []Person
    people = p.([]Person)

    updatedPerson := Person{"John", "Doe", 64}
    if people[0] != updatedPerson {
        fmt.Print("Queried result does not match updated struct")
    }

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
        return txn.Execute("DROP TABLE People")
    })
    if err != nil {
        panic(err)
    }
}
```

Go 用 Amazon QLDB ドライバー — クックブックリファレンス

このリファレンスガイドでは、Go 用の Amazon QLDB ドライバーの一般的なユースケースについて説明します。Go コード例を示して、ドライバーを使用した基本的な作成、読み取り、更新、削除 (CRUD) オペレーションの実行方法を解説します。Amazon Ion データを処理するためのコード例も含まれています。さらに、このガイドでは、トランザクションをべき等にし、一意性の制約を実装するためのベストプラクティスを取り上げています。

> **Note**
>
> 該当する場合、一部のユースケースでは、サポートされているメジャーバージョンの Go 用 QLDB ドライバーに対して異なるコード例があります。

目次

ドライバーのインポート

次のコード例では、ドライバーとその他の必要な AWS パッケージをインポートします。

3.x

```
import (

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/qldbSession"
    "github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver"
)
```

2.x

```
import (

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)
```

> **Note**
>
> この例では、Amazon Ion パッケージ (amzn/ion-go/ion) もインポートします。このリファレンスでいくつかのデータオペレーションを実行するときに Ion データを処理するには、このパッケージが必要です。詳細については、「Amazon Ion の操作」を参照してください。

ドライバーのインスタンス化

次のコード例では、指定した台帳名に接続するドライバーのインスタンスを、指定した AWS リージョン に作成します。

3.x

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
```

```
}

qldbSession := qldbsession.NewFromConfig(cfg, func(options *qldbsession.Options) {
    options.Region = "us-east-1"
})
driver, err := qldbdriver.New(
  "vehicle-registration",
  qldbSession,
  func(options *qldbdriver.DriverOptions) {
    options.LoggerVerbosity = qldbdriver.LogInfo
})
if err != nil {
  panic(err)
}

defer driver.Shutdown(context.Background())
```

2.x

```
awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
qldbSession := qldbsession.New(awsSession)

driver, err := qldbdriver.New(
  "vehicle-registration",
  qldbSession,
  func(options *qldbdriver.DriverOptions) {
    options.LoggerVerbosity = qldbdriver.LogInfo
  })
if err != nil {
  panic(err)
}
```

CRUD オペレーション

QLDB は作成、読み取り、更新、および削除 (CRUD) オペレーションをトランザクションの一部として実行します。

> **Warning**
>
> ベストプラクティスとして、書き込みトランザクションを厳密にべき等にしてください。

トランザクションをべき等にする

再試行の場合に予期しない結果を避けるために、書き込みトランザクションをべき等にすることをお勧めします。トランザクションは、複数回実行して毎回同じ結果を生成できる場合、idempotent です。

例えば、Person という名前のテーブルにドキュメントを挿入するトランザクションについて考えてみます。トランザクションは、まずドキュメントがテーブル内に既に存在するかどうかをチェックする必要があります。このチェックを行わないと、テーブルに重複するドキュメントができる可能性があります。

QLDB がサーバー側でトランザクションを正常にコミットできるが、レスポンスを待機中にクライアントはタイムアウトするとします。トランザクションがべき等でない場合、再試行時に同じドキュメントが複数回挿入される可能性があります。

インデックスを使用してテーブル全体のスキャンを回避する

インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`) こともお勧めします。このインデックス付きのルックアップがない場合、QLDB はテーブルスキャンを実行する必要があり、トランザクションタイムアウトやオプティミスティック同時実行制御 (OCC) 競合が発生する可能性があります。

OCC の詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

暗黙的に作成されたトランザクション

QLDBDriver.Execute 関数は、トランザクションのインスタンス (これを使用してステートメントを実行可能) を受け取る Lambda 関数を受け入れます。Transaction のインスタンスは、暗黙的に作成されたトランザクションをラップします。

`Transaction.Execute` 関数を使用して、Lambda 関数内でステートメントを実行できます。ドライバーは、Lambda 関数が戻ったときに暗黙的にトランザクションをコミットします。

次のセクションでは、基本的な CRUD オペレーションの実行、カスタム再試行ロジックの指定、一意性制約の実装方法について説明します。

目次

テーブルの作成

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("CREATE TABLE Person")
})
```

インデックスの作成

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("CREATE INDEX ON Person(GovId)")
})
```

ドキュメントの読み取り

```
var decodedResult map[string]interface{}

// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName": "Brent" }
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  result, err := txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'")
  if err != nil {
    return nil, err
  }
  for result.Next(txn) {
    ionBinary := result.GetCurrentData()
    err = ion.Unmarshal(ionBinary, &decodedResult)
    if err != nil {
```

```
      return nil, err
    }
    fmt.Println(decodedResult) // prints map[GovId: TOYENC486FH FirstName:Brent]
  }
  if result.Err() != nil {
    return nil, result.Err()
  }
  return nil, nil
})
if err != nil {
  panic(err)
}
```

クエリパラメータの使用

次のコード例では、ネイティブ型のクエリパラメータを使用します。

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("SELECT * FROM Person WHERE GovId = ?", "TOYENC486FH")
})
if err != nil {
  panic(err)
}
```

次のコード例では、複数のクエリパラメータを使用します。

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?",
 "TOYENC486FH", "Brent")
})
if err != nil {
  panic(err)
}
```

次のコード例では、クエリパラメータのリストを使用します。

```
govIDs := []string{}{"TOYENC486FH", "ROEE1", "YH844"}

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("SELECT * FROM Person WHERE GovId IN (?,?,?)", govIDs...)
```

```
})
if err != nil {
  panic(err)
}
```

> **Note**
>
> インデックス付きルックアップなしでクエリを実行すると、完全なテーブルスキャンが呼び出されます。この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、クエリのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの挿入

次のコード例では、ネイティブデータ型を挿入します。

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  // Check if a document with a GovId of TOYENC486FH exists
  // This is critical to make this transaction idempotent
  result, err := txn.Execute("SELECT * FROM Person WHERE GovId = ?", "TOYENC486FH")
  if err != nil {
    return nil, err
  }
  // Check if there are any results
  if result.Next(txn) {
    // Document already exists, no need to insert
  } else {
    person := map[string]interface{}{
      "GovId": "TOYENC486FH",
      "FirstName": "Brent",
    }
    _, err = txn.Execute("INSERT INTO Person ?", person)
    if err != nil {
      return nil, err
    }
  }
  return nil, nil
})
```

このトランザクションは、ドキュメントを Person テーブルに挿入します。挿入する前に、まずドキュメントがテーブル内に既に存在しているかどうかをチェックします。このチェックにより、トランザクションは本質的にべき等になります。このトランザクションを複数回実行しても、意図しない副作用は発生しません。

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

1 つのステートメントで複数のドキュメントの挿入

1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 list のパラメータをステートメントに渡すことができます。

```
// people is a list
txn.Execute("INSERT INTO People ?", people)
```

リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ドキュメントの更新

次のコード例では、ネイティブデータ型を使用します。

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", "John",
 "TOYENC486FH")
})
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテン

シーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの削除

次のコード例では、ネイティブデータ型を使用します。

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("DELETE FROM Person WHERE GovId = ?", "TOYENC486FH")
})
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

トランザクションで複数のステートメントの実行

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
 not.
func InsureCar(driver *qldbdriver.QLDBDriver, vin string) (bool, error) {
    insured, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {

        result, err := txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
        if err != nil {
            return false, err
        }

        hasNext := result.Next(txn)
        if !hasNext && result.Err() != nil {
            return false, result.Err()
        }
```

```
        if hasNext {
            _, err = txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
            if err != nil {
                return false, err
            }
            return true, nil
        }
        return false, nil
    })
    if err != nil {
        panic(err)
    }

    return insured.(bool), err
}
```

再試行ロジック

ドライバーの Execute 関数には、再試行可能な例外 (タイムアウトや OCC 競合など) が発生した場合にトランザクションを再試行する組み込みの再試行メカニズムがあります。再試行の最大数とバックオフ戦略を設定できます。

デフォルトの再試行制限は 4 であり、デフォルトのバックオフ戦略は 10 ミリ秒のベースを使用する ExponentialBackoffStrategy です。再試行ポリシーは、RetryPolicy のインスタンスを使用して、ドライバーインスタンスごとおよびトランザクションごとに設定できます。

次のコード例では、ドライバーのインスタンスのカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。

```
import (
  "github.com/aws/aws-sdk-go/aws"
  "github.com/aws/aws-sdk-go/aws/session"
  "github.com/aws/aws-sdk-go/service/qldbsession"
  "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

func main() {
  awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
  qldbSession := qldbsession.New(awsSession)
```

```
  // Configuring retry limit to 2
  retryPolicy := qldbdriver.RetryPolicy{MaxRetryLimit: 2}

  driver, err := qldbdriver.New("test-ledger", qldbSession, func(options
 *qldbdriver.DriverOptions) {
    options.RetryPolicy = retryPolicy
  })
  if err != nil {
    panic(err)
  }

  // Configuring an exponential backoff strategy with base of 20 milliseconds
  retryPolicy = qldbdriver.RetryPolicy{
    MaxRetryLimit: 2,
    Backoff: qldbdriver.ExponentialBackoffStrategy{SleepBase: 20, SleepCap: 4000,
    }}

  driver, err = qldbdriver.New("test-ledger", qldbSession, func(options
 *qldbdriver.DriverOptions) {
    options.RetryPolicy = retryPolicy
  })
  if err != nil {
    panic(err)
  }
}
```

次のコード例では、特定の無名関数のカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。この SetRetryPolicy 関数は、ドライバーインスタンスに設定された再試行ポリシーを上書きします。

```
import (
  "context"
  "github.com/aws/aws-sdk-go/aws"
  "github.com/aws/aws-sdk-go/aws/session"
  "github.com/aws/aws-sdk-go/service/qldbsession"
  "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

func main() {
  awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
  qldbSession := qldbsession.New(awsSession)
```

```
  // Configuring retry limit to 2
  retryPolicy1 := qldbdriver.RetryPolicy{MaxRetryLimit: 2}

  driver, err := qldbdriver.New("test-ledger", qldbSession, func(options
 *qldbdriver.DriverOptions) {
    options.RetryPolicy = retryPolicy1
  })
  if err != nil {
    panic(err)
  }

  // Configuring an exponential backoff strategy with base of 20 milliseconds
  retryPolicy2 := qldbdriver.RetryPolicy{
    MaxRetryLimit: 2,
    Backoff: qldbdriver.ExponentialBackoffStrategy{SleepBase: 20, SleepCap: 4000,
    }}

  // Overrides the retry policy set by the driver instance
  driver.SetRetryPolicy(retryPolicy2)

  driver.Execute(context.Background(), func(txn qldbdriver.Transaction) (interface{},
 error) {
    return txn.Execute("CREATE TABLE Person")
  })
}
```

一意性制約の実装

QLDB はユニークインデックスをサポートしていませんが、この動作をアプリケーションに実装できます。

Person テーブル内の GovId フィールドに対して一意性制約を実装するとします。これを行うには、以下を実行するトランザクションを記述します。

1. テーブルに指定された GovId を持つ既存のドキュメントがないことをアサートします。
2. アサーションに合格した場合は、ドキュメントを挿入します。

競合するトランザクションが同時にアサーションに合格すると、一方のトランザクションだけが正常にコミットされます。もう一方のトランザクションは OCC 競合例外で失敗します。

次のコード例は、この一意性制約ロジックを実装する方法を示しています。

```
govID := "TOYENC486FH"

document := map[string]interface{}{
  "GovId":     "TOYENC486FH",
  "FirstName": "Brent",
}

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  // Check if doc with GovId = govID exists
  result, err := txn.Execute("SELECT * FROM Person WHERE GovId = ?", govID)
  if err != nil {
    return nil, err
  }
  // Check if there are any results
  if result.Next(txn) {
    // Document already exists, no need to insert
    return nil, nil
  }
  return txn.Execute("INSERT INTO Person ?", document)
})
if err != nil {
  panic(err)
}
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

Amazon Ion の操作

以下のセクションでは、Amazon Ion モジュールを使用して Ion データを処理する方法について説明します。

目次

- Ion 型の作成
- Ion バイナリの取得
- Ion テキストの取得

Ion モジュールのインポート

```
import "github.com/amzn/ion-go/ion"
```

Ion 型の作成

Go 用 Ion ライブラリでは現在、ドキュメントオブジェクトモデル (DOM) がサポートされていないため、Ion データ型を作成することはできません。しかし、QLDB の操作時には、Go ネイティブ型と Ion バイナリの間で、マーシャルとアンマーシャルを行えます。

Ion バイナリの取得

```
aDict := map[string]interface{}{
  "GovId": "TOYENC486FH",
  "FirstName": "Brent",
}

ionBytes, err := ion.MarshalBinary(aDict)
if err != nil {
  panic(err)
}

fmt.Println(ionBytes) // prints [224 1 0 234 238 151 129 131 222 147 135 190 144 133 71
 111 118 73 100 137 70 105 114 115 116 78 97 109 101 222 148 138 139 84 79 89 69 78 67
 52 56 54 70 72 139 133 66 114 101 110 116]
```

Ion テキストの取得

```
aDict := map[string]interface{}{
  "GovId": "TOYENC486FH",
  "FirstName": "Brent",
}

ionBytes, err := ion.MarshalText(aDict)
if err != nil {
```

```
  panic(err)
}

fmt.Println(string(ionBytes)) // prints {FirstName:"Brent",GovId:"TOYENC486FH"}
```

Ion の詳細については、GitHub にある Amazon Ion のドキュメントを参照してください。QLDB で Ion を操作するコード例については、「Amazon QLDB で Amazon Ion のデータ型を操作する」を参照してください。

Node.js 用 Amazon QLDB ドライバー

台帳内のデータを操作するには、AWS が提供するドライバーを使用して Node.js アプリケーションから Amazon QLDB に接続します。次のトピックでは、Node.js 用 QLDB ドライバーの使用を開始する方法について説明します。

トピック

- ドライバーに関するリソース
- 前提条件
- インストール
- セットアップの推奨事項
- Node.js 用 Amazon QLDB ドライバー — クイックスタートチュートリアル
- Node.js 用 Amazon QLDB ドライバー — クックブックリファレンス

ドライバーに関するリソース

Node.js ドライバーでサポートされている機能の詳細については、以下のリソースを参照してください。

- API リファレンス: 3.x、2.x、1.x
- ドライバーのソースコード (GitHub)
- サンプルアプリケーションのソースコード (GitHub)
- Amazon Ion コード例
- AWS Lambda を使用して QLDB で単純な CRUD オペレーションとデータストリームを構築する (AWS ブログ)

前提条件

Node.js 用 QLDB ドライバーの使用を開始する前に、次の操作を行う必要があります。

1. 「Amazon QLDB へのアクセス」にある AWS の設定手順に従います。これには以下が含まれます。

   1. AWS にサインアップする。
   2. QLDB の適切なアクセス許可を持つユーザーを作成します。
   3. 開発に必要なプログラムへのアクセスを提供します。

2. Node.js ダウンロードサイトから Node.js バージョン 14.x 以降をインストールします。(以前のバージョンのドライバーは Node.js バージョン 10.x 以降をサポートしています)

3. AWS SDK for JavaScript in Node.js の開発環境を設定します。

   1. AWS 認証情報をセットアップします。共有認証情報ファイルを作成することをお勧めします。

      手順については、「AWS SDK for JavaScript デベロッパーガイド」の「共有認証情報ファイルから Node.js に認証情報をロードする」を参照してください。

   2. デフォルトの AWS リージョン を設定します。この方法については、「AWS リージョン の設定」を参照してください。

      利用可能なリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

次に、すべてのチュートリアル用のサンプルアプリケーションをダウンロードするか、Node.js プロジェクトにのみドライバーをインストールして短いコード例を実行できます。

- 既存のプロジェクトに QLDB ドライバーと AWS SDK for JavaScript in Node.js をインストールするには、「インストール」に進みます。
- プロジェクトを設定し、台帳の基本的なデータトランザクションを示す短いコード例については、「クイックスタートチュートリアル」を参照してください。
- チュートリアルのサンプルアプリケーション全体のデータプレーンと管理 API の両方のオペレーションを実行する詳細な例については、「Node.js チュートリアル」を参照してください。

インストール

QLDB は、以下のドライバーバージョンと Node.js の依存関係をサポートしています。

ドライバーのバージョン	Node.js バージョン	[ステータス]	最新リリース日
1.x	10.x 以降	本番リリース	2020 年 6 月 5 日
2.x	10.x 以降	本番リリース	2021 年 5 月 6 日
3.x	14.x 以降	本番リリース	2023 年 11 月 10 日

npm (Node.js パッケージマネージャー) を使用して QLDB ドライバーをインストールするには、プロジェクトのルートディレクトリから次のコマンドを入力します。

3.x

```
npm install amazon-qldb-driver-nodejs
```

2.x

```
npm install amazon-qldb-driver-nodejs@2.2.0
```

1.x

```
npm install amazon-qldb-driver-nodejs@1.0.0
```

ドライバーには次のパッケージへのピア依存関係があります。これらのパッケージをプロジェクトの依存関係としてインストールする必要もあります。

3.x

モジュール式集約 QLDB クライアント (管理 API)

```
npm install @aws-sdk/client-qldb
```

モジュール式集約 QLDB セッションクライアント (データ API)

```
npm install @aws-sdk/client-qldb-session
```

Amazon Ion データ形式

```
npm install ion-js
```

純粋な JavaScript の実装 `BigInt`

```
npm install jsbi
```

2.x

AWS SDK for JavaScript

```
npm install aws-sdk
```

Amazon Ion データ形式

```
npm install ion-js@4.0.0
```

純粋な JavaScript の実装 `BigInt`

```
npm install jsbi@3.1.1
```

1.x

AWS SDK for JavaScript

```
npm install aws-sdk
```

Amazon Ion データ形式

```
npm install ion-js@4.0.0
```

純粋な JavaScript の実装 `BigInt`

```
npm install jsbi@3.1.1
```

ドライバーを使用して台帳に接続する

その後、ドライバーをインポートし、それを使用して台帳に接続することができます。次の TypeScript コード例は、指定した台帳名と AWS リージョン のドライバーインスタンスを作成する方法を示しています。

3.x

```
import { Agent } from 'https';
import { QLDBSessionClientConfig } from "@aws-sdk/client-qldb-session";
import { QldbDriver, RetryConfig  } from 'amazon-qldb-driver-nodejs';
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";

const maxConcurrentTransactions: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
    httpAgent: new Agent({
      maxSockets: maxConcurrentTransactions
    })
};

const serviceConfigurationOptions: QLDBSessionClientConfig = {
    region: "us-east-1"
};

//Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
const retryConfig: RetryConfig = new RetryConfig(retryLimit);
const qldbDriver: QldbDriver = new QldbDriver("testLedger",
 serviceConfigurationOptions, lowLevelClientHttpOptions, maxConcurrentTransactions,
 retryConfig);

qldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
});
```

2.x

```
import { Agent } from 'https';
import { QldbDriver, RetryConfig  } from 'amazon-qldb-driver-nodejs';
```

```
const maxConcurrentTransactions: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const agentForQldb: Agent = new Agent({
    keepAlive: true,
    maxSockets: maxConcurrentTransactions
});

const serviceConfigurationOptions = {
    region: "us-east-1",
    httpOptions: {
        agent: agentForQldb
    }
};

//Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
const retryConfig: RetryConfig = new RetryConfig(retryLimit);
const qldbDriver: QldbDriver = new QldbDriver("testLedger",
 serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);

qldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
});
```

1.x

```
import { Agent } from 'https';
import { QldbDriver } from 'amazon-qldb-driver-nodejs';

const poolLimit: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const agentForQldb: Agent = new Agent({
    keepAlive: true,
    maxSockets: poolLimit
});

const serviceConfigurationOptions = {
    region: "us-east-1",
    httpOptions: {
```

```
        agent: agentForQldb
    }
};

const qldbDriver: QldbDriver = new QldbDriver("testLedger",
 serviceConfigurationOptions, retryLimit, poolLimit);
qldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
});
```

台帳に対して基本的なデータトランザクションを実行する方法を示す短いコード例については、「クックブックリファレンス」を参照してください。

セットアップの推奨事項

Keep-alive を使用して接続を再利用する

QLDB Node.js ドライバー v3

デフォルトの Node.js HTTP/HTTPS エージェントは新しいリクエストがあるたびに新しい TCP 接続を作成します。新しい接続を確立するコストを回避するため、AWS SDK for JavaScript v3 はデフォルトで TCP 接続を再利用します。詳細および接続の再利用を無効にする方法については、「AWS SDK for JavaScript デベロッパーガイド」の「Node.js で Keep-alive を使用して接続を再利用する」を参照してください。

Node.js 用 QLDB ドライバーで接続を再利用するには、デフォルト設定を使用することをお勧めします。ドライバーの初期化中に、低レベルのクライアント HTTP オプション `maxSockets` を `maxConcurrentTransactions` に設定した値と同じ値に設定します。

例えば、次の JavaScript コードまたはTypeScript コードを参照してください。

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');
const https = require('https');

//Replace this value as appropriate for your application
const maxConcurrentTransactions = 50;

const agentForQldb = new https.Agent({
```

```
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    "maxSockets": maxConcurrentTransactions
});

const lowLevelClientHttpOptions = {
    httpAgent: agentForQldb
}

let driver = new qldb.QldbDriver("testLedger", undefined, lowLevelClientHttpOptions,
 maxConcurrentTransactions);
```

TypeScript

```
import { Agent } from 'https';
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";
import { QldbDriver } from 'amazon-qldb-driver-nodejs';

//Replace this value as appropriate for your application
const maxConcurrentTransactions: number = 50;

const agentForQldb: Agent = new Agent({
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    maxSockets: maxConcurrentTransactions
});

const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
    httpAgent: agentForQldb
};

let driver = new QldbDriver("testLedger", undefined, lowLevelClientHttpOptions,
 maxConcurrentTransactions);
```

QLDB Node.js ドライバー v2

デフォルトの Node.js HTTP/HTTPS エージェントは新しいリクエストがあるたびに新しい TCP 接続を作成します。新しい接続を確立するコストを回避するために、既存の接続の再利用をお勧めします。

Node.js 用 QLDB ドライバーで接続を再利用するには、次のいずれかのオプションを使用します。

- ドライバーの初期化中に、以下の低レベルクライアント HTTP オプションを設定します。
  - `keepAlive` – `true`
  - `maxSockets` – `maxConcurrentTransactions` に設定したのと同じ値

  例えば、次の JavaScript コードまたはTypeScript コードを参照してください。

  JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');
const https = require('https');

//Replace this value as appropriate for your application
const maxConcurrentTransactions = 50;

const agentForQldb = new https.Agent({
    "keepAlive": true,
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    "maxSockets": maxConcurrentTransactions
});

const serviceConfiguration = { "httpOptions": {
    "agent": agentForQldb
}};

let driver = new qldb.QldbDriver("testLedger", serviceConfiguration,
 maxConcurrentTransactions);
```

  TypeScript

```
import { Agent } from 'https';
import { ClientConfiguration } from 'aws-sdk/clients/acm';
import { QldbDriver } from 'amazon-qldb-driver-nodejs';

//Replace this value as appropriate for your application
const maxConcurrentTransactions: number = 50;

const agentForQldb: Agent = new Agent({
    keepAlive: true,
```

```
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    maxSockets: maxConcurrentTransactions
});

const serviceConfiguration: ClientConfiguration = { httpOptions: {
    agent: agentForQldb
}};

let driver = new QldbDriver("testLedger", serviceConfiguration,
 maxConcurrentTransactions);
```

- または、AWS_NODEJS_CONNECTION_REUSE_ENABLED 環境変数を 1 に設定することもできます。詳細については、「AWS SDK for JavaScript デベロッパーガイド」の「Node.js で Keep-alive を使用して接続を再利用する」を参照してください。

> **Note**
>
> この環境変数を設定すると、AWS SDK for JavaScript を使用するすべての AWS のサービス に影響が生じます。

Node.js 用 Amazon QLDB ドライバー — クイックスタートチュートリアル

このセクションでは、Node.js 用 Amazon QLDB ドライバーを使用して簡単なアプリケーションを設定する方法について説明します。このガイドには、ドライバーのインストール手順と、基本的な作成、読み取り、更新、削除 (CRUD) オペレーションを実行する JavaScript と TypeScript の短いコード例が含まれています。これらのオペレーションを完全なサンプルアプリケーションで実行する詳細な例については、「Node.js チュートリアル」を参照してください。

> **Note**
>
> ステップによっては、サポート対象メジャーバージョンの Node.js 用 QLDB ドライバーごとに異なるコード例を示しています (該当する場合)。

トピック

- 前提条件

前提条件

作業を始める前に、次の操作を実行してください。

1. Node.js ドライバー用の「前提条件」を完了します (まだ完了していない場合)。これには、AWS へのサインアップ、開発のためのプログラムによるアクセスの許可、Node.js のインストールが含まれます。
2. `quick-start` という名前の台帳を作成します。

   台帳の作成方法については、「Amazon QLDB 台帳の基本的なオペレーション」、または「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」を参照してください。

TypeScript を使用している場合、次の設定手順も実行する必要があります。

TypeScript の使用

TypeScript をインストールするには

1. TypeScript パッケージをインストールします。QLDB ドライバーは TypeScript 3.8.x で動作します。

   ```
   $ npm install --global typescript@3.8.0
   ```

2. パッケージがインストールされたら、次のコマンドを実行して TypeScript コンパイラがインストールされていることを確認します。

   ```
   $ tsc --version
   ```

次のステップのコードを実行するには、まず TypeScript ファイルを、次のように、実行可能な JavaScript コードにトランスパイルする必要があります。

```
$ tsc app.ts; node app.js
```

ステップ 1: プロジェクトを設定する

まず、Node.js プロジェクトを設定します。

1. アプリケーション用のフォルダを作成します。

   ```
   $ mkdir myproject
   $ cd myproject
   ```

2. プロジェクトを初期化するには、次の npm コマンドを入力し、設定中に表示される質問に回答します。ほとんどの質問にはデフォルトを使用できます。

   ```
   $ npm init
   ```

3. Node.js 用 Amazon QLDB ドライバーをインストールします。
   - バージョン 3.x の使用

     ```
     $ npm install amazon-qldb-driver-nodejs --save
     ```

   - バージョン 2.x の使用

     ```
     $ npm install amazon-qldb-driver-nodejs@2.2.0 --save
     ```

   - バージョン 1.x の使用

     ```
     $ npm install amazon-qldb-driver-nodejs@1.0.0 --save
     ```

4. ドライバーのピア依存関係をインストールします。
   - バージョン 3.x の使用

     ```
     $ npm install @aws-sdk/client-qldb-session --save
     $ npm install ion-js --save
     $ npm install jsbi --save
     ```

   - バージョン 2.x または 1.x の使用

```
$ npm install aws-sdk --save
$ npm install ion-js@4.0.0 --save
$ npm install jsbi@3.1.1 --save
```

5. JavaScript には app.js、TypeScript には app.ts という名前で新しいファイルを作成します。

   次に、以下のステップのコード例を段階的に追加し、いくつかの基本的な CRUD オペレーションを試します。または、ステップバイステップのチュートリアルをスキップして、代わりに完全なアプリケーションを実行することもできます。

ステップ 2: ドライバーを初期化する

quick-start という名前の台帳に接続するドライバーのインスタンスを初期化します。次のコードを app.js または app.ts ファイルに追加します。

バージョン 3.x の使用

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');
var https = require('https');

function main() {
    const maxConcurrentTransactions = 10;
    const retryLimit = 4;

    const agentForQldb = new https.Agent({
        maxSockets: maxConcurrentTransactions
    });

    const lowLevelClientHttpOptions = {
       httpAgent: agentForQldb
    }

    const serviceConfigurationOptions = {
        region: "us-east-1"
    };

    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
```

```
    var retryConfig = new qldb.RetryConfig(retryLimit);
    var driver = new qldb.QldbDriver("quick-start", serviceConfigurationOptions,
 lowlevelClientHttpOptions, maxConcurrentTransactions, retryConfig);
}

main();
```

TypeScript

```
import { Agent } from "https";
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";
import { QLDBSessionClientConfig } from "@aws-sdk/client-qldb-session";
import { QldbDriver, RetryConfig } from "amazon-qldb-driver-nodejs";

function main(): void {
    const maxConcurrentTransactions: number = 10;
    const agentForQldb: Agent = new Agent({
        maxSockets: maxConcurrentTransactions
    });

    const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
      httpAgent: agentForQldb
    };

    const serviceConfigurationOptions: QLDBSessionClientConfig = {
        region: "us-east-1"
    };

    const retryLimit: number = 4;
    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
    const driver: QldbDriver = new QldbDriver("quick-start",
 serviceConfigurationOptions, lowLevelClientHttpOptions, maxConcurrentTransactions,
 retryConfig);
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> - このコード例では、*us-east-1* を、台帳を作成した AWS リージョン に置き換えます。
> - 簡単にするために、このガイドの残りのコード例では、次のバージョン 1.x の例で指定しているように、デフォルト設定のドライバーを使用しています。その代わりに、RetryConfig をカスタマイズすることで、独自のドライバーインスタンスを使用することもできます。

バージョン 2.x の使用

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');
var https = require('https');

function main() {
    var maxConcurrentTransactions = 10;
    var retryLimit = 4;

    var agentForQldb = new https.Agent({
        keepAlive: true,
        maxSockets: maxConcurrentTransactions
    });

    var serviceConfigurationOptions = {
        region: "us-east-1",
        httpOptions: {
            agent: agentForQldb
        }
    };

    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
    var retryConfig = new qldb.RetryConfig(retryLimit);
    var driver = new qldb.QldbDriver("quick-start", serviceConfigurationOptions,
 maxConcurrentTransactions, retryConfig);
}

main();
```

TypeScript

```
import { QldbDriver, RetryConfig } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/acm";
import { Agent } from "https";

function main(): void {
    const maxConcurrentTransactions: number = 10;
    const agentForQldb: Agent = new Agent({
        keepAlive: true,
        maxSockets: maxConcurrentTransactions
    });
    const serviceConfigurationOptions: ClientConfiguration = {
        region: "us-east-1",
        httpOptions: {
            agent: agentForQldb
        }
    };
    const retryLimit: number = 4;
    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
    const driver: QldbDriver = new QldbDriver("quick-start",
 serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);
}

if (require.main === module) {
    main();
}
```

Note

- このコード例では、*us-east-1* を、台帳を作成した AWS リージョン に置き換えます。
- バージョン 2.x には QldbDriver を初期化する RetryConfig という新しいオプションパラメータが導入されています。
- 簡単にするために、このガイドの残りのコード例では、次のバージョン 1.x の例で指定しているように、デフォルト設定のドライバーを使用しています。その代わりに、RetryConfig をカスタマイズすることで、独自のドライバーインスタンスを使用することもできます。

- このコード例では、Keep-Alive オプションを設定して、既存の接続を再利用するドライバーを初期化します。詳細については、Node.js ドライバー向けの「セットアップの推奨事項」を参照してください。

バージョン 1.x の使用

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

function main() {
    // Use default settings
    const driver = new qldb.QldbDriver("quick-start");
}

main();
```

TypeScript

```
import { QldbDriver } from "amazon-qldb-driver-nodejs";

function main(): void {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> AWS_REGION 環境変数を設定して、リージョンを指定できます。詳細については、「AWS SDK for JavaScript デベロッパーガイド」の「AWS リージョン の設定」を参照してください。

ステップ 3: テーブルとインデックスを作成する

次のコード例は、CREATE TABLE ステートメントと CREATE INDEX ステートメントの実行方法を示しています。

1. 次の関数を追加して People という名前のテーブルを作成します。

   JavaScript

   ```
   async function createTable(txn) {
       await txn.execute("CREATE TABLE People");
   }
   ```

   TypeScript

   ```
   async function createTable(txn: TransactionExecutor): Promise<void> {
       await txn.execute("CREATE TABLE People");
   }
   ```

2. People テーブルの firstName フィールドのインデックスを作成する、以下の関数を追加します。クエリのパフォーマンスを最適化し、オプティミスティック同時実行制御 (OCC) 競合例外を制限するために、インデックスが必要です。

   JavaScript

   ```
   async function createIndex(txn) {
       await txn.execute("CREATE INDEX ON People (firstName)");
   }
   ```

   TypeScript

   ```
   async function createIndex(txn: TransactionExecutor): Promise<void> {
       await txn.execute("CREATE INDEX ON People (firstName)");
   }
   ```

3. main 関数で最初に createTable を呼び出し、次に createIndex を呼び出します。

   JavaScript

   ```
   const qldb = require('amazon-qldb-driver-nodejs');
   ```

```
async function main() {
    // Use default settings
    const driver = new qldb.QldbDriver("quick-start");

    await driver.executeLambda(async (txn) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
    });

    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    await driver.executeLambda(async (txn: TransactionExecutor) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
    });

    driver.close();
}

if (require.main === module) {
    main();
}
```

4. コードを実行して、テーブルとインデックスを作成します。

JavaScript

```
$ node app.js
```

TypeScript

```
$ tsc app.ts; node app.js
```

ステップ 4: ドキュメントを挿入する

次のコード例は、INSERT ステートメントの実行方法を示しています。QLDB は PartiQL クエリ言語 (SQL 互換) と Amazon Ion データ形式 (JSON のスーパーセット) をサポートします。

1. 次の関数を追加して People テーブルにドキュメントを挿入します。

JavaScript

```
async function insertDocument(txn) {
    const person = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}
```

TypeScript

```
async function insertDocument(txn: TransactionExecutor): Promise<void> {
    const person: Record<string, any> = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}
```

この例では、疑問符 (?) を変数プレースホルダーとして使用して、ドキュメント情報をステートメントに渡します。execute メソッドは、Amazon Ion 型と Node.js ネイティブ型の両方の値をサポートします。

> **Tip**
>
> 1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 list のパラメータをステートメントに渡すことができます。
>
> ```
> // people is a list
> txn.execute("INSERT INTO People ?", people);
> ```
>
> リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

2. main 関数で、createTable と createIndex の呼び出しを削除し、insertDocument への呼び出しを追加します。

   JavaScript

   ```
   const qldb = require('amazon-qldb-driver-nodejs');

   async function main() {
       // Use default settings
       const driver = new qldb.QldbDriver("quick-start");

       await driver.executeLambda(async (txn) => {
           console.log("Insert document");
           await insertDocument(txn);
       });

       driver.close();
   }

   main();
   ```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    await driver.executeLambda(async (txn: TransactionExecutor) => {
        console.log("Insert document");
        await insertDocument(txn);
    });

    driver.close();
}

if (require.main === module) {
    main();
}
```

ステップ 5: ドキュメントにクエリを実行する

次のコード例は、SELECT ステートメントの実行方法を示しています。

1. 次の関数を追加して People テーブルのドキュメントにクエリを実行します。

JavaScript

```
async function fetchDocuments(txn) {
    return await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John");
}
```

TypeScript

```
async function fetchDocuments(txn: TransactionExecutor): Promise<dom.Value[]> {
    return (await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John")).getResultList();
}
```

2. main 関数で、次の fetchDocuments への呼び出しを、insertDocument への呼び出しの後に追加します。

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

async function main() {
    // Use default settings
    const driver = new qldb.QldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom } from "ion-js";

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        return await fetchDocuments(txn);
    });
```

```
    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

if (require.main === module) {
    main();
}
```

ステップ 6: ドキュメントを更新する

次のコード例は、UPDATE ステートメントの実行方法を示しています。

1. 次の関数を追加し、lastName を "Stiles" に変更して People テーブルのドキュメントを更新します。

   JavaScript

   ```
   async function updateDocuments(txn) {
       await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
    "Stiles", "John");
   }
   ```

   TypeScript

   ```
   async function updateDocuments(txn: TransactionExecutor): Promise<void> {
       await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
    "Stiles", "John");
   }
   ```

2. main 関数で、次の updateDocuments への呼び出しを、fetchDocuments への呼び出しの後に追加します。次に、fetchDocuments を再度呼び出して、更新された結果を確認します。

   JavaScript

   ```
   const qldb = require('amazon-qldb-driver-nodejs');

   async function main() {
       // Use default settings
       const driver = new qldb.QldbDriver("quick-start");
   ```

```
    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom } from "ion-js";

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        return await fetchDocuments(txn);
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
```

```
}

if (require.main === module) {
    main();
}
```

3. コードを実行して、ドキュメントの挿入、クエリ、更新を行います。

JavaScript

```
$ node app.js
```

TypeScript

```
$ tsc app.ts; node app.js
```

完全なアプリケーションの実行

次のコード例は、app.js と app.ts の完全なバージョンを示しています。上のステップを個別に実行する代わりに、このコードを最初から最後まで実行することもできます。このアプリケーションは、quick-start という名前の台帳に対するいくつかの基本的な CRUD オペレーションを実行します。

> **Note**
>
> このコードを実行する前に、quick-start 台帳に People という名前のアクティブなテーブルが存在しないことを確認してください。

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

async function createTable(txn) {
    await txn.execute("CREATE TABLE People");
}

async function createIndex(txn) {
    await txn.execute("CREATE INDEX ON People (firstName)");
}
```

```
async function insertDocument(txn) {
    const person = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}

async function fetchDocuments(txn) {
    return await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John");
}

async function updateDocuments(txn) {
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
 "Stiles", "John");
}

async function main() {
    // Use default settings
    const driver = new qldb.QldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}
```

```
main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom } from "ion-js";

async function createTable(txn: TransactionExecutor): Promise<void> {
    await txn.execute("CREATE TABLE People");
}

async function createIndex(txn: TransactionExecutor): Promise<void> {
    await txn.execute("CREATE INDEX ON People (firstName)");
}

async function insertDocument(txn: TransactionExecutor): Promise<void> {
    const person: Record<string, any> = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}

async function fetchDocuments(txn: TransactionExecutor): Promise<dom.Value[]> {
    return (await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John")).getResultList();
}

async function updateDocuments(txn: TransactionExecutor): Promise<void> {
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
 "Stiles", "John");
};

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor) => {
        console.log("Create table People");
        await createTable(txn);
```

```
        console.log("Create index on firstName");
        await createIndex(txn);
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        return await fetchDocuments(txn);
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

if (require.main === module) {
    main();
}
```

完全なアプリケーションを実行するには、次のコマンドを入力します。

JavaScript

```
$ node app.js
```

TypeScript

```
$ tsc app.ts; node app.js
```

Node.js 用 Amazon QLDB ドライバー — クックブックリファレンス

このリファレンスガイドでは、Node.js 用 Amazon QLDB ドライバーの一般的なユースケースについて説明します。JavaScript と TypeScript のコード例を示して、ドライバーを使用した基本的な作成、読み取り、更新、削除 (CRUD) オペレーションの実行方法を解説します。Amazon Ion データを処理するためのコード例も含まれています。さらに、このガイドでは、トランザクションをべき等にし、一意性の制約を実装するためのベストプラクティスを取り上げています。

目次

ドライバーのインポート

次のコード例では、ドライバーをインポートします。

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');
var ionjs = require('ion-js');
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom, dumpBinary, load } from "ion-js";
```

> **Note**
>
> この例では、Amazon Ion パッケージ (`ion-js`) もインポートします。このリファレンスでいくつかのデータオペレーションを実行するときに Ion データを処理するには、このパッケージが必要です。詳細については、「Amazon Ion の操作」を参照してください。

ドライバーのインスタンス化

次のコード例は、デフォルト設定を使用して指定された台帳名に接続するドライバーのインスタンスを作成します。

JavaScript

```
const qldbDriver = new qldb.QldbDriver("vehicle-registration");
```

TypeScript

```
const qldbDriver: QldbDriver = new QldbDriver("vehicle-registration");
```

CRUD オペレーション

QLDB は作成、読み取り、更新、および削除 (CRUD) オペレーションをトランザクションの一部として実行します。

> **Warning**
>
> ベストプラクティスとして、書き込みトランザクションを厳密にべき等にしてください。

トランザクションをべき等にする

再試行の場合に予期しない結果を避けるために、書き込みトランザクションをべき等にすることをお勧めします。トランザクションは、複数回実行して毎回同じ結果を生成できる場合、idempotent です。

例えば、`Person` という名前のテーブルにドキュメントを挿入するトランザクションについて考えてみます。トランザクションは、まずドキュメントがテーブル内に既に存在するかどうかをチェックす

る必要があります。このチェックを行わないと、テーブルに重複するドキュメントができる可能性があります。

QLDB がサーバー側でトランザクションを正常にコミットできるが、レスポンスを待機中にクライアントはタイムアウトするとします。トランザクションがべき等でない場合、再試行時に同じドキュメントが複数回挿入される可能性があります。

インデックスを使用してテーブル全体のスキャンを回避する

インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する `WHERE` 述語句でステートメントを実行する (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`) こともお勧めします。このインデックス付きのルックアップがない場合、QLDB はテーブルスキャンを実行する必要があり、トランザクションタイムアウトやオプティミスティック同時実行制御 (OCC) 競合が発生する可能性があります。

OCC の詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

暗黙的に作成されたトランザクション

QldbDriver.executeLambda メソッドは TransactionExecutor のインスタンス (これを使用してステートメントを実行可能) を受け取る Lambda 関数を受け入れます。`TransactionExecutor` のインスタンスは、暗黙的に作成されたトランザクションをラップします。

Lambda 関数内でステートメントを実行するには、トランザクションエグゼキューターの execute メソッドを使用します。ドライバーは、Lambda 関数が戻ったときに暗黙的にトランザクションをコミットします。

> **Note**
>
> `execute` メソッドは、Amazon Ion 型と Node.js ネイティブ型の両方をサポートします。Node.js ネイティブ型を引数として `execute` に渡すと、ドライバーが `ion-js` パッケージを使用して Ion 型に変換します (指定された Node.js データ型の変換がサポートされている場合)。サポートされているデータ型と変換ルールについては、Ion JavaScript DOM の「README」を参照してください。

次のセクションでは、基本的な CRUD オペレーションの実行、カスタム再試行ロジックの指定、一意性制約の実装方法について説明します。

目次

テーブルの作成

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute("CREATE TABLE Person");
    });
})();
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('CREATE TABLE Person');
    });
}());
```

インデックスの作成

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
```

```
        await txn.execute("CREATE INDEX ON Person (GovId)");
    });
})();
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('CREATE INDEX ON Person (GovId)');
    });
}());
```

ドキュメントの読み取り

JavaScript

```
(async function() {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn) => {
        const results = (await txn.execute("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'")).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute("SELECT * FROM Person WHERE
 GovId = 'TOYENC486FH'")).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
```

```
}());
```

クエリパラメータの使用

次のコード例では、ネイティブ型のクエリパラメータを使用します。

JavaScript

```
(async function() {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn) => {
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 'TOYENC486FH')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', 'TOYENC486FH')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

次のコード例では、Ion 型のクエリパラメータを使用します。

JavaScript

```
(async function() {
```

```
    await qldbDriver.executeLambda(async (txn) => {
        const govId = ionjs.load("TOYENC486FH");

        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 govId)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govId: dom.Value = load("TOYENC486FH");

        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', govId)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

次のコード例では、複数のクエリパラメータを使用します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ? AND
 FirstName = ?', 'TOYENC486FH', 'Brent')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ? AND FirstName = ?', 'TOYENC486FH', 'Brent')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

次のコード例では、クエリパラメータのリストを使用します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const govIds = ['TOYENC486FH','LOGANB486CG','LEWISR261LL'];
        /*
        Assumes that Person table has documents as follows:
        { "GovId": "TOYENC486FH", "FirstName": "Brent" }
        { "GovId": "LOGANB486CG", "FirstName": "Brent" }
        { "GovId": "LEWISR261LL", "FirstName": "Raul" }
        */
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId IN
 (?,?,?)', ...govIds)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId'));
            console.log(result.get('FirstName'));
            /*
            prints:
            [String: 'TOYENC486FH']
            [String: 'Brent']
            [String: 'LOGANB486CG']
            [String: 'Brent']
            [String: 'LEWISR261LL']
            [String: 'Raul']
            */
        }
    });
```

```
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govIds: string[] = ['TOYENC486FH','LOGANB486CG','LEWISR261LL'];
        /*
        Assumes that Person table has documents as follows:
        { "GovId": "TOYENC486FH", "FirstName": "Brent" }
        { "GovId": "LOGANB486CG", "FirstName": "Brent" }
        { "GovId": "LEWISR261LL", "FirstName": "Raul" }
        */
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId IN (?,?,?)', ...govIds)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId'));
            console.log(result.get('FirstName'));
            /*
            prints:
            [String: 'TOYENC486FH']
            [String: 'Brent']
            [String: 'LOGANB486CG']
            [String: 'Brent']
            [String: 'LEWISR261LL']
            [String: 'Raul']
            */
        }
    });
}());
```

> **Note**
>
> インデックス付きルックアップなしでクエリを実行すると、完全なテーブルスキャンが呼び出されます。この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、クエリのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの挿入

次のコード例では、ネイティブデータ型を挿入します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            await txn.execute('INSERT INTO Person ?', doc);
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc: Record<string, string> = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            await txn.execute('INSERT INTO Person ?', doc);
        }
    });
}());
```

次のコード例では、Ion データ型を挿入します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            // Create a sample Ion doc
            const ionDoc = ionjs.load(ionjs.dumpBinary(doc));

            await txn.execute('INSERT INTO Person ?', ionDoc);
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc: Record<string, string> = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            // Create a sample Ion doc
            const ionDoc: dom.Value = load(dumpBinary(doc));

            await txn.execute('INSERT INTO Person ?', ionDoc);
        }
```

```
    });
}());
```

このトランザクションは、ドキュメントを Person テーブルに挿入します。挿入する前に、まずドキュメントがテーブル内に既に存在しているかどうかをチェックします。このチェックにより、トランザクションは本質的にべき等になります。このトランザクションを複数回実行しても、意図しない副作用は発生しません。

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

1 つのステートメントで複数のドキュメントの挿入

1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 list のパラメータをステートメントに渡すことができます。

```
// people is a list
txn.execute("INSERT INTO People ?", people);
```

リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ドキュメントの更新

次のコード例では、ネイティブデータ型を使用します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?', 'John',
 'TOYENC486FH');
    });
```

```
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?', 'John',
 'TOYENC486FH');
    });
}());
```

次のコード例では、Ion データ型を使用します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const firstName = ionjs.load("John");
        const govId = ionjs.load("TOYENC486FH");

        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?',
 firstName, govId);
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const firstName: dom.Value = load("John");
        const govId: dom.Value = load("TOYENC486FH");

        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?',
 firstName, govId);
    });
}());
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの削除

次のコード例では、ネイティブデータ型を使用します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute('DELETE FROM Person WHERE GovId = ?', 'TOYENC486FH');
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('DELETE FROM Person WHERE GovId = ?', 'TOYENC486FH');
    });
}());
```

次のコード例では、Ion データ型を使用します。

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const govId = ionjs.load("TOYENC486FH");

        await txn.execute('DELETE FROM Person WHERE GovId = ?', govId);
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govId: dom.Value = load("TOYENC486FH");

        await txn.execute('DELETE FROM Person WHERE GovId = ?', govId);
    });
}());
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

トランザクションで複数のステートメントの実行

TypeScript

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
async function insureCar(driver: QldbDriver, vin: string): Promise<boolean> {

    return await driver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 vin)).getResultList();

        if (results.length > 0) {
            await txn.execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin);
            return true;
        }
        return false;
    });
```

```
};
```

再試行ロジック

ドライバーの executeLambda メソッドには、再試行可能な例外 (タイムアウトや OCC 競合など) が発生した場合にトランザクションを再試行する組み込みの再試行メカニズムがあります。再試行の最大数とバックオフ戦略を設定できます。

デフォルトの再試行制限は 4 であり、デフォルトのバックオフ戦略は 10 ミリ秒のベースを使用する defaultBackoffFunction です。再試行設定は、RetryConfig のインスタンスを使用して、ドライバーインスタンスごとおよびトランザクションごとに設定できます。

次のコード例では、ドライバーのインスタンスのカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');

// Configuring retry limit to 2
const retryConfig = new qldb.RetryConfig(2);
const qldbDriver = new qldb.QldbDriver("test-ledger", undefined, undefined,
 retryConfig);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff = (retryAttempt, error, transactionId) => {
    return 1000 * retryAttempt;
};

const retryConfigCustomBackoff = new qldb.RetryConfig(2, customBackoff);
const qldbDriverCustomBackoff = new qldb.QldbDriver("test-ledger", undefined,
 undefined, retryConfigCustomBackoff);
```

TypeScript

```
import { BackoffFunction, QldbDriver, RetryConfig } from "amazon-qldb-driver-nodejs"

// Configuring retry limit to 2
const retryConfig: RetryConfig = new RetryConfig(2);
const qldbDriver: QldbDriver = new QldbDriver("test-ledger", undefined, undefined,
 retryConfig);
```

```
// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff: BackoffFunction = (retryAttempt: number, error: Error,
 transactionId: string) => {
    return 1000 * retryAttempt;
};

const retryConfigCustomBackoff: RetryConfig = new RetryConfig(2, customBackoff);
const qldbDriverCustomBackoff: QldbDriver = new QldbDriver("test-ledger", undefined,
 undefined, retryConfigCustomBackoff);
```

次のコード例では、特定の Lambda 実行のカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。executeLambda のこの設定は、ドライバーインスタンスに設定された再試行ロジックを上書きします。

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');

// Configuring retry limit to 2
const retryConfig1 = new qldb.RetryConfig(2);
const qldbDriver = new qldb.QldbDriver("test-ledger", undefined, undefined,
 retryConfig1);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff = (retryAttempt, error, transactionId) => {
    return 1000 * retryAttempt;
};

const retryConfig2 = new qldb.RetryConfig(2, customBackoff);

// The config `retryConfig1` will be overridden by `retryConfig2`
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute('CREATE TABLE Person');
    }, retryConfig2);
}());
```

TypeScript

```
import { BackoffFunction, QldbDriver, RetryConfig, TransactionExecutor } from
 "amazon-qldb-driver-nodejs"
```

```
// Configuring retry limit to 2
const retryConfig1: RetryConfig = new RetryConfig(2);
const qldbDriver: QldbDriver = new QldbDriver("test-ledger", undefined, undefined,
 retryConfig1);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff: BackoffFunction = (retryAttempt: number, error: Error,
 transactionId: string) => {
    return 1000 * retryAttempt;
};

const retryConfig2: RetryConfig = new RetryConfig(2, customBackoff);

// The config `retryConfig1` will be overridden by `retryConfig2`
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('CREATE TABLE Person');
    }, retryConfig2);
}());
```

一意性制約の実装

QLDB はユニークインデックスをサポートしていませんが、この動作をアプリケーションに実装できます。

Person テーブル内の GovId フィールドに対して一意性制約を実装するとします。これを行うには、以下を実行するトランザクションを記述します。

1. テーブルに指定された GovId を持つ既存のドキュメントがないことをアサートします。
2. アサーションに合格した場合は、ドキュメントを挿入します。

競合するトランザクションが同時にアサーションに合格すると、一方のトランザクションだけが正常にコミットされます。もう一方のトランザクションは OCC 競合例外で失敗します。

次のコード例は、この一意性制約ロジックを実装する方法を示しています。

JavaScript

```
const govId = 'TOYENC486FH';
const document = {
```

```
    'FirstName': 'Brent',
    'GovId': 'TOYENC486FH',
};
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId = govId exists
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 govId)).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            await txn.execute('INSERT INTO Person ?', document);
        }
    });
})();
```

TypeScript

```
const govId: string = 'TOYENC486FH';
const document: Record<string, string> = {
    'FirstName': 'Brent',
    'GovId': 'TOYENC486FH',
};
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        // Check if doc with GovId = govId exists
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', govId)).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            await txn.execute('INSERT INTO Person ?', document);
        }
    });
})();
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

Amazon Ion の操作

以下のセクションでは、Amazon Ion モジュールを使用して Ion データを処理する方法について説明します。

目次

Ion モジュールのインポート

JavaScript

```
var ionjs = require('ion-js');
```

TypeScript

```
import { dom, dumpBinary, dumpText, load } from "ion-js";
```

Ion 型の作成

次のコード例では、Ion テキストから Ion オブジェクトを作成します。

JavaScript

```
const ionText  = '{GovId: "TOYENC486FH", FirstName: "Brent"}';
const ionObj = ionjs.load(ionText);

console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

TypeScript

```
const ionText: string = '{GovId: "TOYENC486FH", FirstName: "Brent"}';
const ionObj: dom.Value = load(ionText);
```

```
console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

次のコード例では、Node.js ディクショナリから Ion オブジェクトを作成します。

JavaScript

```
const aDict = {
    'GovId': 'TOYENC486FH',
    'FirstName': 'Brent'
};
const ionObj = ionjs.load(ionjs.dumpBinary(aDict));
console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

TypeScript

```
const aDict: Record<string, string> = {
    'GovId': 'TOYENC486FH',
    'FirstName': 'Brent'
};
const ionObj: dom.Value = load(dumpBinary(aDict));
console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

Ion バイナリダンプの取得

JavaScript

```
// ionObj is an Ion struct
console.log(ionjs.dumpBinary(ionObj).toString()); // prints
 224,1,0,234,238,151,129,131,222,147,135,190,144,133,71,111,118,73,100,137,70,105,114,115,11
```

TypeScript

```
// ionObj is an Ion struct
console.log(dumpBinary(ionObj).toString()); // prints
 224,1,0,234,238,151,129,131,222,147,135,190,144,133,71,111,118,73,100,137,70,105,114,115,11
```

Ion テキストダンプの取得

JavaScript

```
// ionObj is an Ion struct
console.log(ionjs.dumpText(ionObj)); // prints
 {GovId:"TOYENC486FH",FirstName:"Brent"}
```

TypeScript

```
// ionObj is an Ion struct
console.log(dumpText(ionObj)); // prints {GovId:"TOYENC486FH",FirstName:"Brent"}
```

Ion の詳細については、GitHub にある Amazon Ion のドキュメントを参照してください。QLDB で Ion を操作するコード例については、「Amazon QLDB で Amazon Ion のデータ型を操作する」を参照してください。

Python 用 Amazon QLDB ドライバー

台帳内のデータを操作するには、AWS が提供するドライバーを使用して Python アプリケーションから Amazon QLDB に接続します。次のトピックでは、Python 用 QLDB ドライバーの使用を開始する方法について説明します。

トピック

- ドライバーに関するリソース
- 前提条件
- インストール
- Python 用 Amazon QLDB ドライバー — クイックスタートチュートリアル
- Python 用 Amazon QLDB ドライバー — クックブックリファレンス

ドライバーに関するリソース

Python ドライバーでサポートされている機能の詳細については、以下のリソースを参照してください。

- API リファレンス: 3.x、2.x
- ドライバーのソースコード (GitHub)

- サンプルアプリケーションのソースコード (GitHub)
- Amazon Ion コード例

前提条件

Python 用 QLDB ドライバーの使用を開始する前に、次のことを行う必要があります。

1. 「Amazon QLDB へのアクセス」にある AWS の設定手順に従います。これには以下が含まれます。

   1. AWS にサインアップする。
   2. QLDB の適切なアクセス許可を持つユーザーを作成します。
   3. 開発に必要なプログラムへのアクセスを提供します。
2. Python ダウンロードサイトから次のバージョンの Python のいずれかをインストールします。

   - 3.6 またはそれ以降 – Python v3 用の QLDB ドライバー
   - 3.4 またはそれ以降 – Python v2 用の QLDB ドライバー
3. AWS の認証情報とデフォルトの AWS リージョン を設定します。手順については、AWS SDK for Python (Boto3) ドキュメントの「クイックスタート」を参照してください。

   利用可能なリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

次に、すべてのチュートリアル用のサンプルアプリケーションをダウンロードするか、Python プロジェクトにのみドライバーをインストールして短いコード例を実行できます。

- 既存のプロジェクトに QLDB ドライバーと AWS SDK for Python (Boto3) をインストールするには、「インストール」に進みます。
- プロジェクトを設定し、台帳の基本的なデータトランザクションを示す短いコード例については、「クイックスタートチュートリアル」を参照してください。
- チュートリアルのサンプルアプリケーション全体のデータプレーンと管理 API の両方のオペレーションを実行する詳細な例については、「Python チュートリアル」を参照してください。

インストール

QLDB は、以下のドライバーバージョンと Python の依存関係をサポートしています。

ドライバーのバージョン	Python バージョン	ステータス	最新リリース日
2.x	3.4 以降	本番リリース	2020 年 5 月 7 日
3.x	3.6 以降	本番リリース	2021 年 10 月 28 日

pip (Python のパッケージマネージャー) を使用して PyPI から QLDB ドライバーをインストールするには、コマンドラインで次のように入力します。

3.x

```
pip install pyqldb
```

2.x

```
pip install pyqldb==2.0.2
```

ドライバーをインストールすると、AWS SDK for Python (Boto3) や Amazon Ion パッケージなどの依存関係もインストールされます。

ドライバーを使用して台帳に接続する

その後、ドライバーをインポートし、それを使用して台帳に接続することができます。次の Python コード例は、指定した台帳名のセッションを作成する方法を示しています。

3.x

```
from pyqldb.driver.qldb_driver import QldbDriver
qldb_driver = QldbDriver(ledger_name='testLedger')

for table in qldb_driver.list_tables():
    print(table)
```

2.x

```
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
```

```
qldb_driver = PooledQldbDriver(ledger_name='testLedger')
qldb_session = qldb_driver.get_session()

for table in qldb_session.list_tables():
    print(table)
```

台帳に対して基本的なデータトランザクションを実行する方法を示す短いコード例については、「クックブックリファレンス」を参照してください。

Python 用 Amazon QLDB ドライバー — クイックスタートチュートリアル

このチュートリアルでは、Python 用 Amazon QLDB ドライバーの最新バージョンを使用して単純なアプリケーションを設定する方法について説明します。このガイドには、ドライバのインストール手順と、作成、読み取り、更新、削除 (CRUD) の基本的なオペレーションを実行する短いコード例が含まれています。これらのオペレーションを完全なサンプルアプリケーションで実行する詳細な例については、「Python チュートリアル」を参照してください。

トピック

前提条件

作業を始める前に、次の操作を実行してください。

1. Python ドライバー用の「前提条件」を完了します (まだ完了していない場合)。これには、AWS へのサインアップ、開発のためのプログラムによるアクセスの許可、Python バージョン 3.6 以降のインストールが含まれます。
2. `quick-start` という名前の台帳を作成します。

台帳の作成方法については、「Amazon QLDB 台帳の基本的なオペレーション」、または「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」を参照してください。

ステップ 1: プロジェクトを設定する

まず、Python プロジェクトをセットアップします。

> **Note**
>
> これらのセットアップ手順を自動化する機能を備えた IDE を使用する場合は、「ステップ 2: ドライバーを初期化する」までスキップできます。

1. アプリケーション用のフォルダを作成します。

```
$ mkdir myproject
$ cd myproject
```

2. PyPI から Python 用 QLDB ドライバーをインストールするには、以下の `pip` コマンドを入力します。

```
$ pip install pyqldb
```

ドライバーをインストールすると、AWS SDK for Python (Boto3) や Amazon Ion パッケージなどの依存関係もインストールされます。

3. `app.py` という名前の新しいファイルを作成します。

次に、以下のステップのコード例を段階的に追加し、いくつかの基本的な CRUD オペレーションを試します。または、ステップバイステップのチュートリアルをスキップして、代わりに完全なアプリケーションを実行することもできます。

ステップ 2: ドライバーを初期化する

`quick-start` という名前の台帳に接続するドライバーのインスタンスを初期化します。以下のコードを `app.py` ファイルに追加します。

```
from pyqldb.config.retry_config import RetryConfig
```

```
from pyqldb.driver.qldb_driver import QldbDriver

# Configure retry limit to 3
retry_config = RetryConfig(retry_limit=3)

# Initialize the driver
print("Initializing the driver")
qldb_driver = QldbDriver("quick-start", retry_config=retry_config)
```

ステップ 3: テーブルとインデックスを作成する

以下のコード例は、CREATE TABLE および CREATE INDEX ステートメントの実行方法を示しています。

以下のコードを追加して、People という名前のテーブルと、そのテーブルの lastName フィールドのインデックスを作成します。クエリのパフォーマンスを最適化し、オプティミスティック同時実行制御 (OCC) 競合例外を制限するために、インデックスが必要です。

```
def create_table(transaction_executor):
    print("Creating a table")
    transaction_executor.execute_statement("Create TABLE People")

def create_index(transaction_executor):
    print("Creating an index")
    transaction_executor.execute_statement("CREATE INDEX ON People(lastName)")

# Create a table
qldb_driver.execute_lambda(lambda executor: create_table(executor))

# Create an index on the table
qldb_driver.execute_lambda(lambda executor: create_index(executor))
```

ステップ 4: ドキュメントを挿入する

次のコード例は、INSERT ステートメントの実行方法を示しています。QLDB は PartiQL クエリ言語 (SQL 互換) と Amazon Ion データ形式 (JSON のスーパーセット) をサポートします。

People テーブルにドキュメントを挿入する以下のコードを追加します。

```
def insert_documents(transaction_executor, arg_1):
    print("Inserting a document")
```

```
    transaction_executor.execute_statement("INSERT INTO People ?", arg_1)

# Insert a document
doc_1 = { 'firstName': "John",
          'lastName': "Doe",
          'age': 32,
        }

qldb_driver.execute_lambda(lambda x: insert_documents(x, doc_1))
```

この例では、疑問符 (?) を変数プレースホルダーとして使用して、ドキュメント情報をステートメントに渡します。`execute_statement` メソッドは、Amazon Ion 型と Python ネイティブ型の両方の値をサポートします。

> **Tip**
>
> 1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 list のパラメータをステートメントに渡すことができます。
>
> ```
> # people is a list
> transaction_executor.execute_statement("INSERT INTO Person ?", people)
> ```
>
> リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ステップ 5: ドキュメントにクエリを実行する

次のコード例は、SELECT ステートメントの実行方法を示しています。

People テーブルからドキュメントをクエリする以下のコードを追加します。

```
def read_documents(transaction_executor):
    print("Querying the table")
    cursor = transaction_executor.execute_statement("SELECT * FROM People WHERE
 lastName = ?", 'Doe')

    for doc in cursor:
        print(doc["firstName"])
```

```
        print(doc["lastName"])
        print(doc["age"])

# Query the table
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

ステップ 6: ドキュメントを更新する

次のコード例は、UPDATE ステートメントの実行方法を示しています。

1. age を 42 に更新して People テーブルのドキュメントを更新する以下のコードを追加します。

```
def update_documents(transaction_executor, age, lastName):
    print("Updating the document")
    transaction_executor.execute_statement("UPDATE People SET age = ? WHERE
 lastName = ?", age, lastName)

# Update the document
age = 42
lastName = 'Doe'

qldb_driver.execute_lambda(lambda x: update_documents(x, age, lastName))
```

2. テーブルを再度クエリして、更新された値を確認します。

```
# Query the updated document
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

3. アプリケーションを実行するには、プロジェクトディレクトリから以下のコマンドを入力します。

```
$ python app.py
```

完全なアプリケーションの実行

以下のコード例は、app.py アプリケーションの完全なバージョンです。上のステップを個別に実行する代わりに、このコード例を最初から最後までコピーして実行することもできます。このアプリケーションは、quick-start という名前の台帳に対するいくつかの基本的な CRUD オペレーションを実行します。

> **Note**
>
> このコードを実行する前に、quick-start 台帳に People という名前のアクティブなテーブルが存在しないことを確認してください。

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

def create_table(transaction_executor):
    print("Creating a table")
    transaction_executor.execute_statement("CREATE TABLE People")

def create_index(transaction_executor):
    print("Creating an index")
    transaction_executor.execute_statement("CREATE INDEX ON People(lastName)")

def insert_documents(transaction_executor, arg_1):
    print("Inserting a document")
    transaction_executor.execute_statement("INSERT INTO People ?", arg_1)

def read_documents(transaction_executor):
    print("Querying the table")
    cursor = transaction_executor.execute_statement("SELECT * FROM People WHERE
 lastName = ?", 'Doe')


    for doc in cursor:
        print(doc["firstName"])
        print(doc["lastName"])
        print(doc["age"])

def update_documents(transaction_executor, age, lastName):
    print("Updating the document")
    transaction_executor.execute_statement("UPDATE People SET age = ? WHERE lastName
 = ?", age, lastName)

# Configure retry limit to 3
retry_config = RetryConfig(retry_limit=3)

# Initialize the driver
print("Initializing the driver")
```

```
qldb_driver = QldbDriver("quick-start", retry_config=retry_config)

# Create a table
qldb_driver.execute_lambda(lambda executor: create_table(executor))

# Create an index on the table
qldb_driver.execute_lambda(lambda executor: create_index(executor))

# Insert a document
doc_1 = { 'firstName': "John",
          'lastName': "Doe",
          'age': 32,
        }

qldb_driver.execute_lambda(lambda x: insert_documents(x, doc_1))

# Query the table
qldb_driver.execute_lambda(lambda executor: read_documents(executor))

# Update the document
age = 42
lastName = 'Doe'

qldb_driver.execute_lambda(lambda x: update_documents(x, age, lastName))

# Query the table for the updated document
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

完全なアプリケーションを実行するには、プロジェクトディレクトリから以下のコマンドを入力します。

```
$ python app.py
```

Python 用 Amazon QLDB ドライバー — クックブックリファレンス

このリファレンスガイドでは、Python 用 Amazon QLDB ドライバーの一般的なユースケースについて説明します。Python コード例を提供し、ドライバーを使用して作成、読み取り、更新、および削除 (CRUD) の基本的なオペレーションを実行する方法を説明しています。Amazon Ion データを処理するためのコード例も含まれています。さらに、このガイドでは、トランザクションをべき等にし、一意性の制約を実装するためのベストプラクティスを取り上げています。

> **Note**
>
> 該当する場合、一部のユースケースでは、サポートされているメジャーバージョンの Python 用 QLDB ドライバーに対して異なるコード例があります。

目次

ドライバーのインポート

次のコード例では、ドライバーをインポートします。

3.x

```
from pyqldb.driver.qldb_driver import QldbDriver
import amazon.ion.simpleion as simpleion
```

2.x

```
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
import amazon.ion.simpleion as simpleion
```

> **Note**
>
> この例では、Amazon Ion パッケージ (`amazon.ion.simpleion`) もインポートします。このリファレンスでいくつかのデータオペレーションを実行するときに Ion データを処理するには、このパッケージが必要です。詳細については、「Amazon Ion の操作」を参照してください。

ドライバーのインスタンス化

次のコード例は、デフォルト設定を使用して指定された台帳名に接続するドライバーのインスタンスを作成します。

3.x

```
qldb_driver = QldbDriver(ledger_name='vehicle-registration')
```

2.x

```
qldb_driver = PooledQldbDriver(ledger_name='vehicle-registration')
```

CRUD オペレーション

QLDB は作成、読み取り、更新、および削除 (CRUD) オペレーションをトランザクションの一部として実行します。

> **Warning**
>
> ベストプラクティスとして、書き込みトランザクションを厳密にべき等にしてください。

トランザクションをべき等にする

再試行の場合に予期しない結果を避けるために、書き込みトランザクションをべき等にすることをお勧めします。トランザクションは、複数回実行して毎回同じ結果を生成できる場合、idempotent です。

例えば、`Person` という名前のテーブルにドキュメントを挿入するトランザクションについて考えてみます。トランザクションは、まずドキュメントがテーブル内に既に存在するかどうかをチェックする必要があります。このチェックを行わないと、テーブルに重複するドキュメントができる可能性があります。

QLDB がサーバー側でトランザクションを正常にコミットできるが、レスポンスを待機中にクライアントはタイムアウトするとします。トランザクションがべき等でない場合、再試行時に同じドキュメントが複数回挿入される可能性があります。

インデックスを使用してテーブル全体のスキャンを回避する

インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する `WHERE` 述語句でステートメントを実行する (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`) こともお勧めします。このインデックス付きのルックアップがない場合、QLDB はテーブルスキャンを実行する必要があり、トランザクションタイムアウトやオプティミスティック同時実行制御 (OCC) 競合が発生する可能性があります。

OCC の詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

暗黙的に作成されたトランザクション

pyqldb.driver.qldb_driver.execute_lambda メソッドは、pyqldb.execution.executor.Executor のインスタンスを受け取る Lambda 関数を受け入れます。これを使用してステートメントを実行できます。`Executor` のインスタンスは、暗黙的に作成されたトランザクションをラップします。

Lambda 関数内でステートメントを実行するには、トランザクションエグゼキューターの execute statement メソッドを使用します。ドライバーは、Lambda 関数が戻ったときに暗黙的にトランザクションをコミットします。

> **Note**
>
> `execute_statement` メソッドは、Amazon Ion 型と Python ネイティブ型の両方をサポートします。Python ネイティブ型を引数として `execute_statement` に渡すと、ドライバーが `amazon.ion.simpleion` モジュールを使用して Ion 型に変換します (指定された Python データ型の変換がサポートされている場合)。サポートされているデータ型と変換ルールについては、simpleion のソースコードを参照してください。

次のセクションでは、基本的な CRUD オペレーションの実行、カスタム再試行ロジックの指定、一意性制約の実装方法について説明します。

目次

テーブルの作成

```
def create_table(transaction_executor):
    transaction_executor.execute_statement("CREATE TABLE Person")

qldb_driver.execute_lambda(lambda executor: create_table(executor))
```

インデックスの作成

```
def create_index(transaction_executor):
    transaction_executor.execute_statement("CREATE INDEX ON Person(GovId)")
```

```
qldb_driver.execute_lambda(lambda executor: create_index(executor))
```

ドキュメントの読み取り

```
# Assumes that Person table has documents as follows:
# { "GovId": "TOYENC486FH", "FirstName": "Brent" }

def read_documents(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'")

    for doc in cursor:
        print(doc["GovId"]) # prints TOYENC486FH
        print(doc["FirstName"]) # prints Brent

qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

クエリパラメータの使用

次のコード例では、ネイティブ型のクエリパラメータを使用します。

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?",
 'TOYENC486FH')
```

次のコード例では、Ion 型のクエリパラメータを使用します。

```
name = ion.loads('Brent')
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE FirstName
 = ?", name)
```

次のコード例では、複数のクエリパラメータを使用します。

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?
 AND FirstName = ?", 'TOYENC486FH', "Brent")
```

次のコード例では、クエリパラメータのリストを使用します。

```
gov_ids = ['TOYENC486FH','ROEE1','YH844']
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId IN
 (?,?,?)", *gov_ids)
```

> **Note**
>
> インデックス付きルックアップなしでクエリを実行すると、完全なテーブルスキャンが呼び出されます。この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、クエリのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの挿入

次のコード例では、ネイティブデータ型を挿入します。

```
def insert_documents(transaction_executor, arg_1):
    # Check if doc with GovId:TOYENC486FH exists
    # This is critical to make this transaction idempotent
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
 = ?", 'TOYENC486FH')
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", arg_1)

doc_1 = { 'FirstName': "Brent",
          'GovId': 'TOYENC486FH',
        }

qldb_driver.execute_lambda(lambda executor: insert_documents(executor, doc_1))
```

次のコード例では、Ion データ型を挿入します。

```
def insert_documents(transaction_executor, arg_1):
    # Check if doc with GovId:TOYENC486FH exists
    # This is critical to make this transaction idempotent
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
 = ?", 'TOYENC486FH')
    # Check if there is any record in the cursor
```

```
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", arg_1)

doc_1 = { 'FirstName': 'Brent',
          'GovId': 'TOYENC486FH',
        }

# create a sample Ion doc
ion_doc_1 = simpleion.loads(simpleion.dumps(doc_1))

qldb_driver.execute_lambda(lambda executor: insert_documents(executor, ion_doc_1))
```

このトランザクションは、ドキュメントを Person テーブルに挿入します。挿入する前に、まずドキュメントがテーブル内に既に存在しているかどうかをチェックします。このチェックにより、トランザクションは本質的にべき等になります。このトランザクションを複数回実行しても、意図しない副作用は発生しません。

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

1 つのステートメントで複数のドキュメントの挿入

1 つの INSERT ステートメントを使用して複数のドキュメントを挿入するために、次のように型 list のパラメータをステートメントに渡すことができます。

```
# people is a list
transaction_executor.execute_statement("INSERT INTO Person ?", people)
```

リストを渡すときには、変数プレースホルダー (?) を二重山括弧 (<<...>>) で囲まないでください。マニュアルの PartiQL ステートメントでは、二重山括弧はバッグと呼ばれる順序付けされていないコレクションを表します。

ドキュメントの更新

次のコード例では、ネイティブデータ型を使用します。

```
def update_documents(transaction_executor, gov_id, name):
    transaction_executor.execute_statement("UPDATE Person SET FirstName = ?  WHERE
 GovId = ?", name, gov_id)

gov_id = 'TOYENC486FH'
name = 'John'

qldb_driver.execute_lambda(lambda executor: update_documents(executor, gov_id, name))
```

次のコード例では、Ion データ型を使用します。

```
def update_documents(transaction_executor, gov_id, name):
    transaction_executor.execute_statement("UPDATE Person SET FirstName = ? WHERE GovId
 = ?", name, gov_id)

# Ion datatypes
gov_id = simpleion.loads('TOYENC486FH')
name = simpleion.loads('John')

qldb_driver.execute_lambda(lambda executor: update_documents(executor, gov_id, name))
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

ドキュメントの削除

次のコード例では、ネイティブデータ型を使用します。

```
def delete_documents(transaction_executor, gov_id):
    cursor = transaction_executor.execute_statement("DELETE FROM Person WHERE GovId
 = ?", gov_id)
```

```
gov_id = 'TOYENC486FH'

qldb_driver.execute_lambda(lambda executor: delete_documents(executor, gov_id))
```

次のコード例では、Ion データ型を使用します。

```
def delete_documents(transaction_executor, gov_id):
    cursor = transaction_executor.execute_statement("DELETE FROM Person WHERE GovId
 = ?", gov_id)

# Ion datatypes
gov_id = simpleion.loads('TOYENC486FH')

qldb_driver.execute_lambda(lambda executor: delete_documents(executor, gov_id))
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

トランザクションで複数のステートメントの実行

```
# This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
# set your UPDATE to filter on vin and insured, and check if you updated something or
 not.

def do_insure_car(transaction_executor, vin):
    cursor = transaction_executor.execute_statement(
        "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
    first_record = next(cursor, None)
    if first_record:
        transaction_executor.execute_statement(
            "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
        return True
    else:
        return False
```

```
def insure_car(qldb_driver, vin_to_insure):
    return qldb_driver.execute_lambda(
        lambda executor: do_insure_car(executor, vin_to_insure))
```

再試行ロジック

ドライバーの execute_lambda メソッドには、再試行可能な例外 (タイムアウトや OCC 競合など) が発生した場合にトランザクションを再試行する組み込みの再試行メカニズムがあります。

3.x

再試行の最大数とバックオフ戦略を設定できます。

デフォルトの再試行制限は 4 であり、デフォルトのバックオフ戦略は 10 ミリ秒のベースを使用するエクスポネンシャルバックオフとジッターです。再試行設定は、pyqldb.config.retry_config.RetryConfig のインスタンスを使用して、ドライバーインスタンスごとおよびトランザクションごとに設定できます。

次のコード例では、ドライバーのインスタンスのカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

# Configuring retry limit to 2
retry_config = RetryConfig(retry_limit=2)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config)

# Configuring a custom backoff which increases delay by 1s for each attempt.
def custom_backoff(retry_attempt, error, transaction_id):
    return 1000 * retry_attempt

retry_config_custom_backoff = RetryConfig(retry_limit=2,
 custom_backoff=custom_backoff)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config_custom_backoff)
```

次のコード例では、特定の Lambda 実行のカスタム再試行制限とカスタムバックオフ戦略を使用して再試行ロジックを指定します。execute_lambda のこの設定は、ドライバーインスタンスに設定された再試行ロジックを上書きします。

```
from pyqldb.config.retry_config import RetryConfig
```

```
from pyqldb.driver.qldb_driver import QldbDriver

# Configuring retry limit to 2
retry_config_1 = RetryConfig(retry_limit=4)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config_1)

# Configuring a custom backoff which increases delay by 1s for each attempt.
def custom_backoff(retry_attempt, error, transaction_id):
    return 1000 * retry_attempt

retry_config_2 = RetryConfig(retry_limit=2, custom_backoff=custom_backoff)

# The config `retry_config_1` will be overriden by `retry_config_2`
qldb_driver.execute_lambda(lambda txn: txn.execute_statement("CREATE TABLE Person"),
 retry_config_2)
```

2.x

再試行の最大数は設定可能です。再試行制限を設定するには、PooledQldbDriver の初期化時に retry_limit プロパティを設定します。

デフォルトの再試行制限回数は 4 です。

一意性制約の実装

QLDB はユニークインデックスをサポートしていませんが、この動作をアプリケーションに実装できます。

Person テーブル内の GovId フィールドに対して一意性制約を実装するとします。これを行うには、以下を実行するトランザクションを記述します。

1. テーブルに指定された GovId を持つ既存のドキュメントがないことをアサートします。
2. アサーションに合格した場合は、ドキュメントを挿入します。

競合するトランザクションが同時にアサーションに合格すると、一方のトランザクションだけが正常にコミットされます。もう一方のトランザクションは OCC 競合例外で失敗します。

次のコード例は、この一意性制約ロジックを実装する方法を示しています。

```
def insert_documents(transaction_executor, gov_id, document):
```

```
    # Check if doc with GovId = gov_id exists
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
 = ?", gov_id)
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", document)

qldb_driver.execute_lambda(lambda executor: insert_documents(executor, gov_id,
 document))
```

> **Note**
>
> この例では、GovId フィールドでインデックスを使用して、パフォーマンスを最適化することをお勧めします。GovId でインデックスをオンにしないと、ステートメントのレイテンシーが大きくなり、OCC 競合例外やトランザクションタイムアウトが発生する可能性があります。

Amazon Ion の操作

以下のセクションでは、Amazon Ion モジュールを使用して Ion データを処理する方法について説明します。

目次

Ion モジュールのインポート

```
import amazon.ion.simpleion as simpleion
```

Ion 型の作成

次のコード例では、Ion テキストから Ion オブジェクトを作成します。

```
ion_text = '{GovId: "TOYENC486FH", FirstName: "Brent"}'
ion_obj = simpleion.loads(ion_text)

print(ion_obj['GovId']) # prints TOYENC486FH
print(ion_obj['Name']) # prints Brent
```

次のコード例では、Python `dict` から Ion オブジェクトを作成します。

```
a_dict = { 'GovId': 'TOYENC486FH',
           'FirstName': "Brent"
         }
ion_obj = simpleion.loads(simpleion.dumps(a_dict))

print(ion_obj['GovId']) # prints TOYENC486FH
print(ion_obj['FirstName']) # prints Brent
```

Ion バイナリダンプの取得

```
# ion_obj is an Ion struct
print(simpleion.dumps(ion_obj)) # b'\xe0\x01\x00\xea\xee\x97\x81\x83\xde\x93\x87\xbe
\x90\x85GovId\x89FirstName\xde\x94\x8a\x8bTOYENC486FH\x8b\x85Brent'
```

Ion テキストダンプの取得

```
# ion_obj is an Ion struct
print(simpleion.dumps(ion_obj, binary=False)) # prints $ion_1_0
 {GovId:'TOYENC486FH',FirstName:"Brent"}
```

Ion の操作方法の詳細については、GitHub で Amazon Ion のドキュメントを参照してください。QLDB で Ion を操作するコード例については、「Amazon QLDB で Amazon Ion のデータ型を操作する」を参照してください。

Amazon QLDB でドライバーを使用してセッションを管理する方法を理解する

リレーショナルデータベース管理システム (RDBMS) を使用した経験のある方は同時接続の知識をお持ちでしょう。QLDB の概念は、従来の RDBMS 接続とは異なります。HTTP のリクエストメッセージとレスポンスメッセージを使用してトランザクションを実行するからです。

QLDB では、同様の概念をアクティブセッションと呼びます。セッションは概念的にはユーザーログインに似ており、これによって、台帳に対するデータトランザクションリクエストの情報が管理されます。アクティブセッションとは、トランザクションがアクティブに実行されているセッションを意味します。トランザクションが終了したばかりで、サービスが別のトランザクションがすぐに開始されることを期待している状態もセッションと見なされることがあります。QLDB ではセッションごとに 1 つのトランザクションがアクティブに実行されます。

台帳ごとの同時アクティブセッションの制限数は、「Amazon QLDB でのクォータと制限」に定義されています。この制限に達すると、トランザクションの開始を試行するセッションはエラー (`LimitExceededException`) になります。

QLDB ドライバーを使用してアプリケーション内のセッションプールを設定するベストプラクティスについては、「Amazon QLDB の推奨事項」の「QldbDriver オブジェクトの設定」を参照してください。

目次

セッションライフサイクル

次の QLDB セッション API オペレーションのシーケンスは、QLDB セッションの一般的なライフサイクルを示しています。

1. `StartSession`

2. `StartTransaction`
3. `ExecuteStatement`
4. `CommitTransaction`
5. 複数のトランザクションを開始するには手順 2 ~ 4 を繰り返します (一度に 1 つのトランザクション)。
6. `EndSession`

セッションの期限切れ

QLDB は、トランザクションがアクティブに実行されているかどうかにかかわらず、13 ~ 17 分の合計ライフタイムが経過した後にそのセッションを期限切れにし、破棄します。ハードウェア障害、ネットワーク障害、アプリケーションの再起動など、さまざまな理由でセッションが失われたり、障害が生じたりすることがあります。QLDB はセッションに最大有効期間を適用し、クライアントアプリケーションがセッション障害に対して回復力を持つようにします。

QLDB ドライバーでのセッション処理

AWS SDK を使用して、QLDB セッション API を直接操作できますが、これにより、複雑性が増し、コミットダイジェストの計算も求められます。QLDB は、このコミットダイジェストを使用して、トランザクションの整合性を確保します。この API と直接やり取りするのではなく、QLDB ドライバーを使用することをお勧めします。

このドライバーにより、トランザクションデータ API 上に高レベルの抽象化レイヤーが形成されます。これにより SendCommand API コールを管理して、元帳データに対する PartiQL ステートメントの実行プロセスが合理化されます。これらの API コールにはいくつかのパラメータが必要です。これにより、ドライバーは、セッション、トランザクション、エラー発生時の再試行ポリシーの管理といった処理を行えるようになります。

トピック

- セッションプールの概要
- セッションプールとトランザクションロジック
- セッションをプールに戻す

セッションプールの概要

古いバージョンの QLDB ドライバー (例: Java v1.1.0) では、プールを使用しない標準の `QldbDriver` と、`PooledQldbDriver` という、2 つのドライバーオブジェクトを実装していました。名前が示すとおり、`PooledQldbDriver` では、トランザクション間で再利用するセッションプールが維持されます。

ユーザーからのフィードバックに基づいて、開発者は、標準ドライバーを使用せず、デフォルトでプーリング機能とそのメリットを得ることができます。そのため、`PooledQldbDriver` を削除し、そのセッションプール機能を `QldbDriver` に移動しました。この変更した機能は、各ドライバーの最新リリースに含まれています (例: Java v2.0.0)。

このドライバーにより、次の 3 つのレベルの抽象化が可能です。

- ドライバー (プールされたドライバー実装) — トップレベルの抽象化。ドライバーは、セッションのプールを維持し、管理します。ドライバーにトランザクションを実行させると、ドライバーはプールからセッションを選択し、それを使用してトランザクションを実行します。セッションエラー (`InvalidSessionException`) が原因でトランザクションが失敗した場合、ドライバーは別のセッションを使用してトランザクションを再試行します。基本的に、ドライバーはフルマネージドのセッションエクスペリエンスを提供します。
- セッション — ドライバーの 1 つ下のレベルとなる抽象化です。セッションはプール内に存在し、セッションのライフサイクルはドライバーが管理しています。トランザクションが失敗すると、ドライバーは同じセッションを使用して指定された回数までトランザクションを再試行します。しかし、セッションでエラー (`InvalidSessionException`) が発生すると、QLDB はそのセッションを内部的に破棄します。その場合、ドライバーは、プールから別のセッションを取得して、トランザクションを再試行する必要があります。
- トランザクション — 最も下のレベルの抽象化です。トランザクションはセッション内に存在し、トランザクションのライフサイクルはセッションによって管理されています。エラーが発生した場合、セッションによって、トランザクションが再試行される必要があります。また、セッションは、コミットされていない、またはキャンセルされたオープントランザクションをリークさせない役割も果たします。

各ドライバーの最新バージョンでは、抽象化のドライバーレベルでのみオペレーションを実行できます。個々のセッションとトランザクションを直接制御することはできません (つまり、新しいセッションまたはトランザクションを手動で開始する API オペレーションはありません)。

セッションプールとトランザクションロジック

各ドライバーの最新リリースでは、プールを使用しないドライバーオブジェクトが実装されなくなりました。デフォルトでは、QldbDriver オブジェクトがセッションプールを管理します。トランザクション実行の呼び出しを行うと、ドライバーは次の手順を実行します。

1. ドライバーは、セッションプールの制限に達したかどうかをチェックします。達している場合、ドライバーは NoSessionAvailable をスローします。それ以外の場合は、次のステップに進みます。
2. ドライバーは、利用可能なセッションがプール内にあるかどうかをチェックします。
   - プール内に使用可能なセッションがある場合、ドライバーはそのセッションを使用してトランザクションを実行します。
   - プール内に使用可能なセッションがない場合、ドライバーはセッションを新規作成し、それを使用してトランザクションを実行します。
3. ステップ 2 でセッションを取得した後、ドライバーは、セッションインスタンス上で execute オペレーションを呼び出します。
4. セッションの execute オペレーション内で、startTransaction を呼び出すことで、トランザクションの開始を試行します。
   - セッションが有効でない場合は、startTransaction の呼び出しが失敗し、ドライバーはステップ 1 に戻ります。
   - startTransaction の呼び出しが成功すると、ドライバーは次のステップに進みます。
5. ドライバーは Lambda 式を実行します。この Lambda 式には、PartiQL ステートメントを実行する 1 つ以上の呼び出しを含めることができます。Lambda 式の実行がエラーなしで終了すると、ドライバーはトランザクションのコミットに進みます。
6. トランザクションのコミットの結果は、次に示す 2 つのいずれかになります。
   - コミットが成功し、ドライバーは制御をアプリケーションコードに返します。
   - オプティミスティック同時実行制御 (OCC) の競合により、コミットが失敗します。この場合、ドライバーは同じセッションを使用して手順 4 ~ 6 を再試行します。再試行の最大数は、アプリケーションコードで設定可能です。デフォルトの制限数は 4 です。

> **Note**
>
> ステップ 4 ~ 6 の間に InvalidSessionException がスローされた場合、ドライバーはセッションを閉じたものとしてマークし、ステップ 1 に戻ってトランザクションを再試行します。

手順 4 ~ 6 で他の例外がスローされた場合、ドライバーは、例外が発生した状態で再試行可能かどうかをチェックします。可能な場合は、指定された再試行回数までトランザクションを再試行します。可能でない場合は、アプリケーションコードに例外を伝播 (バブルアップしてスロー) します。

セッションをプールに戻す

アクティブなトランザクションの過程で `InvalidSessionException` が発生した場合、ドライバーはセッションをプールに戻しません。その代わりに QLDB がセッションを破棄し、ドライバーはプールから別のセッションを取得します。それ以外の場合、ドライバーはセッションをプールに戻します。

Amazon QLDB ドライバーの推奨事項

このセクションでは、サポートされている言語の Amazon QLDB ドライバーを設定および使用するためのベストプラクティスについて説明します。提供されているコード例は Java 専用です。

これらの推奨事項は、ほとんどの一般的なユースケースに適用されますが、1 つのサイズがすべてのサイズに適合するとは限りません。アプリケーションに適した以下の推奨事項を使用してください。

トピック

- QldbDriver オブジェクトの設定
- 例外発生時の再試行
- パフォーマンスの最適化
- トランザクションごとに複数のステートメントを実行する

QldbDriver オブジェクトの設定

この `QldbDriver` は、トランザクション間で再利用されるセッションのプールを維持することによって、台帳への接続を管理します。セッションは、台帳との 1 つの接続を表します。QLDB ではセッションごとに 1 つのトランザクションがアクティブに実行されます。

Important

古いドライバーバージョンのセッションプール機能は、現在も、QldbDriver ではなく PooledQldbDriver にあります。次のいずれかのバージョンを使用している場合、このトピックに記載の QldbDriver は、PooledQldbDriver に置き換えてください。

ドライバー	バージョン
Java	1.1.0 以前
.NET	0.1.0-beta
Node.js	1.0.0-rc.1 以前
Python	2.0.2 以前

PooledQldbDriver オブジェクトは、ドライバーの最新バージョンでは廃止されています。最新バージョンにアップグレードして、PooledQldbDriver のすべてのインスタンスを QldbDriver に変換することをお勧めします。

QldbDriver をグローバルオブジェクトとして設定する

ドライバーとセッションの使用を最適化するには、アプリケーションインスタンスにドライバーのグローバルインスタンスが 1 つのみあるようにします。例えば Java では、Spring、Google Guice、Dagger などの依存関係インジェクションフレームワークを使用できます。次のコード例は、QldbDriver をシングルトンとして設定する方法を示しています。

```
@Singleton
public QldbDriver qldbDriver (AWSCredentialsProvider credentialsProvider,
                                  @Named(LEDGER_NAME_CONFIG_PARAM) String ledgerName)
 {
    QldbSessionClientBuilder builder = QldbSessionClient.builder();
    if (null != credentialsProvider) {
        builder.credentialsProvider(credentialsProvider);
    }
    return QldbDriver.builder()
            .ledger(ledgerName)
            .transactionRetryPolicy(RetryPolicy
```

```
                .builder()
                .maxRetries(3)
                .build())
            .sessionClientBuilder(builder)
            .build();
}
```

再試行を設定する

ドライバーは、一般的な一時例外 (SocketTimeoutException や NoHttpResponseException など) が発生すると、自動的にトランザクションを再試行します。QldbDriver のインスタンスを作成するときに transactionRetryPolicy 設定オブジェクトの maxRetries パラメータを使用すると、最大再試行回数を設定できます。(前のセクションで説明した古いドライバーバージョンの場合は、PooledQldbDriver の retryLimit パラメータを使用します)。

maxRetries の初期値は 4 です。

InvalidParameterException などのクライアント側のエラーは再試行できません。これらが発生すると、トランザクションが中止され、セッションがプールに戻され、例外がドライバーのクライアントにスローされます。

同時セッションと同時トランザクションの最大数を設定する

トランザクションを実行する QldbDriver インスタンスによって使用される台帳セッションの最大数は、その maxConcurrentTransactions パラメータで定義します。(前のセクションで説明した古いドライバーバージョンの場合は、PooledQldbDriver の poolLimit パラメータで定義します)。

この制限は、特定の AWS SDK で定義されているように、ゼロより大きく、セッションクライアントが許可する開いている HTTP 接続の最大数以下であることが必要です。たとえば Java では、接続の最大数は ClientConfiguration オブジェクトで設定されます。

maxConcurrentTransactions のデフォルト値は、AWS SDK で設定した最大接続数と同じものです。

アプリケーションで QldbDriver を設定するときは、スケーリングに関する以下の点を考慮してください。

- プールには常に、同時に実行する予定のトランザクションの数と同数以上のセッションが必要です。

- スーパバイザースレッドがワーカースレッドに委任するマルチスレッドモデルでは、そのドライバーにワーカースレッドの数と同数以上のセッションが必要です。それ以外の場合、ピーク負荷時に、スレッドはセッションが使用可能になるまで待機中になります。
- 台帳ごとの同時アクティブセッションのサービス制限は、「Amazon QLDB でのクォータと制限」で定義されています。すべてのクライアントで 1 つの台帳に使用される同時セッションの数は、この制限を超えないように設定してください。

例外発生時の再試行

QLDB で例外が発生したときに再試行する場合は、以下の推奨事項を考慮してください。

OccConflictException 発生時の再試行

オプティミスティック同時実行制御 (OCC) の競合例外は、トランザクションでアクセスしているデータがトランザクションの開始以降に変更された場合に発生します。トランザクションのコミットを試行する間、QLDB はこの例外をスローします。ドライバーは、`maxRetries` に設定されている回数だけトランザクションを再試行します。

OCC について、また、インデックスを使用して OCC 競合を制限するベストプラクティスについては「Amazon QLDB 同時実行モデル」を参照してください。

その他の例外発生時に QldbDriver の外部で再試行する

アプリケーション定義のカスタム例外が実行中にスローされたときにこのドライバーの外部でトランザクションを再試行するには、トランザクションをラップする必要があります。Java の例を挙げると、以下のコードは、Reslience4J ライブラリを使用して QLDB でトランザクションを再試行する方法を示しています。

```
private final RetryConfig retryConfig = RetryConfig.custom()
        .maxAttempts(MAX_RETRIES)
        .intervalFunction(IntervalFunction.ofExponentialRandomBackoff())
        // Retry this exception
        .retryExceptions(InvalidSessionException.class, MyRetryableException.class)
        // But fail for any other type of exception extended from RuntimeException
        .ignoreExceptions(RuntimeException.class)
        .build();

// Method callable by a client
public void myTransactionWithRetries(Params params) {
```

```
    Retry retry = Retry.of("registerDriver", retryConfig);

    Function<Params, Void> transactionFunction = Retry.decorateFunction(
            retry,
            parameters ->  transactionNoReturn(params));
    transactionFunction.apply(params);
}

private Void transactionNoReturn(Params params) {
    try (driver.execute(txn -> {
            // Transaction code
        });
    }
    return null;
}
```

> **Note**
>
> QLDB ドライバーの外部で行うトランザクションの再試行には乗数効果があります。例えば、QldbDriver が 3 回再試行するように設定されていて、カスタム再試行ロジックも 3 回再試行する場合、同じトランザクションを最大 9 回再試行できます。

トランザクションをべき等にする

ベストプラクティスとして、再試行の場合に予期しない結果を避けるために、書き込みトランザクションをべき等にしてください。トランザクションは、複数回実行して毎回同じ結果を生成できる場合、idempotent です。

詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

パフォーマンスの最適化

このドライバーを使用してトランザクションを実行するときにパフォーマンスを最適化するには、以下の点を考慮してください。

- execute オペレーションは常に、以下のコマンドを含む 3 つ以上の SendCommand API コールを QLDB に対して実行します。
  1. StartTransaction
  2. ExecuteStatement

このコマンドは、execute ブロックで実行する PartiQL ステートメントごとに呼び出されます。

3. CommitTransaction

アプリケーションの全体的なワークロードを計算するときに行われる API コールの合計数を考慮してください。

- 一般的に、シングルスレッドのライターから始め、1 つのトランザクション内で複数のステートメントをバッチ処理することでトランザクションを最適化することをお勧めします。「Amazon QLDB でのクォータと制限」で定義されているように、トランザクションサイズ、ドキュメントサイズ、トランザクションごとのドキュメント数のクォータを最大に設定します。
- 大量のトランザクション負荷に対してバッチ処理では不十分な場合は、ライターを追加してマルチスレッドを試すことができます。ただし、ドキュメントとトランザクションの順序付けに関するアプリケーション要件と、これにより生じる複雑さを慎重に検討する必要があります。

トランザクションごとに複数のステートメントを実行する

前のセクションで説明したように、トランザクションごとに複数のステートメントを実行して、アプリケーションのパフォーマンスを最適化できます。次のコード例では、トランザクション内で、テーブルをクエリし、そのテーブルのドキュメントを更新しています。そのために、Lambda 式を execute オペレーションに渡しています。

Java

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
public static boolean InsureCar(QldbDriver qldbDriver, final String vin) {
    final IonSystem ionSystem = IonSystemBuilder.standard().build();
    final IonString ionVin = ionSystem.newString(vin);

    return qldbDriver.execute(txn -> {
        Result result = txn.execute(
                "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
                ionVin);
        if (!result.isEmpty()) {
            txn.execute("UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
```

```
        }
        return false;
    });
}
```

.NET

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    ValueFactory valueFactory = new ValueFactory();
    IIonValue ionVin = valueFactory.NewString(vin);

    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.IAsyncResult result = await txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 ionVin);

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

Go

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
func InsureCar(driver *qldbdriver.QLDBDriver, vin string) (bool, error) {
    insured, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
```

```
        result, err := txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
        if err != nil {
            return false, err
        }

        hasNext := result.Next(txn)
        if !hasNext && result.Err() != nil {
            return false, result.Err()
        }

        if hasNext {
            _, err = txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
            if err != nil {
                return false, err
            }
            return true, nil
        }
        return false, nil
    })
    if err != nil {
        panic(err)
    }

    return insured.(bool), err
}
```

Node.js

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
async function insureCar(driver: QldbDriver, vin: string): Promise<boolean> {

    return await driver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 vin)).getResultList();

        if (results.length > 0) {
```

```
            await txn.execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin);
            return true;
        }
        return false;
    });
};
```

Python

```
# This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
# set your UPDATE to filter on vin and insured, and check if you updated something
 or not.

def do_insure_car(transaction_executor, vin):
    cursor = transaction_executor.execute_statement(
        "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
    first_record = next(cursor, None)
    if first_record:
        transaction_executor.execute_statement(
            "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
        return True
    else:
        return False

def insure_car(qldb_driver, vin_to_insure):
    return qldb_driver.execute_lambda(
        lambda executor: do_insure_car(executor, vin_to_insure))
```

ドライバーの execute オペレーションは、セッションとそのセッション内のトランザクションを暗黙で開始します。Lambda 式で実行する各ステートメントは、トランザクションでラップされます。すべてのステートメントが実行されると、ドライバーはトランザクションを自動的にコミットします。自動的な再試行の制限を超えたためにステートメントが失敗すると、トランザクションは中止されます。

トランザクションでの例外の伝播

トランザクションごとに複数のステートメントを実行する場合、通常、トランザクション内で例外をキャッチして続行することはお勧めしません。

たとえば、Java の場合、次のプログラムは RuntimeException のインスタンスをキャッチして エラーをログに記録し、続行します。このコード例は、UPDATE ステートメントが失敗してもトランザクションが成功するため、推奨されません。クライアントは、更新が失敗しても成功したと思い込む可能性があります。

> **⚠ Warning**
>
> このコード例は、使用しないでください。これは、推奨されない悪いパターンの例として提供されています。

```
// DO NOT USE this code example because it is considered bad practice
public static void main(final String... args) {
    ConnectToLedger.getDriver().execute(txn -> {
        final Result selectTableResult = txn.execute("SELECT * FROM Vehicle WHERE VIN
 ='123456789'");
        // Catching an error inside the transaction is an anti-pattern because the
 operation might
        // not succeed.
        // In this example, the transaction succeeds even when the update statement
 fails.
        // So, the client might assume that the update succeeded when it didn't.
        try {
            processResults(selectTableResult);
            String model = // some code that extracts the model
            final Result updateResult = txn.execute("UPDATE Vehicle SET model = ? WHERE
 VIN = '123456789'",
                    Constants.MAPPER.writeValueAsIonValue(model));
        } catch (RuntimeException e) {
            log.error("Exception when updating the Vehicle table {}", e.getMessage());
        }
    });
    log.info("Vehicle table updated successfully.");
}
```

その代わりに例外を伝播 (バブルアップ) します。トランザクションのいずれかの部分が失敗した場合は、execute オペレーションにトランザクションを中止させて、クライアントが適切に例外を処理できるようにします。

Amazon QLDB でドライバーが使用する再試行ポリシーを理解する

Amazon QLDB ドライバーは、再試行ポリシーを使用して、失敗したトランザクション を透過的に再試行し、一時的な例外を処理します。`CapacityExceededException` と `RateExceededException` などの例外は、通常、一定時間後に自己修復されます。例外により失敗したトランザクションは、適切な遅延後に再試行されると、成功する可能性があります。これにより、QLDB を使用するアプリケーションの安定性が向上します。

トピック

再試行可能なエラーのタイプ

ドライバーは、そのトランザクション内のオペレーション中に次の例外が発生した場合にのみ、トランザクションを自動的に再試行します。

- CapacityExceededException — リクエストが台帳の処理能力を超えたときに返ります。
- InvalidSessionException – セッションが有効でなくなった場合、またはセッションが存在しない場合に返ります。
- LimitExceededException — アクティブセッション数などのリソース制限を超えた場合に返ります。
- OccConflictException – オプティミスティック同時実行制御 (OCC) フェーズの検証で生じた障害によって、トランザクションをジャーナルに書き込めない場合に返ります。
- RateExceededException — リクエストの割合が、許可されているスループットを超えた場合に返ります。

デフォルトの再試行ポリシー

再試行ポリシーは、再試行条件とバックオフ戦略で構成されます。再試行条件ではトランザクションを再試行するタイミングを定義し、バックオフ戦略ではトランザクションを再試行するまでの待機時間を定義します。

ドライバーのインスタンスを作成する場合、デフォルトの再試行ポリシーでは、最大 4 回までの再試行と、エクスポネンシャルバックオフ戦略の使用を指定します。エクスポネンシャルバックオフ戦略では、等しいジッターを使用して、最小 10 ミリ秒、最大 5000 ミリ秒遅延させます。再試行ポリ

シー内でトランザクションを正常にコミットできない場合は、異なるタイミングでトランザクションを試行することをお勧めします。

エクスポネンシャルバックオフは、再試行間の待機時間を累進的に長くして、連続的なエラー応答に対処するという概念に基づいています。詳細については、AWS ブログ投稿、「エクスポネンシャルバックオフとジッター」を参照してください。

Amazon QLDB ドライバーの一般的なエラー

このセクションでは、QLDB セッション API を操作するときに、Amazon QLDB ドライバーからスローされる可能性のあるランタイムエラーについて説明します。

以下は、ドライバーによって返される一般的な例外の一覧です。それぞれの例外には、特定のエラーメッセージに加え、簡単な説明と考えられる解決方法に関する推奨事項が記載されています。

CapacityExceededException

メッセージ: 処理能力の超過

台帳の処理能力を超えたため、Amazon QLDB はリクエストを拒否しました。QLDB は、サービスのヘルスとパフォーマンスを維持するために、台帳ごとに内部スケーリング制限を適用します。この制限は、各リクエストのワークロードサイズによって異なります。例えば、非インデックス修飾クエリによるテーブルスキャンなど、非効率なデータトランザクションを実行する場合、リクエストのワークロードが増大する可能性があります。

リクエストを再試行する前に、待機することをお勧めします。アプリケーションでこの例外が継続的に発生する場合は、ステートメントを最適化し、台帳に送信するリクエストの割合やボリュームを減らします。ステートメントの最適化の例としては、トランザクションごとに実行するステートメントの数を減らすことや、テーブルインデックスのチューニングなどがあります。ステートメントの最適化方法およびテーブルスキャンの回避方法については、「クエリパフォーマンスの最適化」を参照してください。

最新バージョンの QLDB ドライバーを使用することもお勧めします。このドライバーには、エクスポネンシャルバックオフとジッターを使用するデフォルトの再試行ポリシーがあり、こうした例外発生時に自動的に再試行します。エクスポネンシャルバックオフは、再試行間の待機時間を累進的に長くして、連続的なエラー応答に対処するという概念に基づいています。

InvalidSessionException

メッセージ: トランザクション *`transactionId`* の有効期限が切れています

トランザクションが最大有効期間を超えました。トランザクションは、コミットされるまでに最大 30 秒間実行できます。このタイムアウト制限を超えると、トランザクションに行ったすべての操作が拒否され、QLDB はセッションを破棄します。この制限は、クライアントがトランザクションを開始し、トランザクションをコミットまたはキャンセルしないことで、セッションがリークするのを防ぎます。

この例外がアプリケーションでよく発生する場合、トランザクションの実行に時間がかかりすぎている可能性があります。トランザクションの実行に 30 秒以上かかる場合は、トランザクションを高速化するようにステートメントを最適化します。ステートメントの最適化の例としては、トランザクションごとに実行するステートメントの数を減らすことや、テーブルインデックスのチューニングなどがあります。詳細については、「クエリパフォーマンスの最適化」を参照してください。

InvalidSessionException

メッセージ: セッション *sessionId* の有効期限が切れています

最大合計存続期間を超えたため、QLDB はセッションを破棄しました。QLDB は、アクティブなトランザクションに関係なく、13 ~ 17 分後にセッションを破棄します。ハードウェア障害、ネットワーク障害、アプリケーションの再起動など、さまざまな理由でセッションが失われたり、障害が生じたりすることがあります。したがって、QLDB はセッションに最大有効期間を適用し、クライアントソフトウェアがセッション障害に対して回復力を持つようにします。

この例外が発生した場合は、新しいセッションを取得し、トランザクションを再試行することをお勧めします。また、最新バージョンの QLDB ドライバーを使用することをお勧めします。このドライバーは、アプリケーションに代わってセッションプールとそのヘルスを管理します。

InvalidSessionException

メッセージ: そのようなセッションはありません

クライアントは、存在しないセッションを使用して QLDB でトランザクションを実行しようとしました。以前に存在していたセッションをクライアントが使用していると仮定すると、次のいずれかが原因でセッションは存在しなくなる可能性があります。

- セッションが内部サーバーの障害 (つまり、HTTP 応答コード 500 のエラー) に関与している場合、QLDB は、カスタマーが不確実な状態のセッションでトランザクションを実行することを許可するのではなく、セッションを完全に破棄することを選択することがあります。その後、そのセッションでの再試行は、このエラーで失敗します。

- 期限切れのセッションは、最終的に QLDB に破棄されます。その後、そのセッションの使用を継続しようとすると、最初の `InvalidSessionException` エラーではなく、このエラーが発生します。

この例外が発生した場合は、新しいセッションを取得し、トランザクションを再試行することをお勧めします。また、最新バージョンの QLDB ドライバーを使用することをお勧めします。このドライバーは、アプリケーションに代わってセッションプールとそのヘルスを管理します。

RateExceededException

メッセージ: レートを超えました

QLDB は、呼び出し元のアイデンティティに基づいてクライアントをスロットルしました。QLDB は、トークンバケットスロットリングアルゴリズムを使用して、リージョンごと、アカウント単位でスロットリングを適用します。これは、サービスのパフォーマンスを向上し、QLDB のすべてのお客様に平等にご利用いただくために行われるものです。たとえば、`StartSessionRequest` オペレーションを使用して多数の同時セッションを取得しようとすると、スロットリングが発生する可能性があります。

アプリケーションの正常性を維持し、さらなるスロットリングを軽減するために、エクスポネンシャルバックオフおよびジッターを使用してこの例外を再試行できます。エクスポネンシャルバックオフは、再試行間の待機時間を累進的に長くして、連続的なエラー応答に対処するという概念に基づいています。最新バージョンの QLDB ドライバーを使用することをお勧めします。このドライバーには、エクスポネンシャルバックオフとジッターを使用するデフォルトの再試行ポリシーがあり、こうした例外発生時に自動的に再試行します。

最新バージョンの QLDB ドライバーは、アプリケーションが `StartSessionRequest` コールのために QLDB によって継続的にスロットリングしている場合に役立ちます。このドライバーは、トランザクション間で再利用されるセッションのプールを維持します。これにより、アプリケーションが行う `StartSessionRequest` コールの回数を減らすことができます。API スロットリング制限の引き上げをリクエストするには、AWS Support センターにお問い合わせください。

LimitExceededException

メッセージ: セッション制限を超えました

台帳が、アクティブなセッション数のクォータ (制限とも呼ばれる) を超えました。このクォータは Amazon QLDB でのクォータと制限 で定義されています。台帳のアクティブセッション数は結果的に一貫しており、常時クォータ付近で実行している台帳の場合、この例外が定期的に表示される場合があります。

アプリケーションの正常性を維持するために、この例外を再試行することをお勧めします。この例外を回避するには、すべてのクライアントで単一台帳で使用する同時セッションが 1,500 を超えないようにしてください。例えば、Java 用 Amazon QLDB ドライバーの maxConcurrentTransactions メソッドを使用して、ドライバーインスタンスで使用可能なセッションの最大数を設定できます。

QldbClientException

メッセージ: ストリームされた結果は、親トランザクションが開いている場合にのみ有効です

トランザクションは閉じられており、QLDB から結果を取得するために使用することはできません。トランザクションは、コミットまたはキャンセルされると終了します。

この例外は、クライアントが `Transaction` オブジェクトを直接操作していて、トランザクションのコミットまたはキャンセル後に QLDB から結果を取得しようとした場合に発生します。この問題を軽減するには、クライアントはトランザクションを閉じる前にデータを読み取る必要があります。

サンプルアプリケーションチュートリアルを使用した Amazon QLDB の使用開始

このチュートリアルでは、AWS SDK と Amazon QLDB ドライバーを使用して QLDB 台帳を作成し、サンプルデータを移入します。このドライバーを使用すると、アプリケーションはトランザクションデータ API を使用して QLDB とやり取りできるようになります。AWS SDK は QLDB リソース管理 API との連携をサポートします。

このシナリオで作成するサンプル台帳は、車両登録に関する完全な履歴情報を追跡する自動車部門 (DMV) データベースです。以下のトピックでは、車両登録の追加、変更、および変更履歴の確認方法について説明します。このガイドでは、登録ドキュメントを暗号的に検証する方法も示します。最後に、リソースをクリーンアップしてサンプル台帳を削除します。

このサンプルアプリケーションチュートリアルでは、次のプログラミング言語を使用できます。

トピック

- Amazon QLDB Java チュートリアル
- Amazon QLDB Node.js チュートリアル
- Amazon QLDB Python チュートリアル

Amazon QLDB Java チュートリアル

このチュートリアルのサンプルアプリケーション実装では、AWS SDK for Java と Amazon QLDB ドライバーを使用して QLDB 台帳を作成し、サンプルデータを移入します。

このチュートリアルを進めるときは、管理 API オペレーションについて「AWS SDK for Java API リファレンス」を参照できます。トランザクションデータのオペレーションについては、「QLDB Driver for Java API Reference」を参照してください。

> **Note**
>
> 該当する場合、一部のチュートリアルステップでは、サポートされているメジャーバージョンの Java 用 QLDB ドライバーに対して異なるコマンドまたはコード例があります。

トピック

Amazon QLDB Java サンプルアプリケーションのインストール

このセクションでは、ステップバイステップの Java チュートリアル用に提供されている Amazon QLDB サンプルアプリケーションをインストールして実行する方法について説明します。このサンプルアプリケーションのユースケースは、車両登録に関する完全な履歴情報を追跡する自動車部門 (DMV) データベースです。

Java 用の DMV サンプルアプリケーションは、GitHub リポジトリ aws-samples/amazon-qldb-dmv-sample-java でオープンソースとして公開されています。

前提条件

開始する前に、Java 用 QLDB ドライバーの「前提条件」を完了していることを確認します。これには以下が含まれます。

1. AWS にサインアップする。
2. QLDB の適切なアクセス許可を持つユーザーを作成します。このチュートリアルのすべての手順を完了するには、QLDB API を介して台帳リソースへのフル管理アクセス権が必要です。
3. AWS Cloud9 以外の IDE を使っている場合、Java をインストールして、開発のためのプログラムによるアクセスを許可します。

インストール

以下のステップでは、ローカル開発環境でサンプルアプリケーションをダウンロードして設定する方法について説明します。または、AWS Cloud9 を IDE として使用してサンプルアプリケーションのセットアップを自動化し、AWS CloudFormation テンプレートを使用して開発リソースをプロビジョニングします。

ローカルの開発環境

ここでは、独自のリソースと開発環境を使用して QLDB Java サンプルアプリケーションをダウンロードしてインストールする方法について説明します。

サンプルアプリケーションをダウンロードして実行するには

1. 次のコマンドを入力して、GitHub からサンプルアプリケーションのクローンを作成します。

   2.x

   ```
   git clone https://github.com/aws-samples/amazon-qldb-dmv-sample-java.git
   ```

   1.x

   ```
   git clone -b v1.2.0 https://github.com/aws-samples/amazon-qldb-dmv-sample-
   java.git
   ```

   このパッケージには、Gradle 設定と Java チュートリアル からの完全なコードが含まれています。

2. 提供されたアプリケーションをロードして実行します。
   - Eclipse を使用している場合:
     a. Eclipse を起動し、[Eclipse] メニューで、[File (ファイル)]、[Import (インポート)]、[Existing Gradle Project (既存の Gradle プロジェクト)] の順に選択します。
     b. プロジェクトのルートディレクトリで、`build.gradle` ファイルが格納されているアプリケーションディレクトリを参照して選択します。次に、[Finish (完了)] を選択し、インポートにデフォルトの Gradle 設定を使用します。
     c. 例として `ListLedgers` プログラムを実行してみることができます。`ListLedgers.java` ファイルのコンテキスト (右クリック) メニューを開き、[Run as Java Application (Java アプリケーションとして実行)] を選択します。
   - IntelliJ を使用している場合:
     a. IntelliJ を起動し、[IntelliJ] メニューで [File (ファイル)]、[Open (開く)] の順に選択します。
     b. プロジェクトのルートディレクトリで、`build.gradle` ファイルが格納されているアプリケーションディレクトリを参照して選択します。次に、[OK] を選択します。デフォルト設定をそのままにして、[OK] を再び選択します。
     c. 例として `ListLedgers` プログラムを実行してみることができます。`ListLedgers.java` ファイルのコンテキスト (右クリック) メニューを開き、[Run 'ListLedgers' (「ListLedgers」として実行)] を選択します。
3. ステップ 1: 新しい台帳を作成する に進み、チュートリアルを開始して台帳を作成します。

AWS Cloud9

以下の手順では、AWS Cloud9 を IDE として使用して Java 用の Amazon QLDB 車両登録サンプルアプリケーションのセットアップを自動化する方法について説明します。このガイドでは、AWS CloudFormation テンプレートを使用して、開発リソースをプロビジョニングします。

AWS Cloud9 の詳細については、「AWS Cloud9 ユーザーガイド」を参照してください。AWS CloudFormation の詳細については、「AWS CloudFormation ユーザーガイド」を参照してください。

トピック

- パート 1: リソースのプロビジョニング
- パート 2: IDE をセットアップする

- パート 3: QLDB DMV サンプルアプリケーションを実行する

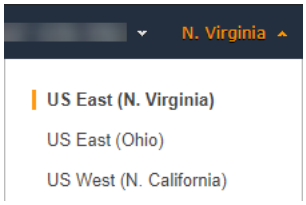
パート 1: リソースのプロビジョニング

この最初のステップでは、AWS CloudFormation を使用して、Amazon QLDB サンプルアプリケーションと共に開発環境をセットアップするために必要なリソースをプロビジョニングします。

AWS CloudFormation コンソールを開いて QLDB サンプルアプリケーションテンプレートをロードするには

1. AWS Management Console にサインインし、AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。

   QLDB をサポートするリージョンに切り替えます。完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。以下の AWS Management Console のスクリーンショットでは、AWS リージョン として米国東部 (バージニア北部) が選択されています。

2. AWS CloudFormation コンソールで [Create stack (スタックの作成)] を選択し、[With new resources (standard) (新しいリソース (標準) を使用)] を選択します。
3. [Create stack (スタックの作成)] ページの [Specify template (テンプレートの指定)] で、[Amazon S3 URL] を選択します。
4. 次の URL を入力し、[Next (次へ)] を選択します。

   ```
   https://amazon-qldb-assets.s3.amazonaws.com/templates/QLDB-DMV-SampleApp.yml
   ```

5. [Stack name (スタック名)] (**qldb-sample-app** など) を入力し、[Next (次へ)] を選択します。
6. 必要に応じてタグを追加し、デフォルトのオプションはそのままにしておきます。続いて、[Next] (次へ) を選択します。
7. スタックの設定を確認し、[Create stack (スタックの作成)] を選択します。AWS CloudFormation スクリプトの実行完了までには数分かかることがあります。

   このスクリプトは、このチュートリアルで QLDB サンプルアプリケーションを実行するために使用する、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスが関連付けられた

AWS Cloud9 環境をプロビジョニングします。また、GitHub から AWS Cloud9 開発環境へ aws-samples/amazon-qldb-dmv-sample-java リポジトリをクローンします。

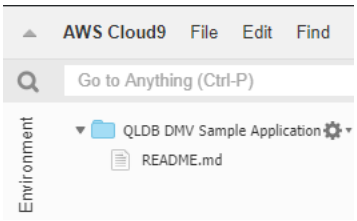
パート 2: IDE をセットアップする

このステップでは、クラウド開発環境のセットアップを完了します。提供されたシェルスクリプトをダウンロードして実行し、サンプルアプリケーションの依存関係を使用して AWS Cloud9 IDE を設定します。

AWS Cloud9 環境をセットアップするには

1. https://console.aws.amazon.com/cloud9/ で、AWS Cloud9 コンソールを開きます。
2. [Your environments (環境)] で、QLDB DMV Sample Application という名前の環境のカードを探し、[Open IDE (IDE を開く)] を選択します。お使いの環境によっては、基盤となる EC2 インスタンスが起動する際、ロードに時間がかかる場合があります。

   お使いの AWS Cloud9 環境には、チュートリアルの実行に必要なシステムの依存関係が事前設定されています。コンソールの [Environment (環境)] ナビゲーションペインで、`QLDB DMV Sample Application` という名前のフォルダが表示されていることを確認します。次の AWS Cloud9 コンソールのスクリーンショットは、QLDB DMV サンプルアプリケーション環境フォルダペインを示しています。

   ナビゲーションペインが表示されない場合は、コンソールの左側にある [Environment (環境)] タブを切り替えます。ペインにフォルダが全く表示されない場合は、設定アイコン () を使用して [Show Environment Root] (環境ルートの表示) を有効にします。
3. コンソールの下部のペインに、bash ターミナルウィンドウが開かれます。これが表示されない場合は、コンソール上部にある [Window] (ウィンドウ) メニューから [New Terminal] (新しいターミナル) を選択します。
4. 次に、セットアップスクリプトをダウンロードして実行して OpenJDK 8 をインストールし、該当する場合は Git リポジトリから適切なブランチをチェックアウトします。前のステップで作成した AWS Cloud9 ターミナルで、以下の 2 つのコマンドを順に実行します。

2.x

```
aws s3 cp s3://amazon-qldb-assets/setup-scripts/dmv-setup-v2.sh .
```

```
sh dmv-setup-v2.sh
```

1.x

```
aws s3 cp s3://amazon-qldb-assets/setup-scripts/dmv-setup.sh .
```

```
sh dmv-setup.sh
```

完了すると、ターミナルに次のメッセージが表示されます。

```
** DMV Sample App setup completed , enjoy!! **
```

5. サンプルアプリケーションのコードを AWS Cloud9 で参照します。特に、ディレクトリパス `src/main/java/software/amazon/qldb/tutorial` にあるコードを確認します。

パート 3: QLDB DMV サンプルアプリケーションを実行する

このステップでは、AWS Cloud9 を使用して Amazon QLDB DMV サンプルアプリケーションタスクを実行する方法について説明します サンプルコードを実行するには、AWS Cloud9 ターミナルに戻るか、「パート 2: IDE をセットアップする」で行ったように、新しいターミナルウィンドウを作成します。

サンプルアプリケーションを実行するには

1. ターミナルで次のコマンドを実行し、プロジェクトのルートディレクトリに切り替えます。

```
cd ~/environment/amazon-qldb-dmv-sample-java
```

次のディレクトリパスの例を実行していることを確認します。

```
/home/ec2-user/environment/amazon-qldb-dmv-sample-java/
```

2. 次のコマンドは、各タスクを実行するための Gradle 構文を示します。

```
./gradlew run -Dtutorial=Task
```

例えば、次のコマンドを実行して、AWS アカウント と現在のリージョンのすべての台帳を一覧表示します。

```
./gradlew run -Dtutorial=ListLedgers
```

3. ステップ 1: 新しい台帳を作成する に進み、チュートリアルを開始して台帳を作成します。
4. (オプション) このチュートリアルを完了した後、不要になった AWS CloudFormation リソースをクリーンアップします。

   a. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開き、「パート 1: リソースのプロビジョニング」で作成したスタックを削除します。

   b. また、AWS CloudFormation テンプレートによって作成された AWS Cloud9 スタックも削除します。

ステップ 1: 新しい台帳を作成する

このステップでは、`vehicle-registration` という名前の新しい Amazon QLDB 台帳を作成します。

新しい台帳を作成するには

1. 次のファイル (`Constants.java`) を確認します。このファイルには、このチュートリアル内の他のすべてのプログラムで使用される定数値が含まれています。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
```

```
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;

/**
 * Constant values used throughout this tutorial.
 */
public final class Constants {
    public static final int RETRY_LIMIT = 4;
    public static final String LEDGER_NAME = "vehicle-registration";
    public static final String STREAM_NAME = "vehicle-registration-stream";
    public static final String VEHICLE_REGISTRATION_TABLE_NAME =
 "VehicleRegistration";
    public static final String VEHICLE_TABLE_NAME = "Vehicle";
    public static final String PERSON_TABLE_NAME = "Person";
    public static final String DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
    public static final String VIN_INDEX_NAME = "VIN";
    public static final String PERSON_GOV_ID_INDEX_NAME = "GovId";
    public static final String
 VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
    public static final String DRIVER_LICENSE_NUMBER_INDEX_NAME =
 "LicenseNumber";
    public static final String DRIVER_LICENSE_PERSONID_INDEX_NAME = "PersonId";
```

```
    public static final String JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-
tutorial-journal-export";
    public static final String USER_TABLES = "information_schema.user_tables";
    public static final String LEDGER_NAME_WITH_TAGS = "tags";
    public static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    public static final IonObjectMapper MAPPER = new IonValueMapper(SYSTEM);

    private Constants() { }

    static {
        MAPPER.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    }
}
```

1.x

> **Important**
>
> Amazon Ion パッケージの場合、アプリケーションで名前空間 `com.amazon.ion` を使用する必要があります。AWS SDK for Java は、名前空間 `software.amazon.ion` の別の Ion パッケージに依存しますが、これは QLDB ドライバーと互換性がないレガシーパッケージです。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;

/**
 * Constant values used throughout this tutorial.
 */
public final class Constants {
    public static final int RETRY_LIMIT = 4;
    public static final String LEDGER_NAME = "vehicle-registration";
    public static final String STREAM_NAME = "vehicle-registration-stream";
    public static final String VEHICLE_REGISTRATION_TABLE_NAME =
 "VehicleRegistration";
    public static final String VEHICLE_TABLE_NAME = "Vehicle";
    public static final String PERSON_TABLE_NAME = "Person";
    public static final String DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
    public static final String VIN_INDEX_NAME = "VIN";
    public static final String PERSON_GOV_ID_INDEX_NAME = "GovId";
    public static final String
 VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
    public static final String DRIVER_LICENSE_NUMBER_INDEX_NAME =
 "LicenseNumber";
    public static final String DRIVER_LICENSE_PERSONID_INDEX_NAME = "PersonId";
    public static final String JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-
tutorial-journal-export";
    public static final String USER_TABLES = "information_schema.user_tables";
    public static final String LEDGER_NAME_WITH_TAGS = "tags";
    public static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    public static final IonObjectMapper MAPPER = new IonValueMapper(SYSTEM);

    private Constants() { }
```

```
    static {
        MAPPER.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    }
}
```

> **Note**
>
> この Constants クラスには、オープンソースの Jackson IonValueMapper クラスのインスタンスが含まれています。このマッパーを使用して、読み取りおよび書き込みトランザクションを実行するときに Amazon Ion データを処理できます。

CreateLedger.java ファイルには、台帳の現在のステータスを説明する次のプログラム (DescribeLedger.java) にも依存しています。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.DescribeLedgerRequest;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Describe a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class DescribeLedger {
    public static AmazonQLDB client = CreateLedger.getClient();
    public static final Logger log = LoggerFactory.getLogger(DescribeLedger.class);

    private DescribeLedger() { }

    public static void main(final String... args) {
        try {

            describe(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to describe a ledger!", e);
        }
    }

    /**
     * Describe a ledger.
     *
     * @param name
     *              Name of the ledger to describe.
     * @return {@link DescribeLedgerResult} from QLDB.
     */
    public static DescribeLedgerResult describe(final String name) {
        log.info("Let's describe ledger with name: {}...", name);
        DescribeLedgerRequest request = new DescribeLedgerRequest().withName(name);
        DescribeLedgerResult result = client.describeLedger(request);
        log.info("Success. Ledger description: {}", result);
```

```
        return result;
    }
}
```

2. CreateLedger.java プログラムをコンパイルして実行し、vehicle-registration という名前の台帳を作成します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.CreateLedgerRequest;
import com.amazonaws.services.qldb.model.CreateLedgerResult;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;
```

```
import com.amazonaws.services.qldb.model.LedgerState;
import com.amazonaws.services.qldb.model.PermissionsMode;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Create a ledger and wait for it to be active.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateLedger {
    public static final Logger log =
 LoggerFactory.getLogger(CreateLedger.class);
    public static final Long LEDGER_CREATION_POLL_PERIOD_MS = 10_000L;
    public static String endpoint = null;
    public static String region = null;
    public static AmazonQLDB client = getClient();

    private CreateLedger() {
    }

    /**
     * Build a low-level QLDB client.
     *
     * @return {@link AmazonQLDB} control plane client.
     */
    public static AmazonQLDB getClient() {
        AmazonQLDBClientBuilder builder = AmazonQLDBClientBuilder.standard();
        if (null != endpoint && null != region) {
            builder.setEndpointConfiguration(new
 AwsClientBuilder.EndpointConfiguration(endpoint, region));
        }
        return builder.build();
    }

    public static void main(final String... args) throws Exception {
        try {
            client = getClient();

            create(Constants.LEDGER_NAME);

            waitForActive(Constants.LEDGER_NAME);
```

```
        } catch (Exception e) {
            log.error("Unable to create the ledger!", e);
            throw e;
        }
    }

    /**
     * Create a new ledger with the specified ledger name.
     *
     * @param ledgerName Name of the ledger to be created.
     * @return {@link CreateLedgerResult} from QLDB.
     */
    public static CreateLedgerResult create(final String ledgerName) {
        log.info("Let's create the ledger with name: {}...", ledgerName);
        CreateLedgerRequest request = new CreateLedgerRequest()
                .withName(ledgerName)
                .withPermissionsMode(PermissionsMode.ALLOW_ALL);
        CreateLedgerResult result = client.createLedger(request);
        log.info("Success. Ledger state: {}.", result.getState());
        return result;
    }

    /**
     * Wait for a newly created ledger to become active.
     *
     * @param ledgerName Name of the ledger to wait on.
     * @return {@link DescribeLedgerResult} from QLDB.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static DescribeLedgerResult waitForActive(final String ledgerName)
throws InterruptedException {
        log.info("Waiting for ledger to become active...");
        while (true) {
            DescribeLedgerResult result = DescribeLedger.describe(ledgerName);
            if (result.getState().equals(LedgerState.ACTIVE.name())) {
                log.info("Success. Ledger is active and ready to use.");
                return result;
            }
            log.info("The ledger is still creating. Please wait...");
            Thread.sleep(LEDGER_CREATION_POLL_PERIOD_MS);
        }
    }
```

```
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.CreateLedgerRequest;
import com.amazonaws.services.qldb.model.CreateLedgerResult;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;
import com.amazonaws.services.qldb.model.LedgerState;
import com.amazonaws.services.qldb.model.PermissionsMode;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
/**
 * Create a ledger and wait for it to be active.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateLedger {
    public static final Logger log =
 LoggerFactory.getLogger(CreateLedger.class);
    public static final Long LEDGER_CREATION_POLL_PERIOD_MS = 10_000L;
    public static AmazonQLDB client = getClient();

    private CreateLedger() { }

    /**
     * Build a low-level QLDB client.
     *
     * @return {@link AmazonQLDB} control plane client.
     */
    public static AmazonQLDB getClient() {
        return AmazonQLDBClientBuilder.standard().build();
    }

    public static void main(final String... args) throws Exception {
        try {

            create(Constants.LEDGER_NAME);

            waitForActive(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to create the ledger!", e);
            throw e;
        }
    }

    /**
     * Create a new ledger with the specified ledger name.
     *
     * @param ledgerName
     *              Name of the ledger to be created.
     * @return {@link CreateLedgerResult} from QLDB.
     */
```

```
    public static CreateLedgerResult create(final String ledgerName) {
        log.info("Let's create the ledger with name: {}...", ledgerName);
        CreateLedgerRequest request = new CreateLedgerRequest()
                .withName(ledgerName)
                .withPermissionsMode(PermissionsMode.ALLOW_ALL);
        CreateLedgerResult result = client.createLedger(request);
        log.info("Success. Ledger state: {}.", result.getState());
        return result;
    }

    /**
     * Wait for a newly created ledger to become active.
     *
     * @param ledgerName
     *              Name of the ledger to wait on.
     * @return {@link DescribeLedgerResult} from QLDB.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static DescribeLedgerResult waitForActive(final String ledgerName)
 throws InterruptedException {
        log.info("Waiting for ledger to become active...");
        while (true) {
            DescribeLedgerResult result = DescribeLedger.describe(ledgerName);
            if (result.getState().equals(LedgerState.ACTIVE.name())) {
                log.info("Success. Ledger is active and ready to use.");
                return result;
            }
            log.info("The ledger is still creating. Please wait...");
            Thread.sleep(LEDGER_CREATION_POLL_PERIOD_MS);
        }
    }
}
```

> **Note**
>
> - `createLedger` の呼び出しで、台帳名とアクセス許可モードを指定する必要があります。台帳データのセキュリティを最大化する、`STANDARD` のアクセス許可モードの使用を推奨します。
> - 台帳を作成すると、デフォルトで削除保護が有効になります。これは QLDB の機能であり、ユーザーによって台帳が削除されるのを防ぎます。QLDB API または AWS

Command Line Interface (AWS CLI) を使用して、台帳作成時に削除保護を無効にすることもできます。

- 必要に応じて、台帳にアタッチするタグを指定することもできます。

新しい台帳への接続を確認するには、「ステップ 2: 台帳への接続をテストする」に進みます。

ステップ 2: 台帳への接続をテストする

このステップでは、トランザクションデータ API エンドポイントを使用して Amazon QLDB の `vehicle-registration` 台帳に接続できることを確認します。

台帳への接続をテストするには

1. 以下のプログラム (ConnectToLedger.java) を確認します。このプログラムは、`vehicle-registration` 台帳へのデータセッション接続を作成します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
```

```
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.net.URI;
import java.net.URISyntaxException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.awssdk.services.qldbsession.QldbSessionClientBuilder;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.RetryPolicy;

/**
 * Connect to a session for a given ledger using default settings.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class ConnectToLedger {
    public static final Logger log =
 LoggerFactory.getLogger(ConnectToLedger.class);
    public static AwsCredentialsProvider credentialsProvider;
    public static String endpoint = null;
    public static String ledgerName = Constants.LEDGER_NAME;
    public static String region = null;
    public static QldbDriver driver;

    private ConnectToLedger() {
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @param retryAttempts How many times the transaction will be retried in
     * case of a retryable issue happens like Optimistic Concurrency Control
 exception,
     * server side failures or network issues.
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver(int retryAttempts) {
```

```
        QldbSessionClientBuilder builder = getAmazonQldbSessionClientBuilder();
        return QldbDriver.builder()
                             .ledger(ledgerName)
                             .transactionRetryPolicy(RetryPolicy
                                   .builder()
                                   .maxRetries(retryAttempts)
                                   .build())
                             .sessionClientBuilder(builder)
                             .build();
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver() {
        QldbSessionClientBuilder builder = getAmazonQldbSessionClientBuilder();
        return QldbDriver.builder()
            .ledger(ledgerName)
            .transactionRetryPolicy(RetryPolicy.builder()

.maxRetries(Constants.RETRY_LIMIT).build())
            .sessionClientBuilder(builder)
            .build();
    }

    /**
     * Creates a QldbSession builder that is passed to the QldbDriver to connect
to the Ledger.
     *
     * @return An instance of the AmazonQLDBSessionClientBuilder
     */
    public static QldbSessionClientBuilder getAmazonQldbSessionClientBuilder() {
        QldbSessionClientBuilder builder = QldbSessionClient.builder();
        if (null != endpoint && null != region) {
            try {
                builder.endpointOverride(new URI(endpoint));
            } catch (URISyntaxException e) {
                throw new IllegalArgumentException(e);
            }
        }
        if (null != credentialsProvider) {
            builder.credentialsProvider(credentialsProvider);
```

```
        }
        return builder;
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver getDriver() {
        if (driver == null) {
            driver = createQldbDriver();
        }
        return driver;
    }


    public static void main(final String... args) {
        Iterable<String> tables = ConnectToLedger.getDriver().getTableNames();
        log.info("Existing tables in the ledger:");
        for (String table : tables) {
            log.info("- {} ", table);
        }
    }
}
```

Note

- 台帳に対してデータオペレーションを実行するには、QldbDriver クラスのインスタンスを作成して、特定の台帳に接続する必要があります。これは、前のステップで台帳の作成に使用した AmazonQLDB クライアントとは異なるクライアントオブジェクトです。前のクライアントは、「Amazon QLDB API リファレンス」に示されている管理 API オペレーションの実行にのみ使用されます。
- まず、QldbDriver オブジェクトを作成します。このドライバーを作成するときに台帳名を指定する必要があります。

  次に、このドライバーの execute メソッドを使用して、PartiQL ステートメントを実行できます。
- オプションで、トランザクション例外の最大再試行回数を指定できます。execute メソッドは、オプティミスティック同時実行制御 (OCC、optimistic

concurrency control) の競合やその他の一般的な一時例外を、この設定可能な制限に達するまで自動的に再試行します。デフォルト値は、「4」です。

この制限に達した後もトランザクションが成功しない場合、ドライバーが例外をスローします。詳細については、「Amazon QLDB でドライバーが使用する再試行ポリシーを理解する」を参照してください。

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.qldbsession.AmazonQLDBSessionClientBuilder;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
import software.amazon.qldb.PooledQldbDriver;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.exceptions.QldbClientException;

/**
 * Connect to a session for a given ledger using default settings.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class ConnectToLedger {
    public static final Logger log =
 LoggerFactory.getLogger(ConnectToLedger.class);
    public static AWSCredentialsProvider credentialsProvider;
    public static String endpoint = null;
    public static String ledgerName = Constants.LEDGER_NAME;
    public static String region = null;
    private static PooledQldbDriver driver;

    private ConnectToLedger() {
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static PooledQldbDriver createQldbDriver() {
        AmazonQLDBSessionClientBuilder builder =
 AmazonQLDBSessionClientBuilder.standard();
        if (null != endpoint && null != region) {
            builder.setEndpointConfiguration(new
 AwsClientBuilder.EndpointConfiguration(endpoint, region));
        }
        if (null != credentialsProvider) {
            builder.setCredentials(credentialsProvider);
        }
        return PooledQldbDriver.builder()
                .withLedger(ledgerName)
                .withRetryLimit(Constants.RETRY_LIMIT)
                .withSessionClientBuilder(builder)
```

```
                .build();
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static PooledQldbDriver getDriver() {
        if (driver == null) {
            driver = createQldbDriver();
        }
        return driver;
    }

    /**
     * Connect to a ledger through a {@link QldbDriver}.
     *
     * @return {@link QldbSession}.
     */
    public static QldbSession createQldbSession() {
        return getDriver().getSession();
    }

    public static void main(final String... args) {
        try (QldbSession qldbSession = createQldbSession()) {
            log.info("Listing table names ");
            for (String tableName : qldbSession.getTableNames()) {
                log.info(tableName);
            }
        } catch (QldbClientException e) {
            log.error("Unable to create session.", e);
        }
    }
}
```

> **Note**
>
> - 台帳に対してデータオペレーションを実行するには、PooledQldbDriver または QldbDriver クラスのインスタンスを作成して、特定の台帳に接続する必要があります。これは、前のステップで台帳の作成に使用した AmazonQLDB クライアントとは異なるクライアントオブジェクトです。前のクライアントは、「Amazon

QLDB API リファレンス」に示されている管理 API オペレーションの実行にのみ使用されます。

`QldbDriver` でカスタムセッションプールを実装する必要がない限り、`PooledQldbDriver` を使用することをお勧めします。`PooledQldbDriver` のデフォルトプールサイズは、セッションクライアントが許可する開いている HTTP 接続の最大数です。

- まず、`PooledQldbDriver` オブジェクトを作成します。このドライバーを作成するときに台帳名を指定する必要があります。

  次に、このドライバーの `execute` メソッドを使用して、PartiQL ステートメントを実行できます。または、このプールされたドライバーオブジェクトから手動でセッションを作成し、セッションの `execute` メソッドを使用することもできます。セッションは、台帳との 1 つの接続を表します。

- オプションで、トランザクション例外の最大再試行回数を指定できます。`execute` メソッドは、オプティミスティック同時実行制御 (OCC、optimistic concurrency control) の競合やその他の一般的な一時例外を、この設定可能な制限に達するまで自動的に再試行します。デフォルト値は、「4」です。

  この制限に達した後もトランザクションが成功しない場合、ドライバーが例外をスローします。詳細については、「Amazon QLDB でドライバーが使用する再試行ポリシーを理解する」を参照してください。

2. `ConnectToLedger.java` プログラムをコンパイルし実行して、`vehicle-registration` 台帳へのデータセッション接続をテストします。

AWS リージョン のオーバーライド

サンプルアプリケーションは、前提条件のステップ デフォルトの AWS の認証情報とリージョンを設定する で説明されているように設定できる、デフォルトの AWS リージョン で QLDB に接続します。QLDB セッションクライアントビルダープロパティを変更することで、リージョンを変更することもできます。

2.x

次のサンプルコードは、新しい `QldbSessionClientBuilder` オブジェクトをインスタンス化します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.qldbsession.QldbSessionClientBuilder;

// This client builder will default to US East (Ohio)
QldbSessionClientBuilder builder = QldbSessionClient.builder()
    .region(Region.US_EAST_2);
```

region メソッドを使用して、利用可能な任意のリージョンで、QLDB を対象としてコードを実行できます。完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

1.x

次のサンプルコードは、新しい AmazonQLDBSessionClientBuilder オブジェクトをインスタンス化します。

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.qldbsession.AmazonQLDBSessionClientBuilder;

// This client builder will default to US East (Ohio)
AmazonQLDBSessionClientBuilder builder = AmazonQLDBSessionClientBuilder.standard()
    .withRegion(Regions.US_EAST_2);
```

withRegion メソッドを使用して、利用可能な任意のリージョンで、QLDB を対象としてコードを実行できます。完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

vehicle-registration 台帳にテーブルを作成するには、「ステップ 3: テーブル、インデックス、およびサンプルデータを作成する」に進みます。

ステップ 3: テーブル、インデックス、およびサンプルデータを作成する

Amazon QLDB 台帳がアクティブになり、接続を受け入れると、車両、車両の所有者、登録情報に関するデータのテーブルを作成できるようになります。テーブルとインデックスを作成した後、これらにデータをロードできます。

このステップでは、vehicle-registration 台帳に 4 つのテーブルを作成します。

- VehicleRegistration
- Vehicle

- Person
- DriversLicense

以下のインデックスも作成します。

テーブル名	フィールド
VehicleRegistration	VIN
VehicleRegistration	LicensePlateNumber
Vehicle	VIN
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

サンプルデータを挿入するときは、まず Person テーブルにドキュメントを挿入します。次に、各 Person ドキュメントからシステムによって割り当てられた id を使用して、適切な VehicleRegistration および DriversLicense ドキュメントの対応するフィールドに入力します。

> **Tip**
>
> ベストプラクティスとして、外部キーには、ドキュメントにシステムによって割り当てられた id を使用します。一意の識別子 (車両の VIN など) 用にフィールドを定義することはできますが、実際のドキュメントの一意の識別子はその id です。このフィールドはドキュメントのメタデータに含まれており、コミット済みビュー (テーブルのシステム定義のビュー) でクエリを実行できます。
> QLDB におけるビューの詳細については、「重要な概念」を参照してください。メタデータの詳細については、「ドキュメントのメタデータのクエリの実行」を参照してください。

サンプルデータを設定するには

1. 以下の .java ファイルを確認します。これらのモデルクラスは、vehicle-registration テーブルに保存するドキュメントを表します。Amazon Ion 形式との間でシリアル化されます。

> **Note**
>
> Amazon QLDB のドキュメント は、 JSON のスーパーセットであるIon フォーマットで保存されます。したがって、FasterXML Jackson ライブラリを使用して、JSON でデータをモデル化できます。

1. DriversLicense.java

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonCreator;
```

```
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

import java.time.LocalDate;

/**
 * Represents a driver's license, serializable to (and from) Ion.
 */
public final class DriversLicense implements RevisionData {
    private final String personId;
    private final String licenseNumber;
    private final String licenseType;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate validFromDate;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate validToDate;

    @JsonCreator
    public DriversLicense(@JsonProperty("PersonId") final String personId,
                          @JsonProperty("LicenseNumber") final String
 licenseNumber,
                          @JsonProperty("LicenseType") final String licenseType,
                          @JsonProperty("ValidFromDate") final LocalDate
 validFromDate,
                          @JsonProperty("ValidToDate") final LocalDate
 validToDate) {
        this.personId = personId;
        this.licenseNumber = licenseNumber;
        this.licenseType = licenseType;
        this.validFromDate = validFromDate;
        this.validToDate = validToDate;
    }

    @JsonProperty("PersonId")
    public String getPersonId() {
        return personId;
    }
```

```
    @JsonProperty("LicenseNumber")
    public String getLicenseNumber() {
        return licenseNumber;
    }

    @JsonProperty("LicenseType")
    public String getLicenseType() {
        return licenseType;
    }

    @JsonProperty("ValidFromDate")
    public LocalDate getValidFromDate() {
        return  validFromDate;
    }

    @JsonProperty("ValidToDate")
    public LocalDate getValidToDate() {
        return  validToDate;
    }

    @Override
    public String toString() {
        return "DriversLicense{" +
                "personId='" + personId + '\'' +
                ", licenseNumber='" + licenseNumber + '\'' +
                ", licenseType='" + licenseType + '\'' +
                ", validFromDate=" + validFromDate +
                ", validToDate=" + validToDate +
                '}';
    }
}
```

2. Person.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
```

```
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import java.time.LocalDate;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;

import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

/**
 * Represents a person, serializable to (and from) Ion.
 */
public final class Person implements RevisionData {
    private final String firstName;
    private final String lastName;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate dob;
    private final String govId;
    private final String govIdType;
    private final String address;

    @JsonCreator
    public Person(@JsonProperty("FirstName") final String firstName,
```

```
                @JsonProperty("LastName") final String lastName,
                @JsonProperty("DOB") final LocalDate dob,
                @JsonProperty("GovId") final String govId,
                @JsonProperty("GovIdType") final String govIdType,
                @JsonProperty("Address") final String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dob = dob;
        this.govId = govId;
        this.govIdType = govIdType;
        this.address = address;
    }

    @JsonProperty("Address")
    public String getAddress() {
        return address;
    }

    @JsonProperty("DOB")
    public LocalDate getDob() {
        return dob;
    }

    @JsonProperty("FirstName")
    public String getFirstName() {
        return firstName;
    }

    @JsonProperty("LastName")
    public String getLastName() {
        return lastName;
    }

    @JsonProperty("GovId")
    public String getGovId() {
        return govId;
    }

    @JsonProperty("GovIdType")
    public String getGovIdType() {
        return govIdType;
    }

    /**
```

```
     * This returns the unique document ID given a specific government ID.
     *
     * @param txn
     *              A transaction executor object.
     * @param govId
     *              The government ID of a driver.
     * @return the unique document ID.
     */
    public static String getDocumentIdByGovId(final TransactionExecutor txn,
 final String govId) {
        return SampleData.getDocumentId(txn, Constants.PERSON_TABLE_NAME,
 "GovId", govId);
    }

    @Override
    public String toString() {
        return "Person{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", dob=" + dob +
                ", govId='" + govId + '\'' +
                ", govIdType='" + govIdType + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

3. VehicleRegistration.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

import java.math.BigDecimal;
import java.time.LocalDate;

/**
 * Represents a vehicle registration, serializable to (and from) Ion.
 */
public final class VehicleRegistration implements RevisionData {

    private final String vin;
    private final String licensePlateNumber;
    private final String state;
    private final String city;
    private final BigDecimal pendingPenaltyTicketAmount;
    private final LocalDate validFromDate;
    private final LocalDate validToDate;
    private final Owners owners;

    @JsonCreator
    public VehicleRegistration(@JsonProperty("VIN") final String vin,
                               @JsonProperty("LicensePlateNumber") final String
 licensePlateNumber,
                               @JsonProperty("State") final String state,
                               @JsonProperty("City") final String city,
```

```
                                @JsonProperty("PendingPenaltyTicketAmount") final
BigDecimal pendingPenaltyTicketAmount,
                                @JsonProperty("ValidFromDate") final LocalDate
validFromDate,
                                @JsonProperty("ValidToDate") final LocalDate
validToDate,
                                @JsonProperty("Owners") final Owners owners) {
        this.vin = vin;
        this.licensePlateNumber = licensePlateNumber;
        this.state = state;
        this.city = city;
        this.pendingPenaltyTicketAmount = pendingPenaltyTicketAmount;
        this.validFromDate = validFromDate;
        this.validToDate = validToDate;
        this.owners = owners;
    }

    @JsonProperty("City")
    public String getCity() {
        return city;
    }

    @JsonProperty("LicensePlateNumber")
    public String getLicensePlateNumber() {
        return licensePlateNumber;
    }

    @JsonProperty("Owners")
    public Owners getOwners() {
        return owners;
    }

    @JsonProperty("PendingPenaltyTicketAmount")
    public BigDecimal getPendingPenaltyTicketAmount() {
        return pendingPenaltyTicketAmount;
    }

    @JsonProperty("State")
    public String getState() {
        return state;
    }

    @JsonProperty("ValidFromDate")
    @JsonSerialize(using = IonLocalDateSerializer.class)
```

```
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    public LocalDate getValidFromDate() {
        return validFromDate;
    }

    @JsonProperty("ValidToDate")
    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    public LocalDate getValidToDate() {
        return validToDate;
    }

    @JsonProperty("VIN")
    public String getVin() {
        return vin;
    }

    /**
     * Returns the unique document ID of a vehicle given a specific VIN.
     *
     * @param txn
     *              A transaction executor object.
     * @param vin
     *              The VIN of a vehicle.
     * @return the unique document ID of the specified vehicle.
     */
    public static String getDocumentIdByVin(final TransactionExecutor txn, final
String vin) {
        return SampleData.getDocumentId(txn,
Constants.VEHICLE_REGISTRATION_TABLE_NAME, "VIN", vin);
    }

    @Override
    public String toString() {
        return "VehicleRegistration{" +
                "vin='" + vin + '\'' +
                ", licensePlateNumber='" + licensePlateNumber + '\'' +
                ", state='" + state + '\'' +
                ", city='" + city + '\'' +
                ", pendingPenaltyTicketAmount=" + pendingPenaltyTicketAmount +
                ", validFromDate=" + validFromDate +
                ", validToDate=" + validToDate +
                ", owners=" + owners +
                '}';
```

```
    }
}
```

4. Vehicle.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

/**
 * Represents a vehicle, serializable to (and from) Ion.
 */
public final class Vehicle implements RevisionData {
    private final String vin;
    private final String type;
    private final int year;
    private final String make;
```

```
    private final String model;
    private final String color;

    @JsonCreator
    public Vehicle(@JsonProperty("VIN") final String vin,
                   @JsonProperty("Type") final String type,
                   @JsonProperty("Year") final int year,
                   @JsonProperty("Make") final String make,
                   @JsonProperty("Model") final String model,
                   @JsonProperty("Color") final String color) {
        this.vin = vin;
        this.type = type;
        this.year = year;
        this.make = make;
        this.model = model;
        this.color = color;
    }

    @JsonProperty("Color")
    public String getColor() {
        return color;
    }

    @JsonProperty("Make")
    public String getMake() {
        return make;
    }

    @JsonProperty("Model")
    public String getModel() {
        return model;
    }

    @JsonProperty("Type")
    public String getType() {
        return type;
    }

    @JsonProperty("VIN")
    public String getVin() {
        return vin;
    }

    @JsonProperty("Year")
```

```
    public int getYear() {
        return year;
    }

    @Override
    public String toString() {
        return "Vehicle{" +
                "vin='" + vin + '\'' +
                ", type='" + type + '\'' +
                ", year=" + year +
                ", make='" + make + '\'' +
                ", model='" + model + '\'' +
                ", color='" + color + '\'' +
                '}';
    }
}
```

5. Owner.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Represents a vehicle owner, serializable to (and from) Ion.
 */
public final class Owner {
    private final String personId;

    public Owner(@JsonProperty("PersonId") final String personId) {
        this.personId = personId;
    }

    @JsonProperty("PersonId")
    public String getPersonId() {
        return personId;
    }

    @Override
    public String toString() {
        return "Owner{" +
                "personId='" + personId + '\'' +
                '}';
    }
}
```

6. Owners.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonProperty;

import java.util.List;

/**
 * Represents a set of owners for a given vehicle, serializable to (and from)
 Ion.
 */
public final class Owners {
    private final Owner primaryOwner;
    private final List<Owner> secondaryOwners;

    public Owners(@JsonProperty("PrimaryOwner") final Owner primaryOwner,
                  @JsonProperty("SecondaryOwners") final List<Owner>
 secondaryOwners) {
        this.primaryOwner = primaryOwner;
        this.secondaryOwners = secondaryOwners;
    }

    @JsonProperty("PrimaryOwner")
    public Owner getPrimaryOwner() {
        return primaryOwner;
    }

    @JsonProperty("SecondaryOwners")
    public List<Owner> getSecondaryOwners() {
        return secondaryOwners;
    }

    @Override
    public String toString() {
```

```
        return "Owners{" +
                "primaryOwner=" + primaryOwner +
                ", secondaryOwners=" + secondaryOwners +
                '}';
    }
}
```

7. DmlResultDocument.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Contains information about an individual document inserted or modified
 * as a result of DML.
 */
public class DmlResultDocument {
```

```
    private String documentId;

    @JsonCreator
    public DmlResultDocument(@JsonProperty("documentId") final String documentId)
 {
        this.documentId = documentId;
    }

    public String getDocumentId() {
        return documentId;
    }

    @Override
    public String toString() {
        return "DmlResultDocument{"
            + "documentId='" + documentId + '\''
            + '}';
    }
}
```

8. RevisionData.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model.streams;

/**
 * Allows modeling the content of all revisions as a generic revision data. Used
 * in the {@link Revision} and extended by domain models in {@link
 * software.amazon.qldb.tutorial.model} to make it easier to write the {@link
 * Revision.RevisionDataDeserializer} that must deserialize the {@link
 * Revision#data} from different domain models.
 */
public interface RevisionData { }
```

9. RevisionMetadata.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;
```

```
import com.amazon.ion.IonInt;
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonTimestamp;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.dataformat.ion.IonTimestampSerializers;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Date;
import java.util.Objects;

/**
 * Represents the metadata field of a QLDB Document
 */
public class RevisionMetadata {
    private static final Logger log =
 LoggerFactory.getLogger(RevisionMetadata.class);
    private final String id;
    private final long version;
    @JsonSerialize(using =
 IonTimestampSerializers.IonTimestampJavaDateSerializer.class)
    private final Date txTime;
    private final String txId;

    @JsonCreator
    public RevisionMetadata(@JsonProperty("id") final String id,
                            @JsonProperty("version") final long version,
                            @JsonProperty("txTime") final Date txTime,
                            @JsonProperty("txId") final String txId) {
        this.id = id;
        this.version = version;
        this.txTime = txTime;
        this.txId = txId;
    }

    /**
     * Gets the unique ID of a QLDB document.
     *
     * @return the document ID.
     */
    public String getId() {
```

```
        return id;
    }

    /**
     * Gets the version number of the document in the document's modification
history.
     * @return the version number.
     */
    public long getVersion() {
        return version;
    }

    /**
     * Gets the time during which the document was modified.
     *
     * @return the transaction time.
     */
    public Date getTxTime() {
        return txTime;
    }

    /**
     * Gets the transaction ID associated with this document.
     *
     * @return the transaction ID.
     */
    public String getTxId() {
        return txId;
    }

    public static RevisionMetadata fromIon(final IonStruct ionStruct) {
        if (ionStruct == null) {
            throw new IllegalArgumentException("Metadata cannot be null");
        }
        try {
            IonString id = (IonString) ionStruct.get("id");
            IonInt version = (IonInt) ionStruct.get("version");
            IonTimestamp txTime = (IonTimestamp) ionStruct.get("txTime");
            IonString txId = (IonString) ionStruct.get("txId");
            if (id == null || version == null || txTime == null || txId == null)
{
                throw new IllegalArgumentException("Document is missing required
fields");
            }
```

```
            return new RevisionMetadata(id.stringValue(), version.longValue(),
new Date(txTime.getMillis()), txId.stringValue());
        } catch (ClassCastException e) {
            log.error("Failed to parse ion document");
            throw new IllegalArgumentException("Document members are not of the
correct type", e);
        }
    }

    /**
     * Converts a {@link RevisionMetadata} object to a string.
     *
     * @return the string representation of the {@link QldbRevision} object.
     */
    @Override
    public String toString() {
        return "Metadata{"
                + "id='" + id + '\''
                + ", version=" + version
                + ", txTime=" + txTime
                + ", txId='" + txId
                + '\''
                + '}';
    }

    /**
     * Check whether two {@link RevisionMetadata} objects are equivalent.
     *
     * @return {@code true} if the two objects are equal, {@code false}
otherwise.
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) { return true; }
        if (o == null || getClass() != o.getClass()) { return false; }
        RevisionMetadata metadata = (RevisionMetadata) o;
        return version == metadata.version
                && id.equals(metadata.id)
                && txTime.equals(metadata.txTime)
                && txId.equals(metadata.txId);
    }

    /**
     * Generate a hash code for the {@link RevisionMetadata} object.
```

```
     *
     * @return the hash code.
     */
    @Override
    public int hashCode() {
        // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
 properties.
        return Objects.hash(id, version, txTime, txId);
        // CHECKSTYLE:ON
    }
}
```

10QldbRevision.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonBlob;
import com.amazon.ion.IonStruct;
import com.fasterxml.jackson.annotation.JsonCreator;
```

```
import com.fasterxml.jackson.annotation.JsonProperty;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.Verifier;

import java.io.IOException;
import java.util.Arrays;
import java.util.Objects;

/**
 * Represents a QldbRevision including both user data and metadata.
 */
public final class QldbRevision {
    private static final Logger log =
 LoggerFactory.getLogger(QldbRevision.class);

    private final BlockAddress blockAddress;
    private final RevisionMetadata metadata;
    private final byte[] hash;
    private final byte[] dataHash;
    private final IonStruct data;

    @JsonCreator
    public QldbRevision(@JsonProperty("blockAddress") final BlockAddress
 blockAddress,
                        @JsonProperty("metadata") final RevisionMetadata
 metadata,
                        @JsonProperty("hash") final byte[] hash,
                        @JsonProperty("dataHash") final byte[] dataHash,
                        @JsonProperty("data") final IonStruct data) {
        this.blockAddress = blockAddress;
        this.metadata = metadata;
        this.hash = hash;
        this.dataHash = dataHash;
        this.data = data;
    }

    /**
     * Gets the unique ID of a QLDB document.
     *
     * @return the {@link BlockAddress} object.
     */
    public BlockAddress getBlockAddress() {
```

```
        return blockAddress;
    }

    /**
     * Gets the metadata of the revision.
     *
     * @return the {@link RevisionMetadata} object.
     */
    public RevisionMetadata getMetadata() {
        return metadata;
    }

    /**
     * Gets the SHA-256 hash value of the revision.
     * This is equivalent to the hash of the revision metadata and data.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getHash() {
        return hash;
    }

    /**
     * Gets the SHA-256 hash value of the data portion of the revision.
     * This is only present if the revision is redacted.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getDataHash() {
        return dataHash;
    }

    /**
     * Gets the revision data.
     *
     * @return the revision data.
     */
    public IonStruct getData() {
        return data;
    }

    /**
     * Returns true if the revision has been redacted.
     * @return a boolean value representing the redaction status
```

```
     * of this revision.
     */
    public Boolean isRedacted() {
        return dataHash != null;
    }

    /**
     * Constructs a new {@link QldbRevision} from an {@link IonStruct}.
     *
     * The specified {@link IonStruct} must include the following fields
     *
     * - blockAddress -- a {@link BlockAddress},
     * - metadata -- a {@link RevisionMetadata},
     * - hash -- the revision's hash calculated by QLDB,
     * - dataHash -- the user data's hash calculated by QLDB (only present if
revision is redacted),
     * - data -- an {@link IonStruct} containing user data in the document.
     *
     * If any of these fields are missing or are malformed, then throws {@link
IllegalArgumentException}.
     *
     * If the document hash calculated from the members of the specified {@link
IonStruct} does not match
     * the hash member of the {@link IonStruct} then throws {@link
IllegalArgumentException}.
     *
     * @param ionStruct
     *              The {@link IonStruct} that contains a {@link QldbRevision}
object.
     * @return the converted {@link QldbRevision} object.
     * @throws IOException if failed to parse parameter {@link IonStruct}.
     */
    public static QldbRevision fromIon(final IonStruct ionStruct) throws
IOException {
        try {
            BlockAddress blockAddress =
Constants.MAPPER.readValue(ionStruct.get("blockAddress"), BlockAddress.class);
            IonBlob revisionHash = (IonBlob) ionStruct.get("hash");
            IonStruct metadataStruct = (IonStruct) ionStruct.get("metadata");
            IonStruct data = ionStruct.get("data") == null ||
ionStruct.get("data").isNullValue() ?
                null : (IonStruct) ionStruct.get("data");
            IonBlob dataHash = ionStruct.get("dataHash") == null ||
ionStruct.get("dataHash").isNullValue() ?
```

```
                null : (IonBlob) ionStruct.get("dataHash");
            if (revisionHash == null || metadataStruct == null) {
                throw new IllegalArgumentException("Document is missing required
fields");
            }
            byte[] dataHashBytes = dataHash != null ? dataHash.getBytes() :
QldbIonUtils.hashIonValue(data);
            verifyRevisionHash(metadataStruct, dataHashBytes,
revisionHash.getBytes());
            RevisionMetadata metadata = RevisionMetadata.fromIon(metadataStruct);
            return new QldbRevision(
                    blockAddress,
                    metadata,
                    revisionHash.getBytes(),
                    dataHash != null ? dataHash.getBytes() : null,
                    data
            );
        } catch (ClassCastException e) {
            log.error("Failed to parse ion document");
            throw new IllegalArgumentException("Document members are not of the
correct type", e);
        }
    }

    /**
     * Converts a {@link QldbRevision} object to string.
     *
     * @return the string representation of the {@link QldbRevision} object.
     */
    @Override
    public String toString() {
        return "QldbRevision{" +
                "blockAddress=" + blockAddress +
                ", metadata=" + metadata +
                ", hash=" + Arrays.toString(hash) +
                ", dataHash=" + Arrays.toString(dataHash) +
                ", data=" + data +
                '}';
    }

    /**
     * Check whether two {@link QldbRevision} objects are equivalent.
     *
```

```
     * @return {@code true} if the two objects are equal, {@code false}
otherwise.
     */
    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof QldbRevision)) {
            return false;
        }
        final QldbRevision that = (QldbRevision) o;
        return Objects.equals(getBlockAddress(), that.getBlockAddress())
                && Objects.equals(getMetadata(), that.getMetadata())
                && Arrays.equals(getHash(), that.getHash())
                && Arrays.equals(getDataHash(), that.getDataHash())
                && Objects.equals(getData(), that.getData());
    }

    /**
     * Create a hash code for the {@link QldbRevision} object.
     *
     * @return the hash code.
     */
    @Override
    public int hashCode() {
        // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
properties.
        int result = Objects.hash(blockAddress, metadata, data);
        // CHECKSTYLE:ON
        result = 31 * result + Arrays.hashCode(hash);
        return result;
    }

    /**
     * Throws an IllegalArgumentException if the hash of the revision data and
metadata
     * does not match the hash provided by QLDB with the revision.
     */
    public void verifyRevisionHash() {
        // Certain internal-only system revisions only contain a hash which
cannot be
        // further computed. However, these system hashes still participate to
validate
```

```
        // the journal block. User revisions will always contain values for all
 fields
        // and can therefore have their hash computed.
        if (blockAddress == null && metadata == null && data == null && dataHash
 == null) {
            return;
        }

        try {
            IonStruct metadataIon = (IonStruct)
 Constants.MAPPER.writeValueAsIonValue(metadata);
            byte[] dataHashBytes = isRedacted() ? dataHash :
 QldbIonUtils.hashIonValue(data);
            verifyRevisionHash(metadataIon, dataHashBytes, hash);
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not encode revision
 metadata to ion.", e);
        }
    }

    private static void verifyRevisionHash(IonStruct metadata, byte[] dataHash,
 byte[] expectedHash) {
        byte[] metadataHash = QldbIonUtils.hashIonValue(metadata);
        byte[] candidateHash = Verifier.dot(metadataHash, dataHash);
        if (!Arrays.equals(candidateHash, expectedHash)) {
            throw new IllegalArgumentException("Hash entry of QLDB revision and
 computed hash "
                    + "of QLDB revision do not match");
        }
    }
}
```

11IonLocalDateDeserializer.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
```

```
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.JsonDeserializer;

import java.io.IOException;
import java.time.LocalDate;

/**
 * Deserializes [java.time.LocalDate] from Ion.
 */
public class IonLocalDateDeserializer extends JsonDeserializer<LocalDate> {

    @Override
    public LocalDate deserialize(JsonParser jp, DeserializationContext ctxt)
 throws IOException {
        return timestampToLocalDate((Timestamp) jp.getEmbeddedObject());
    }

    private LocalDate timestampToLocalDate(Timestamp timestamp) {
        return LocalDate.of(timestamp.getYear(), timestamp.getMonth(),
 timestamp.getDay());
    }
}
```

12IonLocalDateSerializer.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.SerializerProvider;
import com.fasterxml.jackson.databind.ser.std.StdScalarSerializer;
import com.fasterxml.jackson.dataformat.ion.IonGenerator;

import java.io.IOException;
import java.time.LocalDate;

/**
 * Serializes [java.time.LocalDate] to Ion.
 */
public class IonLocalDateSerializer extends StdScalarSerializer<LocalDate> {

    public IonLocalDateSerializer() {
        super(LocalDate.class);
```

```
    }

    @Override
    public void serialize(LocalDate date, JsonGenerator jsonGenerator,
 SerializerProvider serializerProvider) throws IOException {
        Timestamp timestamp = Timestamp.forDay(date.getYear(),
 date.getMonthValue(), date.getDayOfMonth());
        ((IonGenerator) jsonGenerator).writeValue(timestamp);
    }
}
```

2. 次のファイル (SampleData.java) を確認します。このファイルには、vehicle-registration テーブルに挿入するサンプルデータが含まれています。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;
```

```
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import java.io.IOException;
import java.math.BigDecimal;
import java.text.ParseException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.ConnectToLedger;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.qldb.DmlResultDocument;
import software.amazon.qldb.tutorial.qldb.QldbRevision;

/**
 * Sample domain objects for use throughout this tutorial.
 */
public final class SampleData {
    public static final DateTimeFormatter DATE_TIME_FORMAT =
 DateTimeFormatter.ofPattern("yyyy-MM-dd");

    public static final List<VehicleRegistration> REGISTRATIONS =
 Collections.unmodifiableList(Arrays.asList(
            new VehicleRegistration("1N4AL11D75C109151", "LEWISR261LL", "WA",
 "Seattle",
                    BigDecimal.valueOf(90.25), convertToLocalDate("2017-08-21"),
 convertToLocalDate("2020-05-11"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("KM8SRDHF6EU074761", "CA762X", "WA", "Kent",
                    BigDecimal.valueOf(130.75),
 convertToLocalDate("2017-09-14"), convertToLocalDate("2020-06-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("3HGGK5G53FM761765", "CD820Z", "WA",
 "Everett",
                    BigDecimal.valueOf(442.30),
 convertToLocalDate("2011-03-17"), convertToLocalDate("2021-03-24"),
                    new Owners(new Owner(null), Collections.emptyList())),
```

```
            new VehicleRegistration("1HVBBAANXWH544237", "LS477D", "WA",
"Tacoma",
                    BigDecimal.valueOf(42.20), convertToLocalDate("2011-10-26"),
convertToLocalDate("2023-09-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1C4RJFAG0FC625797", "TH393F", "WA",
"Olympia",
                    BigDecimal.valueOf(30.45), convertToLocalDate("2013-09-02"),
convertToLocalDate("2024-03-19"),
                    new Owners(new Owner(null), Collections.emptyList()))
    ));

    public static final List<Vehicle> VEHICLES =
Collections.unmodifiableList(Arrays.asList(
            new Vehicle("1N4AL11D75C109151", "Sedan", 2011,  "Audi", "A5",
"Silver"),
            new Vehicle("KM8SRDHF6EU074761", "Sedan", 2015, "Tesla", "Model S",
"Blue"),
            new Vehicle("3HGGK5G53FM761765", "Motorcycle", 2011, "Ducati",
"Monster 1200", "Yellow"),
            new Vehicle("1HVBBAANXWH544237", "Semi", 2009, "Ford", "F 150",
"Black"),
            new Vehicle("1C4RJFAG0FC625797", "Sedan", 2019, "Mercedes", "CLK
350", "White")
    ));

    public static final List<Person> PEOPLE =
Collections.unmodifiableList(Arrays.asList(
            new Person("Raul", "Lewis", convertToLocalDate("1963-08-19"),
                    "LEWISR261LL", "Driver License",  "1719 University Street,
Seattle, WA, 98109"),
            new Person("Brent", "Logan", convertToLocalDate("1967-07-03"),
                    "LOGANB486CG", "Driver License", "43 Stockert Hollow Road,
Everett, WA, 98203"),
            new Person("Alexis", "Pena", convertToLocalDate("1974-02-10"),
                    "744 849 301", "SSN", "4058 Melrose Street, Spokane Valley,
WA, 99206"),
            new Person("Melvin", "Parker", convertToLocalDate("1976-05-22"),
                    "P626-168-229-765", "Passport", "4362 Ryder Avenue, Seattle,
WA, 98101"),
            new Person("Salvatore", "Spencer", convertToLocalDate("1997-11-15"),
                    "S152-780-97-415-0", "Passport", "4450 Honeysuckle Lane,
Seattle, WA, 98101")
    ));
```

```
   public static final List<DriversLicense> LICENSES =
Collections.unmodifiableList(Arrays.asList(
           new DriversLicense(null, "LEWISR261LL", "Learner",
                   convertToLocalDate("2016-12-20"),
convertToLocalDate("2020-11-15")),
           new DriversLicense(null, "LOGANB486CG", "Probationary",
                   convertToLocalDate("2016-04-06"),
convertToLocalDate("2020-11-15")),
           new DriversLicense(null, "744 849 301", "Full",
                   convertToLocalDate("2017-12-06"),
convertToLocalDate("2022-10-15")),
           new DriversLicense(null, "P626-168-229-765", "Learner",
                   convertToLocalDate("2017-08-16"),
convertToLocalDate("2021-11-15")),
           new DriversLicense(null, "S152-780-97-415-0", "Probationary",
                   convertToLocalDate("2015-08-15"),
convertToLocalDate("2021-08-21"))
   ));

   private SampleData() { }

   /**
    * Converts a date string with the format 'yyyy-MM-dd' into a {@link
java.util.Date} object.
    *
    * @param date
    *              The date string to convert.
    * @return {@link java.time.LocalDate} or null if there is a {@link
ParseException}
    */
   public static synchronized LocalDate convertToLocalDate(String date) {
       return LocalDate.parse(date, DATE_TIME_FORMAT);
   }

   /**
    * Convert the result set into a list of IonValues.
    *
    * @param result
    *              The result set to convert.
    * @return a list of IonValues.
    */
   public static List<IonValue> toIonValues(Result result) {
       final List<IonValue> valueList = new ArrayList<>();
```

```
        result.iterator().forEachRemaining(valueList::add);
        return valueList;
    }

    /**
     * Get the document ID of a particular document.
     *
     * @param txn
     *              A transaction executor object.
     * @param tableName
     *              Name of the table containing the document.
     * @param identifier
     *              The identifier used to narrow down the search.
     * @param value
     *              Value of the identifier.
     * @return the list of document IDs in the result set.
     */
    public static String getDocumentId(final TransactionExecutor txn, final
String tableName,
                                       final String identifier, final String
value) {
        try {
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(value));
            final String query = String.format("SELECT metadata.id FROM
_ql_committed_%s AS p WHERE p.data.%s = ?",
                    tableName, identifier);
            Result result = txn.execute(query, parameters);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document ID
using " + value);
            }
            return getStringValueOfStructField((IonStruct)
result.iterator().next(), "id");
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the document by ID.
     *
     * @param tableName
     *              Name of the table to insert documents into.
```

```
    * @param documentId
    *              The unique ID of a document in the Person table.
    * @return a {@link QldbRevision} object.
    * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
    */
   public static QldbRevision getDocumentById(String tableName, String
documentId) {
       try {
           final IonValue ionValue =
Constants.MAPPER.writeValueAsIonValue(documentId);
           Result result = ConnectToLedger.getDriver().execute(txn -> {
               return txn.execute("SELECT c.* FROM _ql_committed_" + tableName
+ " AS c BY docId "
                                  + "WHERE docId = ?", ionValue);
           });
           if (result.isEmpty()) {
               throw new IllegalStateException("Unable to retrieve document by
id " + documentId + " in table " + tableName);
           }
           return Constants.MAPPER.readValue(result.iterator().next(),
QldbRevision.class);
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   /**
    * Return a list of modified document IDs as strings from a DML {@link
Result}.
    *
    * @param result
    *              The result set from a DML operation.
    * @return the list of document IDs modified by the operation.
    */
   public static List<String> getDocumentIdsFromDmlResult(final Result result)
{
       final List<String> strings = new ArrayList<>();
       result.iterator().forEachRemaining(row ->
strings.add(getDocumentIdFromDmlResultDocument(row)));
       return strings;
   }

   /**
```

```
    * Convert the given DML result row's document ID to string.
    *
    * @param dmlResultDocument
    *              The {@link IonValue} representing the results of a DML
operation.
    * @return a string of document ID.
    */
   public static String getDocumentIdFromDmlResultDocument(final IonValue
dmlResultDocument) {
       try {
           DmlResultDocument result =
Constants.MAPPER.readValue(dmlResultDocument, DmlResultDocument.class);
           return result.getDocumentId();
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   /**
    * Get the String value of a given {@link IonStruct} field name.
    * @param struct the {@link IonStruct} from which to get the value.
    * @param fieldName the name of the field from which to get the value.
    * @return the String value of the field within the given {@link IonStruct}.
    */
   public static String getStringValueOfStructField(final IonStruct struct,
final String fieldName) {
       return ((IonString) struct.get(fieldName)).stringValue();
   }

   /**
    * Return a copy of the given driver's license with updated person Id.
    *
    * @param oldLicense
    *              The old driver's license to update.
    * @param personId
    *              The PersonId of the driver.
    * @return the updated {@link DriversLicense}.
    */
   public static DriversLicense updatePersonIdDriversLicense(final
DriversLicense oldLicense, final String personId) {
       return new DriversLicense(personId, oldLicense.getLicenseNumber(),
oldLicense.getLicenseType(),
               oldLicense.getValidFromDate(), oldLicense.getValidToDate());
   }
```

```
    /**
     * Return a copy of the given vehicle registration with updated person Id.
     *
     * @param oldRegistration
     *              The old vehicle registration to update.
     * @param personId
     *              The PersonId of the driver.
     * @return the updated {@link VehicleRegistration}.
     */
    public static VehicleRegistration updateOwnerVehicleRegistration(final
 VehicleRegistration oldRegistration,
                                                                     final
 String personId) {
        return new VehicleRegistration(oldRegistration.getVin(),
 oldRegistration.getLicensePlateNumber(),
                oldRegistration.getState(), oldRegistration.getCity(),
 oldRegistration.getPendingPenaltyTicketAmount(),
                oldRegistration.getValidFromDate(),
 oldRegistration.getValidToDate(),
                new Owners(new Owner(personId), Collections.emptyList()));
    }
}
```

1.x

> **Important**
>
> Amazon Ion パッケージの場合、アプリケーションで名前空間 `com.amazon.ion` を使用する必要があります。AWS SDK for Java は、名前空間 `software.amazon.ion` の別の Ion パッケージに依存しますが、これは QLDB ドライバーと互換性がないレガシーパッケージです。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.qldb.DmlResultDocument;
import software.amazon.qldb.tutorial.qldb.QldbRevision;

import java.io.IOException;

import java.math.BigDecimal;
import java.text.ParseException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
```

```
 * Sample domain objects for use throughout this tutorial.
 */
public final class SampleData {
    public static final DateTimeFormatter DATE_TIME_FORMAT =
 DateTimeFormatter.ofPattern("yyyy-MM-dd");

    public static final List<VehicleRegistration> REGISTRATIONS =
 Collections.unmodifiableList(Arrays.asList(
            new VehicleRegistration("1N4AL11D75C109151", "LEWISR261LL", "WA",
 "Seattle",
                    BigDecimal.valueOf(90.25), convertToLocalDate("2017-08-21"),
 convertToLocalDate("2020-05-11"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("KM8SRDHF6EU074761", "CA762X", "WA", "Kent",
                    BigDecimal.valueOf(130.75),
 convertToLocalDate("2017-09-14"), convertToLocalDate("2020-06-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("3HGGK5G53FM761765", "CD820Z", "WA",
 "Everett",
                    BigDecimal.valueOf(442.30),
 convertToLocalDate("2011-03-17"), convertToLocalDate("2021-03-24"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1HVBBAANXWH544237", "LS477D", "WA",
 "Tacoma",
                    BigDecimal.valueOf(42.20), convertToLocalDate("2011-10-26"),
 convertToLocalDate("2023-09-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1C4RJFAG0FC625797", "TH393F", "WA",
 "Olympia",
                    BigDecimal.valueOf(30.45), convertToLocalDate("2013-09-02"),
 convertToLocalDate("2024-03-19"),
                    new Owners(new Owner(null), Collections.emptyList()))
    ));

    public static final List<Vehicle> VEHICLES =
 Collections.unmodifiableList(Arrays.asList(
            new Vehicle("1N4AL11D75C109151", "Sedan", 2011,  "Audi", "A5",
 "Silver"),
            new Vehicle("KM8SRDHF6EU074761", "Sedan", 2015, "Tesla", "Model S",
 "Blue"),
            new Vehicle("3HGGK5G53FM761765", "Motorcycle", 2011, "Ducati",
 "Monster 1200", "Yellow"),
            new Vehicle("1HVBBAANXWH544237", "Semi", 2009, "Ford", "F 150",
 "Black"),
```

```
            new Vehicle("1C4RJFAG0FC625797", "Sedan", 2019, "Mercedes", "CLK
350", "White")
    ));

    public static final List<Person> PEOPLE =
Collections.unmodifiableList(Arrays.asList(
            new Person("Raul", "Lewis", convertToLocalDate("1963-08-19"),
                    "LEWISR261LL", "Driver License",  "1719 University Street,
Seattle, WA, 98109"),
            new Person("Brent", "Logan", convertToLocalDate("1967-07-03"),
                    "LOGANB486CG", "Driver License", "43 Stockert Hollow Road,
Everett, WA, 98203"),
            new Person("Alexis", "Pena", convertToLocalDate("1974-02-10"),
                    "744 849 301", "SSN", "4058 Melrose Street, Spokane Valley,
WA, 99206"),
            new Person("Melvin", "Parker", convertToLocalDate("1976-05-22"),
                    "P626-168-229-765", "Passport", "4362 Ryder Avenue, Seattle,
WA, 98101"),
            new Person("Salvatore", "Spencer", convertToLocalDate("1997-11-15"),
                    "S152-780-97-415-0", "Passport", "4450 Honeysuckle Lane,
Seattle, WA, 98101")
    ));

    public static final List<DriversLicense> LICENSES =
Collections.unmodifiableList(Arrays.asList(
            new DriversLicense(null, "LEWISR261LL", "Learner",
                    convertToLocalDate("2016-12-20"),
convertToLocalDate("2020-11-15")),
            new DriversLicense(null, "LOGANB486CG", "Probationary",
                    convertToLocalDate("2016-04-06"),
convertToLocalDate("2020-11-15")),
            new DriversLicense(null, "744 849 301", "Full",
                    convertToLocalDate("2017-12-06"),
convertToLocalDate("2022-10-15")),
            new DriversLicense(null, "P626-168-229-765", "Learner",
                    convertToLocalDate("2017-08-16"),
convertToLocalDate("2021-11-15")),
            new DriversLicense(null, "S152-780-97-415-0", "Probationary",
                    convertToLocalDate("2015-08-15"),
convertToLocalDate("2021-08-21"))
    ));

    private SampleData() { }
```

```
    /**
     * Converts a date string with the format 'yyyy-MM-dd' into a {@link
java.util.Date} object.
     *
     * @param date
     *              The date string to convert.
     * @return {@link LocalDate} or null if there is a {@link ParseException}
     */
    public static synchronized LocalDate convertToLocalDate(String date) {
        return LocalDate.parse(date, DATE_TIME_FORMAT);
    }

    /**
     * Convert the result set into a list of IonValues.
     *
     * @param result
     *              The result set to convert.
     * @return a list of IonValues.
     */
    public static List<IonValue> toIonValues(Result result) {
        final List<IonValue> valueList = new ArrayList<>();
        result.iterator().forEachRemaining(valueList::add);
        return valueList;
    }

    /**
     * Get the document ID of a particular document.
     *
     * @param txn
     *              A transaction executor object.
     * @param tableName
     *              Name of the table containing the document.
     * @param identifier
     *              The identifier used to narrow down the search.
     * @param value
     *              Value of the identifier.
     * @return the list of document IDs in the result set.
     */
    public static String getDocumentId(final TransactionExecutor txn, final
String tableName,
                                       final String identifier, final String
value) {
        try {
```

```
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(value));
            final String query = String.format("SELECT metadata.id FROM
_ql_committed_%s AS p WHERE p.data.%s = ?",
                    tableName, identifier);
            Result result = txn.execute(query, parameters);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document ID
using " + value);
            }
            return getStringValueOfStructField((IonStruct)
result.iterator().next(), "id");
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the document by ID.
     *
     * @param qldbSession
     *              A QLDB session.
     * @param tableName
     *              Name of the table to insert documents into.
     * @param documentId
     *              The unique ID of a document in the Person table.
     * @return a {@link QldbRevision} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
     */
    public static QldbRevision getDocumentById(QldbSession qldbSession, String
tableName, String documentId) {
        try {
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(documentId));
            final String query = String.format("SELECT c.* FROM _ql_committed_%s
AS c BY docId WHERE docId = ?", tableName);
            Result result = qldbSession.execute(query, parameters);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document by
id " + documentId + " in table " + tableName);
            }
            return Constants.MAPPER.readValue(result.iterator().next(),
QldbRevision.class);
```

```
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Return a list of modified document IDs as strings from a DML {@link
Result}.
     *
     * @param result
     *              The result set from a DML operation.
     * @return the list of document IDs modified by the operation.
     */
    public static List<String> getDocumentIdsFromDmlResult(final Result result)
{
        final List<String> strings = new ArrayList<>();
        result.iterator().forEachRemaining(row ->
strings.add(getDocumentIdFromDmlResultDocument(row)));
        return strings;
    }

    /**
     * Convert the given DML result row's document ID to string.
     *
     * @param dmlResultDocument
     *              The {@link IonValue} representing the results of a DML
operation.
     * @return a string of document ID.
     */
    public static String getDocumentIdFromDmlResultDocument(final IonValue
dmlResultDocument) {
        try {
            DmlResultDocument result =
Constants.MAPPER.readValue(dmlResultDocument, DmlResultDocument.class);
            return result.getDocumentId();
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the String value of a given {@link IonStruct} field name.
     * @param struct the {@link IonStruct} from which to get the value.
     * @param fieldName the name of the field from which to get the value.
```

```
     * @return the String value of the field within the given {@link IonStruct}.
     */
    public static String getStringValueOfStructField(final IonStruct struct,
final String fieldName) {
        return ((IonString) struct.get(fieldName)).stringValue();
    }

    /**
     * Return a copy of the given driver's license with updated person Id.
     *
     * @param oldLicense
     *              The old driver's license to update.
     * @param personId
     *              The PersonId of the driver.
     * @return the updated {@link DriversLicense}.
     */
    public static DriversLicense updatePersonIdDriversLicense(final
DriversLicense oldLicense, final String personId) {
        return new DriversLicense(personId, oldLicense.getLicenseNumber(),
oldLicense.getLicenseType(),
                oldLicense.getValidFromDate(), oldLicense.getValidToDate());
    }

    /**
     * Return a copy of the given vehicle registration with updated person Id.
     *
     * @param oldRegistration
     *              The old vehicle registration to update.
     * @param personId
     *              The PersonId of the driver.
     * @return the updated {@link VehicleRegistration}.
     */
    public static VehicleRegistration updateOwnerVehicleRegistration(final
VehicleRegistration oldRegistration,
                                                                     final
String personId) {
        return new VehicleRegistration(oldRegistration.getVin(),
oldRegistration.getLicensePlateNumber(),
                oldRegistration.getState(), oldRegistration.getCity(),
oldRegistration.getPendingPenaltyTicketAmount(),
                oldRegistration.getValidFromDate(),
oldRegistration.getValidToDate(),
                new Owners(new Owner(personId), Collections.emptyList()));
    }
```

```
}
```

> **Note**
>
> - このクラスは Ion ライブラリを使用して、データを Ion 形式との間で変換するヘルパーメソッドを提供します。
> - `getDocumentId` メソッドは、プレフィックス `_ql_committed_` を持つテーブルに対してクエリを実行します。これは、テーブルのコミット済みビューにクエリを実行することを示す予約済みプレフィックスです。このビューでは、データは `data` フィールドにネストされており、メタデータは `metadata` フィールドにネストされています。

3. 次のプログラム (`CreateTable.java`) をコンパイルして実行し、前述のテーブルを作成します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create tables in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateTable {
    public static final Logger log = LoggerFactory.getLogger(CreateTable.class);

    private CreateTable() { }

    /**
     * Registrations, vehicles, owners, and licenses tables being created in a
 single transaction.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @return the number of tables created.
     */
    public static int createTable(final TransactionExecutor txn, final String
 tableName) {
        log.info("Creating the '{}' table...", tableName);
        final String createTable = String.format("CREATE TABLE %s", tableName);
        final Result result = txn.execute(createTable);
        log.info("{} table created successfully.", tableName);
        return SampleData.toIonValues(result).size();
    }
```

```
    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createTable(txn, Constants.DRIVERS_LICENSE_TABLE_NAME);
            createTable(txn, Constants.PERSON_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        });
    }
}
```

1.x

```
/*
 * Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create tables in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateTable {
    public static final Logger log = LoggerFactory.getLogger(CreateTable.class);

    private CreateTable() { }

    /**
     * Registrations, vehicles, owners, and licenses tables being created in a
 single transaction.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @return the number of tables created.
     */
    public static int createTable(final TransactionExecutor txn, final String
 tableName) {
        log.info("Creating the '{}' table...", tableName);
        final String createTable = String.format("CREATE TABLE %s", tableName);
        final Result result = txn.execute(createTable);
        log.info("{} table created successfully.", tableName);
        return SampleData.toIonValues(result).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createTable(txn, Constants.DRIVERS_LICENSE_TABLE_NAME);
            createTable(txn, Constants.PERSON_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
```

```
}
```

> **Note**
>
> このプログラムは、TransactionExecutor Lambda を execute メソッドに渡す方法を示します。この例では、Lambda 式を使用して 1 つのトランザクションで複数の CREATE TABLE PartiQL ステートメントを実行します。
> この execute メソッドは、暗黙的にトランザクションを開始し、Lambda ですべてのステートメントを実行して、トランザクションを自動コミットします。

4. 前述のように、次のプログラム (CreateIndex.java) をコンパイルし実行して、テーブルにインデックスを作成します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create indexes on tables in a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateIndex {
    public static final Logger log = LoggerFactory.getLogger(CreateIndex.class);

    private CreateIndex() { }

    /**
     * In this example, create indexes for registrations and vehicles tables.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @param indexAttribute
     *              The index attribute to use.
     * @return the number of tables created.
     */
    public static int createIndex(final TransactionExecutor txn, final String
 tableName, final String indexAttribute) {
        log.info("Creating an index on {}...", indexAttribute);
        final String createIndex = String.format("CREATE INDEX ON %s (%s)",
 tableName, indexAttribute);
        final Result r = txn.execute(createIndex);
        return SampleData.toIonValues(r).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
```

```
            createIndex(txn, Constants.PERSON_TABLE_NAME,
 Constants.PERSON_GOV_ID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_NUMBER_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_PERSONID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME);
        });
        log.info("Indexes created successfully!");
    }
}
```

1.x

```
/*
 * Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
```

```
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create indexes on tables in a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateIndex {
    public static final Logger log = LoggerFactory.getLogger(CreateIndex.class);

    private CreateIndex() { }

    /**
     * In this example, create indexes for registrations and vehicles tables.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @param indexAttribute
     *              The index attribute to use.
     * @return the number of tables created.
     */
    public static int createIndex(final TransactionExecutor txn, final String
 tableName, final String indexAttribute) {
        log.info("Creating an index on {}...", indexAttribute);
        final String createIndex = String.format("CREATE INDEX ON %s (%s)",
 tableName, indexAttribute);
        final Result r = txn.execute(createIndex);
        return SampleData.toIonValues(r).size();
    }
```

```
    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createIndex(txn, Constants.PERSON_TABLE_NAME,
 Constants.PERSON_GOV_ID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_NUMBER_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_PERSONID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Indexes created successfully!");
    }
}
```

5. 次のプログラム (InsertDocument.java) をコンパイルし実行して、サンプルデータをテーブルに挿入します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.DriversLicense;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

/**
 * Insert documents into a table in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class InsertDocument {
    public static final Logger log =
 LoggerFactory.getLogger(InsertDocument.class);

    private InsertDocument() { }

    /**
     * Insert the given list of documents into the specified table and return
 the document IDs of the inserted documents.
     *
     * @param txn
```

```
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to insert documents into.
     * @param documents
     *              List of documents to insert into the specified table.
     * @return a list of document IDs.
     * @throws IllegalStateException if failed to convert documents into an
{@link IonValue}.
     */
    public static List<String> insertDocuments(final TransactionExecutor txn,
final String tableName,
                                               final List documents) {
        log.info("Inserting some documents in the {} table...", tableName);
        try {
            final String query = String.format("INSERT INTO %s ?", tableName);
            final IonValue ionDocuments =
Constants.MAPPER.writeValueAsIonValue(documents);

            return SampleData.getDocumentIdsFromDmlResult(txn.execute(query,
ionDocuments));
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Update PersonIds in driver's licenses and in vehicle registrations using
document IDs.
     *
     * @param documentIds
     *              List of document IDs representing the PersonIds in
DriversLicense and PrimaryOwners in VehicleRegistration.
     * @param licenses
     *              List of driver's licenses to update.
     * @param registrations
     *              List of registrations to update.
     */
    public static void updatePersonId(final List<String> documentIds, final
List<DriversLicense> licenses,
                                      final List<VehicleRegistration>
registrations) {
        for (int i = 0; i < documentIds.size(); ++i) {
            DriversLicense license = SampleData.LICENSES.get(i);
            VehicleRegistration registration = SampleData.REGISTRATIONS.get(i);
```

```
            licenses.add(SampleData.updatePersonIdDriversLicense(license,
 documentIds.get(i)));

 registrations.add(SampleData.updateOwnerVehicleRegistration(registration,
 documentIds.get(i)));
        }
    }

    public static void main(final String... args) {
        final List<DriversLicense> newDriversLicenses = new ArrayList<>();
        final List<VehicleRegistration> newVehicleRegistrations = new
 ArrayList<>();
        ConnectToLedger.getDriver().execute(txn -> {
            List<String> documentIds = insertDocuments(txn,
 Constants.PERSON_TABLE_NAME, SampleData.PEOPLE);
            updatePersonId(documentIds, newDriversLicenses,
 newVehicleRegistrations);
            insertDocuments(txn, Constants.VEHICLE_TABLE_NAME,
 SampleData.VEHICLES);
            insertDocuments(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
                    Collections.unmodifiableList(newVehicleRegistrations));
            insertDocuments(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
                    Collections.unmodifiableList(newDriversLicenses));
        });
        log.info("Documents inserted successfully!");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
```

```
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonValue;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.DriversLicense;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Insert documents into a table in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class InsertDocument {
    public static final Logger log =
 LoggerFactory.getLogger(InsertDocument.class);

    private InsertDocument() { }

    /**
```

```
    * Insert the given list of documents into the specified table and return
the the document IDs of the inserted documents.
    *
    * @param txn
    *              The {@link TransactionExecutor} for lambda execute.
    * @param tableName
    *              Name of the table to insert documents into.
    * @param documents
    *              List of documents to insert into the specified table.
    * @return a list of document IDs.
    * @throws IllegalStateException if failed to convert documents into an
{@link IonValue}.
    */
   public static List<String> insertDocuments(final TransactionExecutor txn,
final String tableName,
                                              final List documents) {
       log.info("Inserting some documents in the {} table...", tableName);
       try {
           final String statement = String.format("INSERT INTO %s ?",
tableName);
           final IonValue ionDocuments =
Constants.MAPPER.writeValueAsIonValue(documents);
           final List<IonValue> parameters =
Collections.singletonList(ionDocuments);
           return SampleData.getDocumentIdsFromDmlResult(txn.execute(statement,
parameters));
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   /**
    * Update PersonIds in driver's licenses and in vehicle registrations using
document IDs.
    *
    * @param documentIds
    *              List of document IDs representing the PersonIds in
DriversLicense and PrimaryOwners in VehicleRegistration.
    * @param licenses
    *              List of driver's licenses to update.
    * @param registrations
    *              List of registrations to update.
    */
```

```
    public static void updatePersonId(final List<String> documentIds, final
 List<DriversLicense> licenses,
                                      final List<VehicleRegistration>
 registrations) {
        for (int i = 0; i < documentIds.size(); ++i) {
            DriversLicense license = SampleData.LICENSES.get(i);
            VehicleRegistration registration = SampleData.REGISTRATIONS.get(i);
            licenses.add(SampleData.updatePersonIdDriversLicense(license,
 documentIds.get(i)));

 registrations.add(SampleData.updateOwnerVehicleRegistration(registration,
 documentIds.get(i)));
        }
    }

    public static void main(final String... args) {
        final List<DriversLicense> newDriversLicenses = new ArrayList<>();
        final List<VehicleRegistration> newVehicleRegistrations = new
 ArrayList<>();
        ConnectToLedger.getDriver().execute(txn -> {
            List<String> documentIds = insertDocuments(txn,
 Constants.PERSON_TABLE_NAME, SampleData.PEOPLE);
            updatePersonId(documentIds, newDriversLicenses,
 newVehicleRegistrations);
            insertDocuments(txn, Constants.VEHICLE_TABLE_NAME,
 SampleData.VEHICLES);
            insertDocuments(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
                    Collections.unmodifiableList(newVehicleRegistrations));
            insertDocuments(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
                    Collections.unmodifiableList(newDriversLicenses));
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Documents inserted successfully!");
    }
}
```

> **Note**
>
> - このプログラムは、パラメータ化された値を渡して execute メソッドを呼び出す方法を示します。実行する PartiQL ステートメントに加えて、IonValue 型のデータパラメータを渡すことができます。ステートメント文字列の変数プレースホルダーとして疑問符 (?) を使用します。

- INSERT ステートメントが成功すると、 挿入された各ドキュメントの id が返されます。

次に、SELECT ステートメントを使用すると、vehicle-registration 台帳のテーブルからデータを読み取ることができます。ステップ 4: 台帳のテーブルにクエリを実行する に進みます。

ステップ 4: 台帳のテーブルにクエリを実行する

Amazon QLDB 台帳にテーブルを作成し、データをロードした後は、クエリを実行して、挿入した車両登録データを確認できます。QLDB は PartiQL をクエリ言語として使用し、Amazon Ion をドキュメント指向のデータモデルとして使用します。

PartiQL は、Ion で動作するように拡張されたオープンソースの SQL 互換のクエリ言語です。PartiQL を使用すると、使い慣れた SQL 演算子を使用してデータを挿入、クエリ、および管理できます。Amazon Ion は JSON のスーパーセットです。Ion はオープンソースのドキュメントベースのデータ形式であり、構造化データ、半構造化データ、およびネストされたデータを柔軟に保存および処理できます。

このステップでは、SELECT ステートメントを使用して、vehicle-registration 台帳のテーブルからデータを読み取ります。

> **Warning**
>
> インデックス付きルックアップなしで QLDB でクエリを実行すると、完全なテーブルスキャンが呼び出されます。PartiQL は SQL 互換であるため、このようなクエリをサポートしています。ただし、QLDB の本番環境のユースケースではテーブルスキャンを実行しないでください。テーブルスキャンより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する必要があります (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`)。詳細については、「クエリパフォーマンスの最適化」を参照してください。

テーブルのクエリを実行するには

- 次のプログラム (FindVehicles.java) をコンパイルし実行して、台帳の人物に登録されているすべての車両のクエリを実行します。

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
```

```
 * Find all vehicles registered under a person.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class FindVehicles {
    public static final Logger log =
 LoggerFactory.getLogger(FindVehicles.class);

    private FindVehicles() { }

    /**
     * Find vehicles registered under a driver using their government ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param govId
     *              The government ID of the owner.
     * @throws IllegalStateException if failed to convert parameters into {@link
 IonValue}.
     */
    public static void findVehiclesForOwner(final TransactionExecutor txn, final
 String govId) {
        try {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            final String query = "SELECT v FROM Vehicle AS v INNER JOIN
 VehicleRegistration AS r "
                    + "ON v.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId
 = ?";

            final Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(documentId));
            log.info("List of Vehicles for owner with GovId: {}...", govId);
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final Person person = SampleData.PEOPLE.get(0);
        ConnectToLedger.getDriver().execute(txn -> {
            findVehiclesForOwner(txn, person.getGovId());
```

```
        });
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
```

```
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find all vehicles registered under a person.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class FindVehicles {
    public static final Logger log =
 LoggerFactory.getLogger(FindVehicles.class);

    private FindVehicles() { }

    /**
     * Find vehicles registered under a driver using their government ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param govId
     *              The government ID of the owner.
     * @throws IllegalStateException if failed to convert parameters into {@link
 IonValue}.
     */
    public static void findVehiclesForOwner(final TransactionExecutor txn, final
 String govId) {
        try {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            final String query = "SELECT v FROM Vehicle AS v INNER JOIN
 VehicleRegistration AS r "
                    + "ON v.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId
 = ?";

            final Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(documentId));
            log.info("List of Vehicles for owner with GovId: {}...", govId);
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
```

```
    public static void main(final String... args) {
        final Person person = SampleData.PEOPLE.get(0);
        ConnectToLedger.getDriver().execute(txn -> {
            findVehiclesForOwner(txn, person.getGovId());
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
}
```

> **Note**
>
> まず、このプログラムでは、GovId LEWISR261LL を使用してドキュメントの Person テーブルに対してクエリを実行し、id メタデータフィールドを取得します。
> 次に、このドキュメント id を外部キーとして使用して、PrimaryOwner.PersonId によって VehicleRegistration テーブルをクエリします。また、VehicleRegistration が VIN フィールドの Vehicle テーブルと結合されます。

vehicle-registration 台帳のテーブルのドキュメントの変更方法については、「ステップ 5: 台帳内のドキュメントを変更する」を参照してください。

ステップ 5: 台帳内のドキュメントを変更する

操作するデータを用意できたところで、Amazon QLDB で vehicle-registration 台帳のドキュメントに変更を加えることができます。このステップでは、次のコード例により、データ操作言語 (DML) ステートメントを実行する方法を示します。これらのステートメントは、ある車両の主所有者を更新し、別の車両に第二所有者を追加します。

ドキュメントに変更を加えるには

1. 次のプログラム (TransferVehicleOwnership.java) をコンパイルおよび実行して、台帳で VIN が 1N4AL11D75C109151 の車両の主所有者を更新します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonReaderBuilder;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;
```

```
/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class TransferVehicleOwnership {
    public static final Logger log =
 LoggerFactory.getLogger(TransferVehicleOwnership.class);

    private TransferVehicleOwnership() { }

    /**
     * Query a driver's information using the given ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param documentId
     *              The unique ID of a document in the Person table.
     * @return a {@link Person} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
 IonValue}.
     */
    public static Person findPersonFromDocumentId(final TransactionExecutor txn,
 final String documentId) {
        try {
            log.info("Finding person for documentId: {}...", documentId);
            final String query = "SELECT p.* FROM Person AS p BY pid WHERE pid
 = ?";

            Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(documentId));
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to find person with ID:
 " + documentId);
            }

            return Constants.MAPPER.readValue(result.iterator().next(),
 Person.class);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
```

```
    }

    /**
     * Find the primary owner for the given VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @return a {@link Person} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
     */
    public static Person findPrimaryOwnerForVehicle(final TransactionExecutor
txn, final String vin) {
        try {
            log.info("Finding primary owner for vehicle with Vin: {}...", vin);
            final String query = "SELECT Owners.PrimaryOwner.PersonId FROM
VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            Result result = txn.execute(query, parameters);
            final List<IonStruct> documents = ScanTable.toIonStructs(result);
            ScanTable.printDocuments(documents);
            if (documents.isEmpty()) {
                throw new IllegalStateException("Unable to find registrations
with VIN: " + vin);
            }

            final IonReader reader =
IonReaderBuilder.standard().build(documents.get(0));
            final String personId = Constants.MAPPER.readValue(reader,
LinkedHashMap.class).get("PersonId").toString();
            return findPersonFromDocumentId(txn, personId);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Update the primary owner for a vehicle registration with the given
documentId.
     *
     * @param txn
```

```
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param documentId
     *              New PersonId for the primary owner.
     * @throws IllegalStateException if no vehicle registration was found using
the given document ID and VIN, or if failed
     * to convert parameters into {@link IonValue}.
     */
    public static void updateVehicleRegistration(final TransactionExecutor txn,
final String vin, final String documentId) {
        try {
            log.info("Updating primary owner for vehicle with Vin: {}...", vin);
            final String query = "UPDATE VehicleRegistration AS v SET
v.Owners.PrimaryOwner = ? WHERE v.VIN = ?";

            final List<IonValue> parameters = new ArrayList<>();
            parameters.add(Constants.MAPPER.writeValueAsIonValue(new
Owner(documentId)));
            parameters.add(Constants.MAPPER.writeValueAsIonValue(vin));

            Result result = txn.execute(query, parameters);
            ScanTable.printDocuments(result);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to transfer vehicle,
could not find registration.");
            } else {
                log.info("Successfully transferred vehicle with VIN '{}' to new
owner.", vin);
            }
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String vin = SampleData.VEHICLES.get(0).getVin();
        final String primaryOwnerGovId = SampleData.PEOPLE.get(0).getGovId();
        final String newPrimaryOwnerGovId = SampleData.PEOPLE.get(1).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final Person primaryOwner = findPrimaryOwnerForVehicle(txn, vin);
            if (!primaryOwner.getGovId().equals(primaryOwnerGovId)) {
                // Verify the primary owner.
```

```
                throw new IllegalStateException("Incorrect primary owner
 identified for vehicle, unable to transfer.");
            }

            final String newOwner = Person.getDocumentIdByGovId(txn,
 newPrimaryOwnerGovId);
            updateVehicleRegistration(txn, vin, newOwner);
        });
        log.info("Successfully transferred vehicle ownership!");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
```

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonReaderBuilder;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class TransferVehicleOwnership {
    public static final Logger log =
 LoggerFactory.getLogger(TransferVehicleOwnership.class);

    private TransferVehicleOwnership() { }

    /**
     * Query a driver's information using the given ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param documentId
     *              The unique ID of a document in the Person table.
     * @return a {@link Person} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
 IonValue}.
     */
```

```
   public static Person findPersonFromDocumentId(final TransactionExecutor txn,
final String documentId) {
       try {
           log.info("Finding person for documentId: {}...", documentId);
           final String query = "SELECT p.* FROM Person AS p BY pid WHERE pid
= ?";

           Result result = txn.execute(query,
Constants.MAPPER.writeValueAsIonValue(documentId));
           if (result.isEmpty()) {
               throw new IllegalStateException("Unable to find person with ID:
" + documentId);
           }

           return Constants.MAPPER.readValue(result.iterator().next(),
Person.class);
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   /**
    * Find the primary owner for the given VIN.
    *
    * @param txn
    *              The {@link TransactionExecutor} for lambda execute.
    * @param vin
    *              Unique VIN for a vehicle.
    * @return a {@link Person} object.
    * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
    */
   public static Person findPrimaryOwnerForVehicle(final TransactionExecutor
txn, final String vin) {
       try {
           log.info("Finding primary owner for vehicle with Vin: {}...", vin);
           final String query = "SELECT Owners.PrimaryOwner.PersonId FROM
VehicleRegistration AS v WHERE v.VIN = ?";
           final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
           Result result = txn.execute(query, parameters);
           final List<IonStruct> documents = ScanTable.toIonStructs(result);
           ScanTable.printDocuments(documents);
           if (documents.isEmpty()) {
```

```
                throw new IllegalStateException("Unable to find registrations
with VIN: " + vin);
            }

            final IonReader reader =
IonReaderBuilder.standard().build(documents.get(0));
            final String personId = Constants.MAPPER.readValue(reader,
LinkedHashMap.class).get("PersonId").toString();
            return findPersonFromDocumentId(txn, personId);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Update the primary owner for a vehicle registration with the given
documentId.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param documentId
     *              New PersonId for the primary owner.
     * @throws IllegalStateException if no vehicle registration was found using
the given document ID and VIN, or if failed
     * to convert parameters into {@link IonValue}.
     */
    public static void updateVehicleRegistration(final TransactionExecutor txn,
final String vin, final String documentId) {
        try {
            log.info("Updating primary owner for vehicle with Vin: {}...", vin);
            final String query = "UPDATE VehicleRegistration AS v SET
v.Owners.PrimaryOwner = ? WHERE v.VIN = ?";

            final List<IonValue> parameters = new ArrayList<>();
            parameters.add(Constants.MAPPER.writeValueAsIonValue(new
Owner(documentId)));
            parameters.add(Constants.MAPPER.writeValueAsIonValue(vin));

            Result result = txn.execute(query, parameters);
            ScanTable.printDocuments(result);
            if (result.isEmpty()) {
```

```
                throw new IllegalStateException("Unable to transfer vehicle,
 could not find registration.");
            } else {
                log.info("Successfully transferred vehicle with VIN '{}' to new
 owner.", vin);
            }
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String vin = SampleData.VEHICLES.get(0).getVin();
        final String primaryOwnerGovId = SampleData.PEOPLE.get(0).getGovId();
        final String newPrimaryOwnerGovId = SampleData.PEOPLE.get(1).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final Person primaryOwner = findPrimaryOwnerForVehicle(txn, vin);
            if (!primaryOwner.getGovId().equals(primaryOwnerGovId)) {
                // Verify the primary owner.
                throw new IllegalStateException("Incorrect primary owner
 identified for vehicle, unable to transfer.");
            }

            final String newOwner = Person.getDocumentIdByGovId(txn,
 newPrimaryOwnerGovId);
            updateVehicleRegistration(txn, vin, newOwner);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Successfully transferred vehicle ownership!");
    }
}
```

2. 次のプログラム (AddSecondaryOwner.java) をコンパイルおよび実行して、台帳で VIN が KM8SRDHF6EU074761 の車両に第二所有者を追加します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Owners;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Finds and adds secondary owners for a vehicle.
 *
 * This code expects that you have AWS credentials setup per:
```

```
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class AddSecondaryOwner {
    public static final Logger log =
 LoggerFactory.getLogger(AddSecondaryOwner.class);

    private AddSecondaryOwner() { }

    /**
     * Check whether a secondary owner has already been registered for the given
 VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param secondaryOwnerId
     *              The secondary owner to add.
     * @return {@code true} if the given secondary owner has already been
 registered, {@code false} otherwise.
     * @throws IllegalStateException if failed to convert VIN to an {@link
 IonValue}.
     */
    public static boolean isSecondaryOwnerForVehicle(final TransactionExecutor
 txn, final String vin,
                                                     final String
 secondaryOwnerId) {
        try {
            log.info("Finding secondary owners for vehicle with VIN: {}...",
 vin);
            final String query = "SELECT Owners.SecondaryOwners FROM
 VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            final Iterator<IonValue> itr = result.iterator();
            if (!itr.hasNext()) {
                return false;
            }

            final Owners owners = Constants.MAPPER.readValue(itr.next(),
 Owners.class);
            if (null != owners.getSecondaryOwners()) {
```

```
                for (Owner owner : owners.getSecondaryOwners()) {
                    if (secondaryOwnerId.equalsIgnoreCase(owner.getPersonId()))
{
                        return true;
                    }
                }
            }

            return false;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Adds a secondary owner for the specified VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param secondaryOwner
     *              The secondary owner to add.
     * @throws IllegalStateException if failed to convert parameter into an
{@link IonValue}.
     */
    public static void addSecondaryOwnerForVin(final TransactionExecutor txn,
final String vin,
                                               final String secondaryOwner) {
        try {
            log.info("Inserting secondary owner for vehicle with VIN: {}...",
vin);
            final String query = String.format("FROM VehicleRegistration AS v
WHERE v.VIN = ?" +
                    "INSERT INTO v.Owners.SecondaryOwners VALUE ?");
            final IonValue newOwner = Constants.MAPPER.writeValueAsIonValue(new
Owner(secondaryOwner));
            final IonValue vinAsIonValue =
Constants.MAPPER.writeValueAsIonValue(vin);
            Result result = txn.execute(query, vinAsIonValue, newOwner);
            log.info("VehicleRegistration Document IDs which had secondary
owners added: ");
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
```

```
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String vin = SampleData.VEHICLES.get(1).getVin();
        final String govId = SampleData.PEOPLE.get(0).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            if (isSecondaryOwnerForVehicle(txn, vin, documentId)) {
                log.info("Person with ID {} has already been added as a
 secondary owner of this vehicle.", govId);
            } else {
                addSecondaryOwnerForVin(txn, vin, documentId);
            }
        });
        log.info("Secondary owners successfully updated.");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Owners;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Finds and adds secondary owners for a vehicle.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class AddSecondaryOwner {
    public static final Logger log =
 LoggerFactory.getLogger(AddSecondaryOwner.class);

    private AddSecondaryOwner() { }

    /**
     * Check whether a secondary owner has already been registered for the given
 VIN.
     *
     * @param txn
```

```
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param secondaryOwnerId
     *              The secondary owner to add.
     * @return {@code true} if the given secondary owner has already been
registered, {@code false} otherwise.
     * @throws IllegalStateException if failed to convert VIN to an {@link
IonValue}.
     */
    public static boolean isSecondaryOwnerForVehicle(final TransactionExecutor
txn, final String vin,
                                                     final String
secondaryOwnerId) {
        try {
            log.info("Finding secondary owners for vehicle with VIN: {}...",
vin);
            final String query = "SELECT Owners.SecondaryOwners FROM
VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            final Iterator<IonValue> itr = result.iterator();
            if (!itr.hasNext()) {
                return false;
            }

            final Owners owners = Constants.MAPPER.readValue(itr.next(),
Owners.class);
            if (null != owners.getSecondaryOwners()) {
                for (Owner owner : owners.getSecondaryOwners()) {
                    if (secondaryOwnerId.equalsIgnoreCase(owner.getPersonId()))
{
                        return true;
                    }
                }
            }

            return false;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
```

```
    /**
     * Adds a secondary owner for the specified VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param secondaryOwner
     *              The secondary owner to add.
     * @throws IllegalStateException if failed to convert parameter into an
{@link IonValue}.
     */
    public static void addSecondaryOwnerForVin(final TransactionExecutor txn,
final String vin,
                                               final String secondaryOwner) {
        try {
            log.info("Inserting secondary owner for vehicle with VIN: {}...",
vin);
            final String query = String.format("FROM VehicleRegistration AS v
WHERE v.VIN = '%s' " +
                    "INSERT INTO v.Owners.SecondaryOwners VALUE ?", vin);
            final IonValue newOwner = Constants.MAPPER.writeValueAsIonValue(new
Owner(secondaryOwner));
            Result result = txn.execute(query, newOwner);
            log.info("VehicleRegistration Document IDs which had secondary
owners added: ");
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String vin = SampleData.VEHICLES.get(1).getVin();
        final String govId = SampleData.PEOPLE.get(0).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            if (isSecondaryOwnerForVehicle(txn, vin, documentId)) {
                log.info("Person with ID {} has already been added as a
secondary owner of this vehicle.", govId);
            } else {
                addSecondaryOwnerForVin(txn, vin, documentId);
            }
```

```
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Secondary owners successfully updated.");
    }
}
```

これらの変更を vehicle-registration 台帳で確認するには、「ステップ 6: ドキュメントのリビジョン履歴を表示する」を参照してください。

ステップ 6: ドキュメントのリビジョン履歴を表示する

前のステップで車両の登録データを変更した後、登録されているすべての所有者とその他の更新されたフィールドの履歴に対してクエリを実行できます。このステップでは、vehicle-registration 台帳の VehicleRegistration テーブルに含まれているドキュメントのリビジョン履歴のクエリを実行します。

リビジョン履歴を表示するには

1. 次のプログラムを確認します (QueryHistory.java)。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.time.Instant;
import java.time.temporal.ChronoUnit;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

/**
 * Query a table's history for a particular set of documents.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class QueryHistory {
    public static final Logger log =
 LoggerFactory.getLogger(QueryHistory.class);
    private static final int THREE_MONTHS = 90;

    private QueryHistory() { }

    /**
     * In this example, query the 'VehicleRegistration' history table to find
 all previous primary owners for a VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              VIN to find previous primary owners for.
     * @param query
```

```
     *              The query to find previous primary owners.
     * @throws IllegalStateException if failed to convert document ID to an
 {@link IonValue}.
     */
    public static void previousPrimaryOwners(final TransactionExecutor txn,
 final String vin, final String query) {
        try {
            final String docId = VehicleRegistration.getDocumentIdByVin(txn,
 vin);

            log.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}...", vin);
            final Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(docId));
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String threeMonthsAgo = Instant.now().minus(THREE_MONTHS,
 ChronoUnit.DAYS).toString();
        final String query = String.format("SELECT data.Owners.PrimaryOwner,
 metadata.version "
                                           + "FROM history(VehicleRegistration,
 `%s`) "
                                           + "AS h WHERE h.metadata.id = ?",
 threeMonthsAgo);
        ConnectToLedger.getDriver().execute(txn -> {
            final String vin = SampleData.VEHICLES.get(0).getVin();
            previousPrimaryOwners(txn, vin, query);
        });
        log.info("Successfully queried history.");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonValue;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

import java.io.IOException;
import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.Collections;
import java.util.List;

/**
 * Query a table's history for a particular set of documents.
 *
 * This code expects that you have AWS credentials setup per:
```

```
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class QueryHistory {
    public static final Logger log =
 LoggerFactory.getLogger(QueryHistory.class);
    private static final int THREE_MONTHS = 90;

    private QueryHistory() { }

    /**
     * In this example, query the 'VehicleRegistration' history table to find
 all previous primary owners for a VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              VIN to find previous primary owners for.
     * @param query
     *              The query to find previous primary owners.
     * @throws IllegalStateException if failed to convert document ID to an
 {@link IonValue}.
     */
    public static void previousPrimaryOwners(final TransactionExecutor txn,
 final String vin, final String query) {
        try {
            final String docId = VehicleRegistration.getDocumentIdByVin(txn,
 vin);
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(docId));
            log.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}...", vin);
            final Result result = txn.execute(query, parameters);
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String threeMonthsAgo = Instant.now().minus(THREE_MONTHS,
 ChronoUnit.DAYS).toString();
        final String query = String.format("SELECT data.Owners.PrimaryOwner,
 metadata.version "
```

```
                                                + "FROM history(VehicleRegistration,
 `%s`) "
                                                + "AS h WHERE h.metadata.id = ?",
 threeMonthsAgo);
        ConnectToLedger.getDriver().execute(txn -> {
            final String vin = SampleData.VEHICLES.get(0).getVin();
            previousPrimaryOwners(txn, vin, query);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Successfully queried history.");
    }
}
```

> **Note**
>
> - 以下の構文で組み込みの「履歴関数」のクエリを実行することで、ドキュメントのリビジョン履歴を表示できます。
>
>   ```
>   SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
>   [ WHERE h.metadata.id = 'id' ]
>   ```
>
> - start-time および end-time はいずれもオプションです。これらは、バックティック (`` `...` ``) で示すことができる Amazon Ion リテラル値です。詳細については、「Amazon QLDB での PartiQL による Ion のクエリ」を参照してください。
> - ベストプラクティスとして、履歴クエリは日付範囲 (start-time および end-time) とドキュメント ID (`metadata.id`) の両方で修飾します。QLDB はトランザクション内の SELECT クエリを処理します。これらは、トランザクションタイムアウト制限が適用されます。
>
>   QLDB 履歴はドキュメント ID によってインデックス付けされるため、現時点では追加の履歴インデックスを作成することはできません。開始時刻と終了時刻を含む履歴クエリでは、日付範囲修飾のメリットが得られます。

2. `QueryHistory.java` プログラムをコンパイルおよび実行して、VIN が `1N4AL11D75C109151` の `VehicleRegistration` ドキュメントのリビジョン履歴をクエリします。

`vehicle-registration` 台帳のドキュメントリビジョンを暗号的に検証するには、「ステップ 7: 台帳内のドキュメントを検証する」に進みます。

ステップ 7: 台帳内のドキュメントを検証する

Amazon QLDB では、SHA-256 の暗号的ハッシュを使用して、台帳のジャーナルのドキュメントの整合性を効率的に検証できます。検証と暗号的ハッシュが QLDB でどのように機能するかについては、「Amazon QLDB でのデータ検証」を参照してください。

このステップでは、vehicle-registration 台帳の VehicleRegistration テーブルのドキュメントリビジョンを確認します。まず、ダイジェストをリクエストします。ダイジェストは出力ファイルとして返され、台帳の変更履歴全体の署名として機能します。次に、そのダイジェストに関連するリビジョンの証明をリクエストします。この証明を使用して、すべての検証チェックに合格すると、リビジョンの整合性が検証されます。

ドキュメントのリビジョンを検証するには

1. 以下の .java ファイルを確認します。これらのファイルは、検証に必要な QLDB オブジェクトと、Ion 値および文字列値のヘルパーメソッドを含むユーティリティクラスを表します。

   1. BlockAddress.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
 */

package software.amazon.qldb.tutorial.qldb;

import java.util.Objects;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Represents the BlockAddress field of a QLDB document.
 */
public final class BlockAddress {

    private static final Logger log =
 LoggerFactory.getLogger(BlockAddress.class);

    private final String strandId;
    private final long sequenceNo;

    @JsonCreator
    public BlockAddress(@JsonProperty("strandId") final String strandId,
                        @JsonProperty("sequenceNo") final long sequenceNo) {
        this.strandId = strandId;
        this.sequenceNo = sequenceNo;
    }

    public long getSequenceNo() {
        return sequenceNo;
    }

    public String getStrandId() {
        return strandId;
    }

    @Override
    public String toString() {
        return "BlockAddress{"
                + "strandId='" + strandId + '\''
                + ", sequenceNo=" + sequenceNo
                + '}';
```

```
    }

    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        BlockAddress that = (BlockAddress) o;
        return sequenceNo == that.sequenceNo
                && strandId.equals(that.strandId);
    }

    @Override
    public int hashCode() {
        // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
 properties.
        return Objects.hash(strandId, sequenceNo);
        // CHECKSTYLE:ON
    }
}
```

2. Proof.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;

import java.util.ArrayList;
import java.util.List;

/**
 * A Java representation of the {@link Proof} object.
 * Returned from the {@link
 com.amazonaws.services.qldb.AmazonQLDB#getRevision(GetRevisionRequest)} api.
 */
public final class Proof {
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();

    private List<byte[]> internalHashes;

    public Proof(final List<byte[]> internalHashes) {
        this.internalHashes = internalHashes;
    }

    public List<byte[]> getInternalHashes() {
        return internalHashes;
    }

    /**
     * Decodes a {@link Proof} from an ion text String. This ion text is returned
 in
     * a {@link GetRevisionResult#getProof()}
     *
     * @param ionText
     *              The ion text representing a {@link Proof} object.
```

```
     * @return {@link JournalBlock} parsed from the ion text.
     * @throws IllegalStateException if failed to parse the {@link Proof} object
 from the given ion text.
     */
    public static Proof fromBlob(final String ionText) {
        try {
            IonReader reader = SYSTEM.newReader(ionText);
            List<byte[]> list = new ArrayList<>();
            reader.next();
            reader.stepIn();
            while (reader.next() != null) {
                list.add(reader.newBytes());
            }
            return new Proof(list);
        } catch (Exception e) {
            throw new IllegalStateException("Failed to parse a Proof from byte
 array");
        }
    }
}
```

3. QldbIonUtils.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonValue;
import com.amazon.ionhash.IonHashReader;
import com.amazon.ionhash.IonHashReaderBuilder;
import com.amazon.ionhash.MessageDigestIonHasherProvider;
import software.amazon.qldb.tutorial.Constants;

public class QldbIonUtils {

    private static MessageDigestIonHasherProvider ionHasherProvider = new
 MessageDigestIonHasherProvider("SHA-256");

    private QldbIonUtils() {}

    /**
     * Builds a hash value from the given {@link IonValue}.
     *
     * @param ionValue
     *              The {@link IonValue} to hash.
     * @return a byte array representing the hash value.
     */
    public static byte[] hashIonValue(final IonValue ionValue) {
        IonReader reader = Constants.SYSTEM.newReader(ionValue);
        IonHashReader hashReader = IonHashReaderBuilder.standard()
                .withHasherProvider(ionHasherProvider)
                .withReader(reader)
                .build();
        while (hashReader.next() != null) {  }
        return hashReader.digest();
    }

}
```

4. QldbStringUtils.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
```

```
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonTextWriterBuilder;
import com.amazonaws.services.qldb.model.GetBlockResult;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.ValueHolder;

import java.io.IOException;

/**
 * Helper methods to pretty-print certain QLDB response types.
 */
public class QldbStringUtils {

    private QldbStringUtils() {}

    /**
     * Returns the string representation of a given {@link ValueHolder}.
     * Adapted from the AWS SDK autogenerated {@code toString()} method, with
 sensitive values un-redacted.
```

```
    * Additionally, this method pretty-prints any IonText included in the {@link
ValueHolder}.
    *
    * @param valueHolder the {@link ValueHolder} to convert to a String.
    * @return the String representation of the supplied {@link ValueHolder}.
    */
   public static String toUnredactedString(ValueHolder valueHolder) {
       StringBuilder sb = new StringBuilder();
       sb.append("{");
       if (valueHolder.getIonText() != null) {

           sb.append("IonText: ");
           IonWriter prettyWriter = IonTextWriterBuilder.pretty().build(sb);
           try {

prettyWriter.writeValues(IonReaderBuilder.standard().build(valueHolder.getIonText()));
           } catch (IOException ioe) {
               sb.append("**Exception while printing this IonText**");
           }
       }

       sb.append("}");
       return sb.toString();
   }

   /**
    * Returns the string representation of a given {@link GetBlockResult}.
    * Adapted from the AWS SDK autogenerated {@code toString()} method, with
sensitive values un-redacted.
    *
    * @param getBlockResult the {@link GetBlockResult} to convert to a String.
    * @return the String representation of the supplied {@link GetBlockResult}.
    */
   public static String toUnredactedString(GetBlockResult getBlockResult) {
       StringBuilder sb = new StringBuilder();
       sb.append("{");
       if (getBlockResult.getBlock() != null) {
           sb.append("Block:
").append(toUnredactedString(getBlockResult.getBlock())).append(",");
       }

       if (getBlockResult.getProof() != null) {
           sb.append("Proof:
").append(toUnredactedString(getBlockResult.getProof()));
```

```
        }

        sb.append("}");
        return sb.toString();
    }

    /**
     * Returns the string representation of a given {@link GetDigestResult}.
     * Adapted from the AWS SDK autogenerated {@code toString()} method, with
 sensitive values un-redacted.
     *
     * @param getDigestResult the {@link GetDigestResult} to convert to a String.
     * @return the String representation of the supplied {@link GetDigestResult}.
     */
    public static String toUnredactedString(GetDigestResult getDigestResult) {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        if (getDigestResult.getDigest() != null) {
            sb.append("Digest:
 ").append(getDigestResult.getDigest()).append(",");
        }

        if (getDigestResult.getDigestTipAddress() != null) {
            sb.append("DigestTipAddress:
 ").append(toUnredactedString(getDigestResult.getDigestTipAddress()));
        }

        sb.append("}");
        return sb.toString();
    }
}
```

5. `Verifier.java`

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 copy of this
 * software and associated documentation files (the "Software"), to deal in
 the Software
```

```
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
 A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.util.Base64;

import software.amazon.qldb.tutorial.qldb.Proof;

/**
 * Encapsulates the logic to verify the integrity of revisions or blocks in a
 QLDB ledger.
 *
 * The main entry point is {@link #verify(byte[], byte[], String)}.
```

```
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class Verifier {
    public static final Logger log = LoggerFactory.getLogger(Verifier.class);
    private static final int HASH_LENGTH = 32;
    private static final int UPPER_BOUND = 8;

    /**
     * Compares two hashes by their <em>signed</em> byte values in little-
endian order.
     */
    private static Comparator<byte[]> hashComparator = (h1, h2) -> {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }
        for (int i = h1.length - 1; i >= 0; i--) {
            int byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                return byteEqual;
            }
        }

        return 0;
    };

    private Verifier() { }

    /**
     * Verify the integrity of a document with respect to a QLDB ledger
 digest.
     *
     * The verification algorithm includes the following steps:
     *
     * 1. {@link #buildCandidateDigest(Proof, byte[])} build the candidate
 digest from the internal hashes
     * in the {@link Proof}.
     * 2. Check that the {@code candidateLedgerDigest} is equal to the {@code
 ledgerDigest}.
     *
     * @param documentHash
     *              The hash of the document to be verified.
```

```
    * @param digest
    *              The QLDB ledger digest. This digest should have been
retrieved using
    *              {@link com.amazonaws.services.qldb.AmazonQLDB#getDigest}
    * @param proofBlob
    *              The ion encoded bytes representing the {@link Proof}
associated with the supplied
    *              {@code digestTipAddress} and {@code address} retrieved
using
    *              {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
    * @return {@code true} if the record is verified or {@code false} if it
is not verified.
    */
   public static boolean verify(
           final byte[] documentHash,
           final byte[] digest,
           final String proofBlob
   ) {
       Proof proof = Proof.fromBlob(proofBlob);

       byte[] candidateDigest = buildCandidateDigest(proof, documentHash);

       return Arrays.equals(digest, candidateDigest);
   }

   /**
    * Build the candidate digest representing the entire ledger from the
internal hashes of the {@link Proof}.
    *
    * @param proof
    *              A Java representation of {@link Proof}
    *              returned from {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
    * @param leafHash
    *              Leaf hash to build the candidate digest with.
    * @return a byte array of the candidate digest.
    */
   private static byte[] buildCandidateDigest(final Proof proof, final byte[]
leafHash) {
       return calculateRootHashFromInternalHashes(proof.getInternalHashes(),
leafHash);
   }
```

```
    /**
     * Get a new instance of {@link MessageDigest} using the SHA-256
algorithm.
     *
     * @return an instance of {@link MessageDigest}.
     * @throws IllegalStateException if the algorithm is not available on the
current JVM.
     */
    static MessageDigest newMessageDigest() {
        try {
            return MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            log.error("Failed to create SHA-256 MessageDigest", e);
            throw new IllegalStateException("SHA-256 message digest is
unavailable", e);
        }
    }

    /**
     * Takes two hashes, sorts them, concatenates them, and then returns the
     * hash of the concatenated array.
     *
     * @param h1
     *              Byte array containing one of the hashes to compare.
     * @param h2
     *              Byte array containing one of the hashes to compare.
     * @return the concatenated array of hashes.
     */
    public static byte[] dot(final byte[] h1, final byte[] h2) {
        if (h1.length == 0) {
            return h2;
        }
        if (h2.length == 0) {
            return h1;
        }
        byte[] concatenated = new byte[h1.length + h2.length];
        if (hashComparator.compare(h1, h2) < 0) {
            System.arraycopy(h1, 0, concatenated, 0, h1.length);
            System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
        } else {
            System.arraycopy(h2, 0, concatenated, 0, h2.length);
            System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
        }
        MessageDigest messageDigest = newMessageDigest();
```

```
        messageDigest.update(concatenated);

        return messageDigest.digest();
    }

    /**
     * Starting with the provided {@code leafHash} combined with the provided
{@code internalHashes}
     * pairwise until only the root hash remains.
     *
     * @param internalHashes
     *              Internal hashes of Merkle tree.
     * @param leafHash
     *              Leaf hashes of Merkle tree.
     * @return the root hash.
     */
    private static byte[] calculateRootHashFromInternalHashes(final
List<byte[]> internalHashes, final byte[] leafHash) {
        return internalHashes.stream().reduce(leafHash, Verifier::dot);
    }

    /**
     * Flip a single random bit in the given byte array. This method is used
to demonstrate
     * QLDB's verification features.
     *
     * @param original
     *              The original byte array.
     * @return the altered byte array with a single random bit changed.
     */
    public static byte[] flipRandomBit(final byte[] original) {
        if (original.length == 0) {
            throw new IllegalArgumentException("Array cannot be empty!");
        }
        int alteredPosition =
ThreadLocalRandom.current().nextInt(original.length);
        int b = ThreadLocalRandom.current().nextInt(UPPER_BOUND);
        byte[] altered = new byte[original.length];
        System.arraycopy(original, 0, altered, 0, original.length);
        altered[alteredPosition] = (byte) (altered[alteredPosition] ^ (1 <<
b));
        return altered;
    }
```

```
    public static String toBase64(byte[] arr) {
        return new String(Base64.encode(arr), StandardCharsets.UTF_8);
    }

    /**
     * Convert a {@link ByteBuffer} into byte array.
     *
     * @param buffer
     *              The {@link ByteBuffer} to convert.
     * @return the converted byte array.
     */
    public static byte[] convertByteBufferToByteArray(final ByteBuffer buffer)
{
        byte[] arr = new byte[buffer.remaining()];
        buffer.get(arr);
        return arr;
    }

    /**
     * Calculates the root hash from a list of hashes that represent the base
of a Merkle tree.
     *
     * @param hashes
     *              The list of byte arrays representing hashes making up base
of a Merkle tree.
     * @return a byte array that is the root hash of the given list of hashes.
     */
    public static byte[] calculateMerkleTreeRootHash(List<byte[]> hashes) {
        if (hashes.isEmpty()) {
            return new byte[0];
        }

        List<byte[]> remaining = combineLeafHashes(hashes);
        while (remaining.size() > 1) {
            remaining = combineLeafHashes(remaining);
        }
        return remaining.get(0);
    }

    private static List<byte[]> combineLeafHashes(List<byte[]> hashes) {
        List<byte[]> combinedHashes = new ArrayList<>();
        Iterator<byte[]> it = hashes.stream().iterator();

        while (it.hasNext()) {
```

```
            byte[] left = it.next();
            if (it.hasNext()) {
                byte[] right = it.next();
                byte[] combined = dot(left, right);
                combinedHashes.add(combined);
            } else {
                combinedHashes.add(left);
            }
        }

        return combinedHashes;
    }
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 copy of this
 * software and associated documentation files (the "Software"), to deal in
 the Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
 A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial;

import com.amazonaws.util.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.qldb.Proof;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
import java.util.concurrent.ThreadLocalRandom;

/**
 * Encapsulates the logic to verify the integrity of revisions or blocks in a
 QLDB ledger.
 *
 * The main entry point is {@link #verify(byte[], byte[], String)}.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class Verifier {
    public static final Logger log = LoggerFactory.getLogger(Verifier.class);
    private static final int HASH_LENGTH = 32;
    private static final int UPPER_BOUND = 8;

    /**
     * Compares two hashes by their <em>signed</em> byte values in little-
endian order.
     */
    private static Comparator<byte[]> hashComparator = (h1, h2) -> {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }
        for (int i = h1.length - 1; i >= 0; i--) {
            int byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                return byteEqual;
            }
        }
```

```
        return 0;
    };

    private Verifier() { }

    /**
     * Verify the integrity of a document with respect to a QLDB ledger
digest.
     *
     * The verification algorithm includes the following steps:
     *
     * 1. {@link #buildCandidateDigest(Proof, byte[])} build the candidate
digest from the internal hashes
     * in the {@link Proof}.
     * 2. Check that the {@code candidateLedgerDigest} is equal to the {@code
ledgerDigest}.
     *
     * @param documentHash
     *              The hash of the document to be verified.
     * @param digest
     *              The QLDB ledger digest. This digest should have been
retrieved using
     *              {@link com.amazonaws.services.qldb.AmazonQLDB#getDigest}
     * @param proofBlob
     *              The ion encoded bytes representing the {@link Proof}
associated with the supplied
     *              {@code digestTipAddress} and {@code address} retrieved
using
     *              {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
     * @return {@code true} if the record is verified or {@code false} if it
is not verified.
     */
    public static boolean verify(
            final byte[] documentHash,
            final byte[] digest,
            final String proofBlob
    ) {
        Proof proof = Proof.fromBlob(proofBlob);

        byte[] candidateDigest = buildCandidateDigest(proof, documentHash);

        return Arrays.equals(digest, candidateDigest);
    }
```

```
   /**
    * Build the candidate digest representing the entire ledger from the
internal hashes of the {@link Proof}.
    *
    * @param proof
    *              A Java representation of {@link Proof}
    *              returned from {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
    * @param leafHash
    *              Leaf hash to build the candidate digest with.
    * @return a byte array of the candidate digest.
    */
   private static byte[] buildCandidateDigest(final Proof proof, final byte[]
leafHash) {
       return calculateRootHashFromInternalHashes(proof.getInternalHashes(),
leafHash);
   }

   /**
    * Get a new instance of {@link MessageDigest} using the SHA-256
algorithm.
    *
    * @return an instance of {@link MessageDigest}.
    * @throws IllegalStateException if the algorithm is not available on the
current JVM.
    */
   static MessageDigest newMessageDigest() {
       try {
           return MessageDigest.getInstance("SHA-256");
       } catch (NoSuchAlgorithmException e) {
           log.error("Failed to create SHA-256 MessageDigest", e);
           throw new IllegalStateException("SHA-256 message digest is
unavailable", e);
       }
   }

   /**
    * Takes two hashes, sorts them, concatenates them, and then returns the
    * hash of the concatenated array.
    *
    * @param h1
    *              Byte array containing one of the hashes to compare.
    * @param h2
```

```
     *              Byte array containing one of the hashes to compare.
     * @return the concatenated array of hashes.
     */
    public static byte[] dot(final byte[] h1, final byte[] h2) {
        if (h1.length == 0) {
            return h2;
        }
        if (h2.length == 0) {
            return h1;
        }
        byte[] concatenated = new byte[h1.length + h2.length];
        if (hashComparator.compare(h1, h2) < 0) {
            System.arraycopy(h1, 0, concatenated, 0, h1.length);
            System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
        } else {
            System.arraycopy(h2, 0, concatenated, 0, h2.length);
            System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
        }
        MessageDigest messageDigest = newMessageDigest();
        messageDigest.update(concatenated);

        return messageDigest.digest();
    }

    /**
     * Starting with the provided {@code leafHash} combined with the provided
{@code internalHashes}
     * pairwise until only the root hash remains.
     *
     * @param internalHashes
     *              Internal hashes of Merkle tree.
     * @param leafHash
     *              Leaf hashes of Merkle tree.
     * @return the root hash.
     */
    private static byte[] calculateRootHashFromInternalHashes(final
List<byte[]> internalHashes, final byte[] leafHash) {
        return internalHashes.stream().reduce(leafHash, Verifier::dot);
    }

    /**
     * Flip a single random bit in the given byte array. This method is used
to demonstrate
     * QLDB's verification features.
```

```
     *
     * @param original
     *              The original byte array.
     * @return the altered byte array with a single random bit changed.
     */
    public static byte[] flipRandomBit(final byte[] original) {
        if (original.length == 0) {
            throw new IllegalArgumentException("Array cannot be empty!");
        }
        int alteredPosition =
ThreadLocalRandom.current().nextInt(original.length);
        int b = ThreadLocalRandom.current().nextInt(UPPER_BOUND);
        byte[] altered = new byte[original.length];
        System.arraycopy(original, 0, altered, 0, original.length);
        altered[alteredPosition] = (byte) (altered[alteredPosition] ^ (1 <<
b));
        return altered;
    }

    public static String toBase64(byte[] arr) {
        return new String(Base64.encode(arr), StandardCharsets.UTF_8);
    }

    /**
     * Convert a {@link ByteBuffer} into byte array.
     *
     * @param buffer
     *              The {@link ByteBuffer} to convert.
     * @return the converted byte array.
     */
    public static byte[] convertByteBufferToByteArray(final ByteBuffer buffer)
{
        byte[] arr = new byte[buffer.remaining()];
        buffer.get(arr);
        return arr;
    }

    /**
     * Calculates the root hash from a list of hashes that represent the base
of a Merkle tree.
     *
     * @param hashes
     *              The list of byte arrays representing hashes making up base
of a Merkle tree.
```

```
     * @return a byte array that is the root hash of the given list of hashes.
     */
    public static byte[] calculateMerkleTreeRootHash(List<byte[]> hashes) {
        if (hashes.isEmpty()) {
            return new byte[0];
        }

        List<byte[]> remaining = combineLeafHashes(hashes);
        while (remaining.size() > 1) {
            remaining = combineLeafHashes(remaining);
        }
        return remaining.get(0);
    }

    private static List<byte[]> combineLeafHashes(List<byte[]> hashes) {
        List<byte[]> combinedHashes = new ArrayList<>();
        Iterator<byte[]> it = hashes.stream().iterator();

        while (it.hasNext()) {
            byte[] left = it.next();
            if (it.hasNext()) {
                byte[] right = it.next();
                byte[] combined = dot(left, right);
                combinedHashes.add(combined);
            } else {
                combinedHashes.add(left);
            }
        }

        return combinedHashes;
    }
}
```

2. 2 つの .java ファイル (GetDigest.java および GetRevision.java) を使用して、次の手順を実行します。

   - vehicle-registration 台帳に新しいダイジェストをリクエストします。
   - VehicleRegistration テーブルにドキュメントの各リビジョンの証明をリクエストします。
   - 返されたダイジェストと証明を使用して、ダイジェストを再計算することで、リビジョンを検証します。

GetDigest.java プログラムには、次のコードが含まれています。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestRequest;
import com.amazonaws.services.qldb.model.GetDigestResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

/**
 * This is an example for retrieving the digest of a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
```

```
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class GetDigest {
    public static final Logger log = LoggerFactory.getLogger(GetDigest.class);
    public static AmazonQLDB client = CreateLedger.getClient();

    private GetDigest() { }

    /**
     * Calls {@link #getDigest(String)} for a ledger.
     *
     * @param args
     *              Arbitrary command-line arguments.
     * @throws Exception if failed to get a ledger digest.
     */
    public static void main(final String... args) throws Exception {
        try {

            getDigest(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to get a ledger digest!", e);
            throw e;
        }
    }

    /**
     * Get the digest for the specified ledger.
     *
     * @param ledgerName
     *              The ledger to get digest from.
     * @return {@link GetDigestResult}.
     */
    public static GetDigestResult getDigest(final String ledgerName) {
        log.info("Let's get the current digest of the ledger named {}.",
 ledgerName);
        GetDigestRequest request = new GetDigestRequest()
                .withName(ledgerName);
        GetDigestResult result = client.getDigest(request);
        log.info("Success. LedgerDigest: {}.",
 QldbStringUtils.toUnredactedString(result));
        return result;
    }
```

```
}
```

> **Note**
>
> getDigest メソッドを使用して、台帳のジャーナルの現在のティップを含むダイジェストをリクエストします。ジャーナルのティップとは、QLDB がリクエストを受けた時点でコミット済みの最新のブロックのことです。

GetRevision.java プログラムには、次のコードが含まれています。

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
```

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.qldb.BlockAddress;
import software.amazon.qldb.tutorial.qldb.QldbRevision;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class GetRevision {
    public static final Logger log = LoggerFactory.getLogger(GetRevision.class);
    public static AmazonQLDB client = CreateLedger.getClient();
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();

    private GetRevision() { }

    public static void main(String... args) throws Exception {

        final String vin = SampleData.REGISTRATIONS.get(0).getVin();
```

```
        verifyRegistration(ConnectToLedger.getDriver(), Constants.LEDGER_NAME,
 vin);
    }

    /**
     * Verify each version of the registration for the given VIN.
     *
     * @param driver
     *              A QLDB driver.
     * @param ledgerName
     *              The ledger to get digest from.
     * @param vin
     *              VIN to query the revision history of a specific registration
 with.
     * @throws Exception if failed to verify digests.
     * @throws AssertionError if document revision verification failed.
     */
    public static void verifyRegistration(final QldbDriver driver, final String
 ledgerName, final String vin)
            throws Exception {
        log.info(String.format("Let's verify the registration with VIN=%s, in
 ledger=%s.", vin, ledgerName));

        try {
            log.info("First, let's get a digest.");
            GetDigestResult digestResult = GetDigest.getDigest(ledgerName);

            ValueHolder digestTipAddress = digestResult.getDigestTipAddress();
            byte[] digestBytes =
 Verifier.convertByteBufferToByteArray(digestResult.getDigest());

            log.info("Got a ledger digest. Digest end address={}, digest={}.",
                QldbStringUtils.toUnredactedString(digestTipAddress),
                Verifier.toBase64(digestBytes));

            log.info(String.format("Next, let's query the registration with VIN=
%s. "
                    + "Then we can verify each version of the registration.",
 vin));
            List<IonStruct> documentsWithMetadataList = new ArrayList<>();
            driver.execute(txn -> {
                documentsWithMetadataList.addAll(queryRegistrationsByVin(txn,
 vin));
```

```
            });
            log.info("Registrations queried successfully!");

            log.info(String.format("Found %s revisions of the registration with
VIN=%s.",
                    documentsWithMetadataList.size(), vin));

            for (IonStruct ionStruct : documentsWithMetadataList) {

                QldbRevision document = QldbRevision.fromIon(ionStruct);
                log.info(String.format("Let's verify the document: %s",
document));

                log.info("Let's get a proof for the document.");
                GetRevisionResult proofResult = getRevision(
                        ledgerName,
                        document.getMetadata().getId(),
                        digestTipAddress,
                        document.getBlockAddress()
                );

                final IonValue proof =
Constants.MAPPER.writeValueAsIonValue(proofResult.getProof());
                final IonReader reader =
IonReaderBuilder.standard().build(proof);
                reader.next();
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                IonWriter writer = SYSTEM.newBinaryWriter(baos);
                writer.writeValue(reader);
                writer.close();
                baos.flush();
                baos.close();
                byte[] byteProof = baos.toByteArray();

                log.info(String.format("Got back a proof: %s",
Verifier.toBase64(byteProof)));

                boolean verified = Verifier.verify(
                        document.getHash(),
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (!verified) {
```

```
                    throw new AssertionError("Document revision is not
verified!");
                } else {
                    log.info("Success! The document is verified");
                }

                byte[] alteredDigest = Verifier.flipRandomBit(digestBytes);
                log.info(String.format("Flipping one bit in the digest and
assert that the document is NOT verified. "
                        + "The altered digest is: %s",
Verifier.toBase64(alteredDigest)));
                verified = Verifier.verify(
                        document.getHash(),
                        alteredDigest,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected document to not be
verified against altered digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the digest
causes verification to fail.");
                }

                byte[] alteredDocumentHash =
Verifier.flipRandomBit(document.getHash());
                log.info(String.format("Flipping one bit in the document's hash
and assert that it is NOT verified. "
                        + "The altered document hash is: %s.",
Verifier.toBase64(alteredDocumentHash)));
                verified = Verifier.verify(
                        alteredDocumentHash,
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected altered document hash to
not be verified against digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the
document hash causes verification to fail.");
                }
```

```
            }

        } catch (Exception e) {
            log.error("Failed to verify digests.", e);
            throw e;
        }

        log.info(String.format("Finished verifying the registration with VIN=%s
in ledger=%s.", vin, ledgerName));
    }

    /**
     * Get the revision of a particular document specified by the given document
ID and block address.
     *
     * @param ledgerName
     *              Name of the ledger containing the document.
     * @param documentId
     *              Unique ID for the document to be verified, contained in the
committed view of the document.
     * @param digestTipAddress
     *              The latest block location covered by the digest.
     * @param blockAddress
     *              The location of the block to request.
     * @return the requested revision.
     */
    public static GetRevisionResult getRevision(final String ledgerName, final
String documentId,
                                                final ValueHolder
digestTipAddress, final BlockAddress blockAddress) {
        try {
            GetRevisionRequest request = new GetRevisionRequest()
                    .withName(ledgerName)
                    .withDigestTipAddress(digestTipAddress)
                    .withBlockAddress(new
ValueHolder().withIonText(Constants.MAPPER.writeValueAsIonValue(blockAddress)
                            .toString()))
                    .withDocumentId(documentId);
            return client.getRevision(request);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
```

```
    /**
     * Query the registration history for the given VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              The unique VIN to query.
     * @return a list of {@link IonStruct} representing the registration
 history.
     * @throws IllegalStateException if failed to convert parameters into {@link
 IonValue}
     */
    public static List<IonStruct> queryRegistrationsByVin(final
 TransactionExecutor txn, final String vin) {
        log.info(String.format("Let's query the 'VehicleRegistration' table for
 VIN: %s...", vin));
        log.info("Let's query the 'VehicleRegistration' table for VIN: {}...",
 vin);
        final String query = String.format("SELECT * FROM _ql_committed_%s WHERE
 data.VIN = ?",
                Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        try {
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            List<IonStruct> list = ScanTable.toIonStructs(result);
            log.info(String.format("Found %d document(s)!", list.size()));
            return list;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.qldb.BlockAddress;
import software.amazon.qldb.tutorial.qldb.QldbRevision;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class GetRevision {
    public static final Logger log = LoggerFactory.getLogger(GetRevision.class);
    public static AmazonQLDB client = CreateLedger.getClient();

    private GetRevision() { }

    public static void main(String... args) throws Exception {

        final String vin = SampleData.REGISTRATIONS.get(0).getVin();

        try (QldbSession qldbSession = ConnectToLedger.createQldbSession()) {
            verifyRegistration(qldbSession, Constants.LEDGER_NAME, vin);
        }
    }

    /**
     * Verify each version of the registration for the given VIN.
     *
     * @param qldbSession
     *              A QLDB session.
     * @param ledgerName
     *              The ledger to get digest from.
     * @param vin
     *              VIN to query the revision history of a specific registration
 with.
     * @throws Exception if failed to verify digests.
     * @throws AssertionError if document revision verification failed.
     */
    public static void verifyRegistration(final QldbSession qldbSession, final
 String ledgerName, final String vin)
            throws Exception {
        log.info(String.format("Let's verify the registration with VIN=%s, in
 ledger=%s.", vin, ledgerName));
```

```
        try {
            log.info("First, let's get a digest.");
            GetDigestResult digestResult = GetDigest.getDigest(ledgerName);

            ValueHolder digestTipAddress = digestResult.getDigestTipAddress();
            byte[] digestBytes =
 Verifier.convertByteBufferToByteArray(digestResult.getDigest());

            log.info("Got a ledger digest. Digest end address={}, digest={}.",
                QldbStringUtils.toUnredactedString(digestTipAddress),
                Verifier.toBase64(digestBytes));

            log.info(String.format("Next, let's query the registration with VIN=
%s. "
                    + "Then we can verify each version of the registration.",
 vin));
            List<IonStruct> documentsWithMetadataList = new ArrayList<>();
            qldbSession.execute(txn -> {
                documentsWithMetadataList.addAll(queryRegistrationsByVin(txn,
 vin));
            }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
            log.info("Registrations queried successfully!");

            log.info(String.format("Found %s revisions of the registration with
 VIN=%s.",
                    documentsWithMetadataList.size(), vin));

            for (IonStruct ionStruct : documentsWithMetadataList) {

                QldbRevision document = QldbRevision.fromIon(ionStruct);
                log.info(String.format("Let's verify the document: %s",
 document));

                log.info("Let's get a proof for the document.");
                GetRevisionResult proofResult = getRevision(
                        ledgerName,
                        document.getMetadata().getId(),
                        digestTipAddress,
                        document.getBlockAddress()
                );

                final IonValue proof =
 Constants.MAPPER.writeValueAsIonValue(proofResult.getProof());
```

```
                final IonReader reader =
IonReaderBuilder.standard().build(proof);
                reader.next();
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                IonWriter writer = Constants.SYSTEM.newBinaryWriter(baos);
                writer.writeValue(reader);
                writer.close();
                baos.flush();
                baos.close();
                byte[] byteProof = baos.toByteArray();

                log.info(String.format("Got back a proof: %s",
Verifier.toBase64(byteProof)));

                boolean verified = Verifier.verify(
                        document.getHash(),
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (!verified) {
                    throw new AssertionError("Document revision is not
verified!");
                } else {
                    log.info("Success! The document is verified");
                }

                byte[] alteredDigest = Verifier.flipRandomBit(digestBytes);
                log.info(String.format("Flipping one bit in the digest and
assert that the document is NOT verified. "
                        + "The altered digest is: %s",
Verifier.toBase64(alteredDigest)));
                verified = Verifier.verify(
                        document.getHash(),
                        alteredDigest,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected document to not be
verified against altered digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the digest
causes verification to fail.");
```

```
                }

                byte[] alteredDocumentHash =
Verifier.flipRandomBit(document.getHash());
                log.info(String.format("Flipping one bit in the document's hash
and assert that it is NOT verified. "
                        + "The altered document hash is: %s.",
Verifier.toBase64(alteredDocumentHash)));
                verified = Verifier.verify(
                        alteredDocumentHash,
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected altered document hash to
not be verified against digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the
document hash causes verification to fail.");
                }
            }

        } catch (Exception e) {
            log.error("Failed to verify digests.", e);
            throw e;
        }

        log.info(String.format("Finished verifying the registration with VIN=%s
in ledger=%s.", vin, ledgerName));
    }

    /**
     * Get the revision of a particular document specified by the given document
ID and block address.
     *
     * @param ledgerName
     *              Name of the ledger containing the document.
     * @param documentId
     *              Unique ID for the document to be verified, contained in the
committed view of the document.
     * @param digestTipAddress
     *              The latest block location covered by the digest.
     * @param blockAddress
```

```
     *              The location of the block to request.
     * @return the requested revision.
     */
    public static GetRevisionResult getRevision(final String ledgerName, final
String documentId,
                                                final ValueHolder
digestTipAddress, final BlockAddress blockAddress) {
        try {
            GetRevisionRequest request = new GetRevisionRequest()
                    .withName(ledgerName)
                    .withDigestTipAddress(digestTipAddress)
                    .withBlockAddress(new
ValueHolder().withIonText(Constants.MAPPER.writeValueAsIonValue(blockAddress)
                            .toString()))
                    .withDocumentId(documentId);
            return client.getRevision(request);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Query the registration history for the given VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              The unique VIN to query.
     * @return a list of {@link IonStruct} representing the registration
history.
     * @throws IllegalStateException if failed to convert parameters into {@link
IonValue}
     */
    public static List<IonStruct> queryRegistrationsByVin(final
TransactionExecutor txn, final String vin) {
        log.info(String.format("Let's query the 'VehicleRegistration' table for
VIN: %s...", vin));
        log.info("Let's query the 'VehicleRegistration' table for VIN: {}...",
vin);
        final String query = String.format("SELECT * FROM _ql_committed_%s WHERE
data.VIN = ?",
                Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        try {
```

```
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            List<IonStruct> list = ScanTable.toIonStructs(result);
            log.info(String.format("Found %d document(s)!", list.size()));
            return list;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
}
```

> **Note**
>
> getRevision メソッドから、指定したドキュメントリビジョンの証明が返された後、このプログラムはクライアント側 API を使用してそのリビジョンを検証します。この API で使用されているアルゴリズムの概要については、「証明を使用したダイジェストの再計算」を参照してください。

3. GetRevision.java プログラムをコンパイルおよび実行して、VIN が 1N4AL11D75C109151 の VehicleRegistration ドキュメントを暗号的に検証します。

vehicle-registration 台帳でジャーナルデータをエクスポートして検証するには、「ステップ 8: 元帳のジャーナルデータをエクスポートして検証する」に進みます。

ステップ 8: 元帳のジャーナルデータをエクスポートして検証する

Amazon QLDB では、台帳のジャーナルのコンテンツには、データ保持、分析、監査などのさまざまな目的でアクセスできます。詳細については、「Amazon QLDB からのジャーナルデータのエクスポート」を参照してください。

このステップでは、ジャーナルブロックを vehicle-registration 台帳から Amazon S3 バケットにエクスポートします。次に、エクスポートされたデータを使用して、ジャーナルブロックと各ブロック内の個々のハッシュコンポーネント間のハッシュチェーンを検証します。

使用する AWS Identity and Access Management (IAM) プリンシパルエンティティは、AWS アカウント で Amazon S3 バケットを作成するのに十分な IAM アクセス許可を持っている必要があります。詳細については、「Amazon S3 ユーザーガイド」の「Amazon S3 でのポリシーとアクセス許可」を参照してください。また、QLDB が Simple Storage Service (Amazon S3) バケットにオブ

ジェクトを書き込むことを許可するアクセス許可ポリシーがアタッチされた IAM ロールを作成するアクセス許可も必要です。詳細については、「IAM ユーザーガイド」の「IAM リソースにアクセスするために必要な許可」を参照してください。

ジャーナルデータをエクスポートして検証するには

1. ジャーナルブロックとそのデータ内容を表す以下のファイル (JournalBlock.java) を確認します。これには、verifyBlockHash() という名前のメソッドが含まれています。これは、ブロックハッシュの各コンポーネントを計算する方法を示しています。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.dataformat.ion.IonTimestampSerializers;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.Verifier;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

import static java.nio.ByteBuffer.wrap;

/**
 * Represents a JournalBlock that was recorded after executing a transaction
 * in the ledger.
 */
public final class JournalBlock {
    private static final Logger log = LoggerFactory.getLogger(JournalBlock.class);

    private BlockAddress blockAddress;
    private String transactionId;
    @JsonSerialize(using =
 IonTimestampSerializers.IonTimestampJavaDateSerializer.class)
    private Date blockTimestamp;
    private byte[] blockHash;
    private byte[] entriesHash;
    private byte[] previousBlockHash;
    private byte[][] entriesHashList;
    private TransactionInfo transactionInfo;
    private RedactionInfo redactionInfo;
    private List<QldbRevision> revisions;

    @JsonCreator
    public JournalBlock(@JsonProperty("blockAddress") final BlockAddress
 blockAddress,
                        @JsonProperty("transactionId") final String transactionId,
                        @JsonProperty("blockTimestamp") final Date blockTimestamp,
                        @JsonProperty("blockHash") final byte[] blockHash,
                        @JsonProperty("entriesHash") final byte[] entriesHash,
                        @JsonProperty("previousBlockHash") final byte[]
 previousBlockHash,
```

```
                        @JsonProperty("entriesHashList") final byte[][]
entriesHashList,
                        @JsonProperty("transactionInfo") final TransactionInfo
transactionInfo,
                        @JsonProperty("redactionInfo") final RedactionInfo
redactionInfo,
                        @JsonProperty("revisions") final List<QldbRevision>
revisions) {
       this.blockAddress = blockAddress;
       this.transactionId = transactionId;
       this.blockTimestamp = blockTimestamp;
       this.blockHash = blockHash;
       this.entriesHash = entriesHash;
       this.previousBlockHash = previousBlockHash;
       this.entriesHashList = entriesHashList;
       this.transactionInfo = transactionInfo;
       this.redactionInfo = redactionInfo;
       this.revisions = revisions;
   }

   public BlockAddress getBlockAddress() {
       return blockAddress;
   }

   public String getTransactionId() {
       return transactionId;
   }

   public Date getBlockTimestamp() {
       return blockTimestamp;
   }

   public byte[][] getEntriesHashList() {
       return entriesHashList;
   }

   public TransactionInfo getTransactionInfo() {
       return transactionInfo;
   }

   public RedactionInfo getRedactionInfo() {
       return redactionInfo;
   }
```

```
    public List<QldbRevision> getRevisions() {
        return revisions;
    }

    public byte[] getEntriesHash() {
        return entriesHash;
    }

    public byte[] getBlockHash() {
        return blockHash;
    }

    public byte[] getPreviousBlockHash() {
        return previousBlockHash;
    }

    @Override
    public String toString() {
        return "JournalBlock{"
                + "blockAddress=" + blockAddress
                + ", transactionId='" + transactionId + '\''
                + ", blockTimestamp=" + blockTimestamp
                + ", blockHash=" + Arrays.toString(blockHash)
                + ", entriesHash=" + Arrays.toString(entriesHash)
                + ", previousBlockHash=" + Arrays.toString(previousBlockHash)
                + ", entriesHashList=" + Arrays.toString(entriesHashList)
                + ", transactionInfo=" + transactionInfo
                + ", redactionInfo=" + redactionInfo
                + ", revisions=" + revisions
                + '}';
    }

    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof JournalBlock)) {
            return false;
        }

        final JournalBlock that = (JournalBlock) o;

        if (!getBlockAddress().equals(that.getBlockAddress())) {
```

```
            return false;
        }
        if (!getTransactionId().equals(that.getTransactionId())) {
            return false;
        }
        if (!getBlockTimestamp().equals(that.getBlockTimestamp())) {
            return false;
        }
        if (!Arrays.equals(getBlockHash(), that.getBlockHash())) {
            return false;
        }
        if (!Arrays.equals(getEntriesHash(), that.getEntriesHash())) {
            return false;
        }
        if (!Arrays.equals(getPreviousBlockHash(), that.getPreviousBlockHash())) {
            return false;
        }
        if (!Arrays.deepEquals(getEntriesHashList(), that.getEntriesHashList())) {
            return false;
        }
        if (!getTransactionInfo().equals(that.getTransactionInfo())) {
            return false;
        }
        if (getRedactionInfo() != null ? !
getRedactionInfo().equals(that.getRedactionInfo()) : that.getRedactionInfo() !=
 null) {
            return false;
        }
        return getRevisions() != null ?
 getRevisions().equals(that.getRevisions()) : that.getRevisions() == null;
    }

    @Override
    public int hashCode() {
        int result = getBlockAddress().hashCode();
        result = 31 * result + getTransactionId().hashCode();
        result = 31 * result + getBlockTimestamp().hashCode();
        result = 31 * result + Arrays.hashCode(getBlockHash());
        result = 31 * result + Arrays.hashCode(getEntriesHash());
        result = 31 * result + Arrays.hashCode(getPreviousBlockHash());
        result = 31 * result + Arrays.deepHashCode(getEntriesHashList());
        result = 31 * result + getTransactionInfo().hashCode();
        result = 31 * result + (getRedactionInfo() != null ?
 getRedactionInfo().hashCode() : 0);
```

```
        result = 31 * result + (getRevisions() != null ?
getRevisions().hashCode() : 0);
        return result;
    }

    /**
     * This method validates that the hashes of the components of a journal block
make up the block
     * hash that is provided with the block itself.
     *
     * The components that contribute to the hash of the journal block consist of
the following:
     *   - user transaction information (contained in [transactionInfo])
     *   - user redaction information (contained in [redactionInfo])
     *   - user revisions (contained in [revisions])
     *   - hashes of internal-only system metadata (contained in [revisions] and in
[entriesHashList])
     *   - the previous block hash
     *
     * If any of the computed hashes of user information cannot be validated or any
of the system
     * hashes do not result in the correct computed values, this method will throw
an IllegalArgumentException.
     *
     * Internal-only system metadata is represented by its hash, and can be present
in the form of certain
     * items in the [revisions] list that only contain a hash and no user data, as
well as some hashes
     * in [entriesHashList].
     *
     * To validate that the hashes of the user data are valid components of the
[blockHash], this method
     * performs the following steps:
     *
     * 1. Compute the hash of the [transactionInfo] and validate that it is
included in the [entriesHashList].
     * 2. Compute the hash of the [redactionInfo], if present, and validate that it
is included in the [entriesHashList].
     * 3. Validate the hash of each user revision was correctly computed and
matches the hash published
     * with that revision.
     * 4. Compute the hash of the [revisions] by treating the revision hashes as
the leaf nodes of a Merkle tree
```

```
    * and calculating the root hash of that tree. Then validate that hash is
 included in the [entriesHashList].
    * 5. Compute the hash of the [entriesHashList] by treating the hashes as the
 leaf nodes of a Merkle tree
    * and calculating the root hash of that tree. Then validate that hash matches
 [entriesHash].
    * 6. Finally, compute the block hash by computing the hash resulting from
 concatenating the [entriesHash]
    * and previous block hash, and validate that the result matches the
 [blockHash] provided by QLDB with the block.
    *
    * This method is called by ValidateQldbHashChain::verify for each journal
 block to validate its
    * contents before verifying that the hash chain between consecutive blocks is
 correct.
    */
   public void verifyBlockHash() {
       Set<ByteBuffer> entriesHashSet = new HashSet<>();
       Arrays.stream(entriesHashList).forEach(hash ->
 entriesHashSet.add(wrap(hash).asReadOnlyBuffer()));

       byte[] computedTransactionInfoHash = computeTransactionInfoHash();
       if (!
entriesHashSet.contains(wrap(computedTransactionInfoHash).asReadOnlyBuffer())) {
           throw new IllegalArgumentException(
                   "Block transactionInfo hash is not contained in the QLDB block
 entries hash list.");
       }

       if (redactionInfo != null) {
           byte[] computedRedactionInfoHash = computeRedactionInfoHash();
           if (!
entriesHashSet.contains(wrap(computedRedactionInfoHash).asReadOnlyBuffer())) {
               throw new IllegalArgumentException(
                       "Block redactionInfo hash is not contained in the QLDB
 block entries hash list.");
           }
       }

       if (revisions != null) {
           revisions.forEach(QldbRevision::verifyRevisionHash);
           byte[] computedRevisionsHash = computeRevisionsHash();
           if (!
entriesHashSet.contains(wrap(computedRevisionsHash).asReadOnlyBuffer())) {
```

```
                throw new IllegalArgumentException(
                        "Block revisions list hash is not contained in the QLDB
block entries hash list.");
            }
        }

        byte[] computedEntriesHash = computeEntriesHash();
        if (!Arrays.equals(computedEntriesHash, entriesHash)) {
            throw new IllegalArgumentException("Computed entries hash does not
match entries hash provided in the block.");
        }

        byte[] computedBlockHash = Verifier.dot(computedEntriesHash,
previousBlockHash);
        if (!Arrays.equals(computedBlockHash, blockHash)) {
            throw new IllegalArgumentException("Computed block hash does not match
block hash provided in the block.");
        }
    }

    private byte[] computeTransactionInfoHash() {
        try {
            return
QldbIonUtils.hashIonValue(Constants.MAPPER.writeValueAsIonValue(transactionInfo));
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not compute transactionInfo
hash to verify block hash.", e);
        }
    }

    private byte[] computeRedactionInfoHash() {
        try {
            return
QldbIonUtils.hashIonValue(Constants.MAPPER.writeValueAsIonValue(redactionInfo));
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not compute redactionInfo
hash to verify block hash.", e);
        }
    }

    private byte[] computeRevisionsHash() {
        return
Verifier.calculateMerkleTreeRootHash(revisions.stream().map(QldbRevision::getHash).collect
    }
```

```
    private byte[] computeEntriesHash() {
        return
 Verifier.calculateMerkleTreeRootHash(Arrays.asList(entriesHashList));
    }
}
```

2. 以下のプログラム (ValidateQldbHashChain.java) をコンパイルして実行し、次のステップを実行します。

   1. ジャーナルブロックを vehicle-registration 台帳から **qldb-tutorial-journal-export-*111122223333*** (自分の AWS アカウント 番号で置き換え) という名前の Amazon S3 バケットにエクスポートします。
   2. verifyBlockHash( ) を呼び出して、各ブロック内の個々のハッシュコンポーネントを検証します。
   3. ジャーナルブロック間のハッシュチェーンを検証します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.model.ExportJournalToS3Result;
import com.amazonaws.services.qldb.model.S3EncryptionConfiguration;
import com.amazonaws.services.qldb.model.S3ObjectEncryptionType;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import java.time.Instant;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.GetCallerIdentityRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.tutorial.qldb.JournalBlock;

/**
 * Validate the hash chain of a QLDB ledger by stepping through its S3 export.
 *
 * This code accepts an exportId as an argument, if exportId is passed the code
 * will use that or request QLDB to generate a new export to perform QLDB hash
 * chain validation.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class ValidateQldbHashChain {
    public static final Logger log =
 LoggerFactory.getLogger(ValidateQldbHashChain.class);
    private static final int TIME_SKEW = 20;

    private ValidateQldbHashChain() { }

    /**
     * Export journal contents to a S3 bucket.
     *
     * @return the ExportId of the journal export.
     * @throws InterruptedException if the thread is interrupted while waiting for
 export to complete.
     */
```

```
   private static String createExport() throws InterruptedException {
       String accountId = AWSSecurityTokenServiceClientBuilder.defaultClient()
           .getCallerIdentity(new GetCallerIdentityRequest()).getAccount();
       String bucketName = Constants.JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX + "-" +
accountId;
       String prefix = Constants.LEDGER_NAME + "-" +
Instant.now().getEpochSecond() + "/";

       S3EncryptionConfiguration encryptionConfiguration = new
S3EncryptionConfiguration()
               .withObjectEncryptionType(S3ObjectEncryptionType.SSE_S3);
       ExportJournalToS3Result exportJournalToS3Result =

ExportJournal.createJournalExportAndAwaitCompletion(Constants.LEDGER_NAME,
                   bucketName, prefix, null, encryptionConfiguration,
ExportJournal.DEFAULT_EXPORT_TIMEOUT_MS);

       return exportJournalToS3Result.getExportId();
   }

   /**
    * Validates that the chain hash on the {@link JournalBlock} is valid.
    *
    * @param journalBlocks
    *              {@link JournalBlock} containing hashes to validate.
    * @throws IllegalStateException if previous block hash does not match.
    */
   public static void verify(final List<JournalBlock> journalBlocks) {
       if (journalBlocks.size() == 0) {
           return;
       }

       journalBlocks.stream().reduce(null, (previousJournalBlock, journalBlock) ->
{
           journalBlock.verifyBlockHash();
           if (previousJournalBlock == null) { return journalBlock; }
           if (!Arrays.equals(previousJournalBlock.getBlockHash(),
journalBlock.getPreviousBlockHash())) {
               throw new IllegalStateException("Previous block hash doesn't
match.");
           }
           byte[] blockHash = Verifier.dot(journalBlock.getEntriesHash(),
previousJournalBlock.getBlockHash());
           if (!Arrays.equals(blockHash, journalBlock.getBlockHash())) {
```

```
                throw new IllegalStateException("Block hash doesn't match
 entriesHash dot previousBlockHash, the chain is "
                        + "broken.");
            }
            return journalBlock;
        });
    }

    public static void main(final String... args) throws InterruptedException {
        try {
            String exportId;
            if (args.length == 1) {
                exportId = args[0];
                log.info("Validating QLDB hash chain for exportId: " + exportId);
            } else {
                log.info("Requesting QLDB to create an export.");
                exportId = createExport();
            }
            List<JournalBlock> journalBlocks =

 JournalS3ExportReader.readExport(DescribeJournalExport.describeExport(Constants.LEDGER_NAM
                    exportId), AmazonS3ClientBuilder.defaultClient());
            verify(journalBlocks);
        } catch (Exception e) {
            log.error("Unable to perform hash chain verification.", e);
            throw e;
        }
    }

}
```

vehicle-registration 台帳を使用する必要がなくなった場合は、「ステップ 9 (オプション): リソースをクリーンアップする」に進みます。

ステップ 9 (オプション): リソースをクリーンアップする

vehicle-registration 台帳は引き続き使用できます。ただし、不要になった場合は、削除することをお勧めします。

台帳を削除するには

1. 次のプログラム (DeleteLedger.java) をコンパイルし実行して、vehicle-registration 台帳とその内容のすべてを削除します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.DeleteLedgerRequest;
import com.amazonaws.services.qldb.model.DeleteLedgerResult;
import com.amazonaws.services.qldb.model.ResourceNotFoundException;
import com.amazonaws.services.qldb.model.UpdateLedgerRequest;
import com.amazonaws.services.qldb.model.UpdateLedgerResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
```

```
 * Delete a ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class DeleteLedger {
    public static final Logger log = LoggerFactory.getLogger(DeleteLedger.class);
    public static final Long LEDGER_DELETION_POLL_PERIOD_MS = 20_000L;
    public static AmazonQLDB client = CreateLedger.getClient();

    private DeleteLedger() { }

    public static void main(String... args) throws Exception {
        try {
            setDeletionProtection(Constants.LEDGER_NAME, false);

            delete(Constants.LEDGER_NAME);

            waitForDeleted(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to delete the ledger.", e);
            throw e;
        }
    }

    /**
     * Send a request to the QLDB database to delete the specified ledger.
     *
     * @param ledgerName
     *              Name of the ledger to be deleted.
     * @return DeleteLedgerResult.
     */
    public static DeleteLedgerResult delete(final String ledgerName) {
        log.info("Attempting to delete the ledger with name: {}...", ledgerName);
        DeleteLedgerRequest request = new
 DeleteLedgerRequest().withName(ledgerName);
        DeleteLedgerResult result = client.deleteLedger(request);
        log.info("Success.");
        return result;
    }

    /**
```

```
     * Wait for the ledger to be deleted.
     *
     * @param ledgerName
     *              Name of the ledger being deleted.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static void waitForDeleted(final String ledgerName) throws
 InterruptedException {
        log.info("Waiting for the ledger to be deleted...");
        while (true) {
            try {
                DescribeLedger.describe(ledgerName);
                log.info("The ledger is still being deleted. Please wait...");
                Thread.sleep(LEDGER_DELETION_POLL_PERIOD_MS);
            } catch (ResourceNotFoundException ex) {
                log.info("Success. The ledger is deleted.");
                break;
            }
        }
    }

    public static UpdateLedgerResult setDeletionProtection(String ledgerName,
 boolean deletionProtection) {
        log.info("Let's set deletionProtection to {} for the ledger with name {}",
 deletionProtection, ledgerName);
        UpdateLedgerRequest request = new UpdateLedgerRequest()
                .withName(ledgerName)
                .withDeletionProtection(deletionProtection);

        UpdateLedgerResult result = client.updateLedger(request);
        log.info("Success. Ledger updated: {}", result);
        return result;
    }
}
```

> **Note**
>
> 台帳に対して削除保護が有効になっている場合は、QLDB API を使用して台帳を削除する前に、まず無効にする必要があります。

2. 前のステップでジャーナルデータをエクスポートした場合、不要になったら Simple Storage Service (Amazon S3) コンソールを使用して S3 バケットを削除します。

Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

Amazon QLDB Node.js チュートリアル

このチュートリアルのサンプルアプリケーション実装では、AWS SDK for JavaScript in Node.js と Amazon QLDB ドライバーを使用して QLDB 台帳を作成し、サンプルデータを移入します。

このチュートリアルを進めるときは、管理 API オペレーションについて「AWS SDK for JavaScript API リファレンス」を参照できます。トランザクションデータのオペレーションについては、「Node.js 用 QLDB ドライバー API リファレンス」を参照してください。

> **Note**
>
> 該当する場合、一部のチュートリアルステップでは、サポートされているメジャーバージョンの Node.js 用 QLDB ドライバーに対して異なるコマンドまたはコード例があります。

トピック

Amazon QLDB Node.js サンプルアプリケーションのインストール

このセクションでは、ステップバイステップの Node.js のチュートリアル用に提供されている Amazon QLDB サンプルアプリケーションをインストールして実行する方法について説明します。このサンプルアプリケーションのユースケースは、車両登録に関する完全な履歴情報を追跡する自動車部門 (DMV) データベースです。

Node.js 用の DMV サンプルアプリケーションは、GitHub リポジトリ aws-samples/amazon-qldb-dmv-sample-nodejs でオープンソースとして公開されています。

前提条件

開始する前に、Node.js 用 QLDB ドライバーの「前提条件」を完了していることを確認します。これには、Node.js をインストールし、次の操作を行うことが含まれます。

1. AWS にサインアップする。
2. QLDB の適切なアクセス許可を持つユーザーを作成します。
3. 開発に必要なプログラムへのアクセスを提供します。

このチュートリアルのすべての手順を完了するには、QLDB API を介して台帳リソースへのフル管理アクセス権が必要です。

インストール

サンプルアプリケーションをインストールするには

1. 次のコマンドを入力して、GitHub からサンプルアプリケーションのクローンを作成します。

   2.x

   ```
   git clone https://github.com/aws-samples/amazon-qldb-dmv-sample-nodejs.git
   ```

   1.x

   ```
   git clone -b v1.0.0 https://github.com/aws-samples/amazon-qldb-dmv-sample-nodejs.git
   ```

   サンプルアプリケーションは、このチュートリアルの完全なソースコードとその依存関係 (Node.js ドライバーや AWS SDK for JavaScript in Node.js を含む) をパッケージ化しています。このアプリケーションは、TypeScript で書かれています。

2. `amazon-qldb-dmv-sample-nodejs` パッケージのクローンが作成されているディレクトリに切り替えます。

   ```
   cd amazon-qldb-dmv-sample-nodejs
   ```

3. 依存関係のクリーンインストールを実行します。

```
npm ci
```

4. パッケージをトランスパイルします。

```
npm run build
```

トランスパイルされた JavaScript ファイルが、./dist ディレクトリに書き込まれます。

5. ステップ 1: 新しい台帳を作成する に進み、チュートリアルを開始して台帳を作成します。

ステップ 1: 新しい台帳を作成する

このステップでは、vehicle-registration という名前の新しい Amazon QLDB 台帳を作成します。

新しい台帳を作成するには

1. 次のファイル (Constants.ts) を確認します。このファイルには、このチュートリアル内の他のすべてのプログラムで使用される定数値が含まれています。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

/**
 * Constant values used throughout this tutorial.
 */
export const LEDGER_NAME = "vehicle-registration";
export const LEDGER_NAME_WITH_TAGS = "tags";

export const DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
export const PERSON_TABLE_NAME = "Person";
export const VEHICLE_REGISTRATION_TABLE_NAME = "VehicleRegistration";
export const VEHICLE_TABLE_NAME = "Vehicle";

export const GOV_ID_INDEX_NAME = "GovId";
export const LICENSE_NUMBER_INDEX_NAME = "LicenseNumber";
export const LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
export const PERSON_ID_INDEX_NAME = "PersonId";
export const VIN_INDEX_NAME = "VIN";

export const RETRY_LIMIT = 4;

export const JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-tutorial-journal-export";
export const USER_TABLES = "information_schema.user_tables";
```

2. 次のプログラム (CreateLedger.ts) を使用して、vehicle-registration という名前の台帳を作成します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
```

```
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QLDB } from "aws-sdk";
import {
    CreateLedgerRequest,
    CreateLedgerResponse,
    DescribeLedgerRequest,
    DescribeLedgerResponse
} from "aws-sdk/clients/qldb";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { sleep } from "./qldb/Util";

const LEDGER_CREATION_POLL_PERIOD_MS = 10000;
const ACTIVE_STATE = "ACTIVE";

/**
 * Create a new ledger with the specified name.
 * @param ledgerName Name of the ledger to be created.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a CreateLedgerResponse.
 */
export async function createLedger(ledgerName: string, qldbClient: QLDB):
 Promise<CreateLedgerResponse> {
    log(`Creating a ledger named: ${ledgerName}...`);
    const request: CreateLedgerRequest = {
        Name: ledgerName,
        PermissionsMode: "ALLOW_ALL"
    }
    const result: CreateLedgerResponse = await
 qldbClient.createLedger(request).promise();
    log(`Success. Ledger state: ${result.State}.`);
```

```
    return result;
}

/**
 * Wait for the newly created ledger to become active.
 * @param ledgerName Name of the ledger to be checked on.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a DescribeLedgerResponse.
 */
export async function waitForActive(ledgerName: string, qldbClient: QLDB):
 Promise<DescribeLedgerResponse> {
    log(`Waiting for ledger ${ledgerName} to become active...`);
    const request: DescribeLedgerRequest = {
        Name: ledgerName
    }
    while (true) {
        const result: DescribeLedgerResponse = await
 qldbClient.describeLedger(request).promise();
        if (result.State === ACTIVE_STATE) {
            log("Success. Ledger is active and ready to be used.");
            return result;
        }
        log("The ledger is still creating. Please wait...");
        await sleep(LEDGER_CREATION_POLL_PERIOD_MS);
    }
}

/**
 * Create a ledger and wait for it to be active.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        await createLedger(LEDGER_NAME, qldbClient);
        await waitForActive(LEDGER_NAME, qldbClient);
    } catch (e) {
        error(`Unable to create the ledger: ${e}`);
    }
}

if (require.main === module) {
    main();
```

```
}
```

> **Note**
>
> - `createLedger` の呼び出しで、台帳名とアクセス許可モードを指定する必要があります。台帳データのセキュリティを最大化する、`STANDARD` のアクセス許可モードの使用を推奨します。
> - 台帳を作成すると、デフォルトで削除保護が有効になります。これは QLDB の機能であり、ユーザーによって台帳が削除されるのを防ぎます。QLDB API または AWS Command Line Interface (AWS CLI) を使用して、台帳作成時に削除保護を無効にすることもできます。
> - 必要に応じて、台帳にアタッチするタグを指定することもできます。

3. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/CreateLedger.js
```

新しい台帳への接続を確認するには、「ステップ 2: 台帳への接続をテストする」に進みます。

ステップ 2: 台帳への接続をテストする

このステップでは、トランザクションデータ API エンドポイントを使用して Amazon QLDB の `vehicle-registration` 台帳に接続できることを確認します。

台帳への接続をテストするには

1. 次のプログラム (`ConnectToLedger.ts`) を使用して、`vehicle-registration` 台帳へのデータセッション接続を作成します。

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, RetryConfig  } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/qldbsession";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

const qldbDriver: QldbDriver = createQldbDriver();

/**
 * Create a driver for creating sessions.
 * @param ledgerName The name of the ledger to create the driver on.
 * @param serviceConfigurationOptions The configurations for the AWS SDK client
 that the driver uses.
 * @returns The driver for creating sessions.
 */
export function createQldbDriver(
    ledgerName: string = LEDGER_NAME,
    serviceConfigurationOptions: ClientConfiguration = {}
): QldbDriver {
    const retryLimit = 4;
    const maxConcurrentTransactions = 10;
    //Use driver's default backoff function (and hence, no second parameter
 provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
```

```
    const qldbDriver: QldbDriver = new QldbDriver(ledgerName,
 serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);
    return qldbDriver;
}

export function getQldbDriver(): QldbDriver {
    return qldbDriver;
}

/**
 * Connect to a session for a given ledger using default settings.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        log("Listing table names...");
        const tableNames: string[] = await qldbDriver.getTableNames();
        tableNames.forEach((tableName: string): void => {
            log(tableName);
        });
    } catch (e) {
        error(`Unable to create session: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
```

```
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver  } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/qldbsession";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

const qldbDriver: QldbDriver = createQldbDriver();

/**
 * Create a driver for creating sessions.
 * @param ledgerName The name of the ledger to create the driver on.
 * @param serviceConfigurationOptions The configurations for the AWS SDK client
 that the driver uses.
 * @returns The driver for creating sessions.
 */
export function createQldbDriver(
    ledgerName: string = LEDGER_NAME,
    serviceConfigurationOptions: ClientConfiguration = {}
): QldbDriver {
    const qldbDriver: QldbDriver = new QldbDriver(ledgerName,
 serviceConfigurationOptions);
    return qldbDriver;
}

export function getQldbDriver(): QldbDriver {
    return qldbDriver;
}

/**
 * Connect to a session for a given ledger using default settings.
```

```
 * @returns Promise which fulfills with void.
 */
var main = async function(): Promise<void> {
    try {
        log("Listing table names...");
        const tableNames: string[] = await qldbDriver.getTableNames();
        tableNames.forEach((tableName: string): void => {
            log(tableName);
        });
    } catch (e) {
        error(`Unable to create session: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> 台帳に対してデータトランザクションを実行するには、QLDB ドライバーオブジェクトを作成して、指定した台帳に接続する必要があります。これは、前のステップで台帳の作成に使用した `qldbClient` オブジェクトとは異なるクライアントオブジェクトです。前のクライアントは、「Amazon QLDB API リファレンス」に示されている管理 API オペレーションの実行にのみ使用されます。

2. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/ConnectToLedger.js
```

`vehicle-registration` 台帳にテーブルを作成するには、「ステップ 3: テーブル、インデックス、およびサンプルデータを作成する」に進みます。

ステップ 3: テーブル、インデックス、およびサンプルデータを作成する

Amazon QLDB 台帳がアクティブになり、接続を受け入れると、車両、車両の所有者、登録情報に関するデータのテーブルを作成できるようになります。テーブルとインデックスを作成した後、これらにデータをロードできます。

このステップでは、vehicle-registration 台帳に 4 つのテーブルを作成します。

- VehicleRegistration
- Vehicle
- Person
- DriversLicense

以下のインデックスも作成します。

テーブル名	フィールド
VehicleRegistration	VIN
VehicleRegistration	LicensePlateNumber
Vehicle	VIN
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

サンプルデータを挿入するときは、まず Person テーブルにドキュメントを挿入します。次に、各 Person ドキュメントからシステムによって割り当てられた id を使用して、適切な VehicleRegistration および DriversLicense ドキュメントの対応するフィールドに入力します。

> **Tip**
>
> ベストプラクティスとして、外部キーには、ドキュメントにシステムによって割り当てられた id を使用します。一意の識別子 (車両の VIN など) 用にフィールドを定義することはできますが、実際のドキュメントの一意の識別子はその id です。このフィールドはドキュメントのメタデータに含まれており、コミット済みビュー (テーブルのシステム定義のビュー) でクエリを実行できます。
> QLDB におけるビューの詳細については、「重要な概念」を参照してください。メタデータの詳細については、「ドキュメントのメタデータのクエリの実行」を参照してください。

テーブルとインデックスを作成するには

1. 次のプログラム (CreateTable.ts) を使用して前述のテーブルを作成します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";

import { getQldbDriver } from "./ConnectToLedger";
import {
    DRIVERS_LICENSE_TABLE_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Create multiple tables in a single transaction.
```

```
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to create.
 * @returns Promise which fulfills with the number of changes to the database.
 */
export async function createTable(txn: TransactionExecutor, tableName: string):
 Promise<number> {
    const statement: string = `CREATE TABLE ${tableName}`;
    return await txn.execute(statement).then((result: Result) => {
        log(`Successfully created table ${tableName}.`);
        return result.getResultList().length;
    });
}

/**
 * Create tables in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            Promise.all([
                createTable(txn, VEHICLE_REGISTRATION_TABLE_NAME),
                createTable(txn, VEHICLE_TABLE_NAME),
                createTable(txn, PERSON_TABLE_NAME),
                createTable(txn, DRIVERS_LICENSE_TABLE_NAME)
            ]);
        });
    } catch (e) {
        error(`Unable to create tables: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> このプログラムは、QLDB ドライバーインスタンスで executeLambda 関数を使用する方法を示しています。この例では、1 つの Lambda 式で複数の CREATE TABLE PartiQL ステートメントを実行します。

この execute 関数は、暗黙的にトランザクションを開始し、Lambda ですべてのステートメントを実行して、トランザクションを自動コミットします。

2. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/CreateTable.js
```

3. 次のプログラム (CreateIndex.ts) を使用して前述のテーブルにインデックスを作成します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";

import { getQldbDriver } from "./ConnectToLedger";
import {
    DRIVERS_LICENSE_TABLE_NAME,
    GOV_ID_INDEX_NAME,
    LICENSE_NUMBER_INDEX_NAME,
    LICENSE_PLATE_NUMBER_INDEX_NAME,
    PERSON_ID_INDEX_NAME,
```

```
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME,
    VIN_INDEX_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Create an index for a particular table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to add indexes for.
 * @param indexAttribute Index to create on a single attribute.
 * @returns Promise which fulfills with the number of changes to the database.
 */
export async function createIndex(
    txn: TransactionExecutor,
    tableName: string,
    indexAttribute: string
): Promise<number> {
    const statement: string = `CREATE INDEX on ${tableName} (${indexAttribute})`;
    return await txn.execute(statement).then((result) => {
        log(`Successfully created index ${indexAttribute} on table ${tableName}.`);
        return result.getResultList().length;
    });
}

/**
 * Create indexes on tables in a particular ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            Promise.all([
                createIndex(txn, PERSON_TABLE_NAME, GOV_ID_INDEX_NAME),
                createIndex(txn, VEHICLE_TABLE_NAME, VIN_INDEX_NAME),
                createIndex(txn, VEHICLE_REGISTRATION_TABLE_NAME, VIN_INDEX_NAME),
                createIndex(txn, VEHICLE_REGISTRATION_TABLE_NAME,
 LICENSE_PLATE_NUMBER_INDEX_NAME),
                createIndex(txn, DRIVERS_LICENSE_TABLE_NAME, PERSON_ID_INDEX_NAME),
                createIndex(txn, DRIVERS_LICENSE_TABLE_NAME,
 LICENSE_NUMBER_INDEX_NAME)
            ]);
```

```
        });
    } catch (e) {
        error(`Unable to create indexes: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

4. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/CreateIndex.js
```

データをテーブルにロードするには

1. 以下の .ts ファイルを確認します。

    1. SampleData.ts - vehicle-registration テーブルに挿入するサンプルデータが含まれています。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Decimal } from "ion-js";

const EMPTY_SECONDARY_OWNERS: object[] = [];
export const DRIVERS_LICENSE = [
    {
        PersonId: "",
        LicenseNumber: "LEWISR261LL",
        LicenseType: "Learner",
        ValidFromDate: new Date("2016-12-20"),
        ValidToDate: new Date("2020-11-15")
    },
    {
        PersonId: "",
        LicenseNumber : "LOGANB486CG",
        LicenseType: "Probationary",
        ValidFromDate : new Date("2016-04-06"),
        ValidToDate : new Date("2020-11-15")
    },
    {
        PersonId: "",
        LicenseNumber : "744 849 301",
        LicenseType: "Full",
        ValidFromDate : new Date("2017-12-06"),
        ValidToDate : new Date("2022-10-15")
    },
    {
        PersonId: "",
        LicenseNumber : "P626-168-229-765",
        LicenseType: "Learner",
        ValidFromDate : new Date("2017-08-16"),
        ValidToDate : new Date("2021-11-15")
    },
    {
        PersonId: "",
        LicenseNumber : "S152-780-97-415-0",
        LicenseType: "Probationary",
        ValidFromDate : new Date("2015-08-15"),
        ValidToDate : new Date("2021-08-21")
    }
];
```

```
export const PERSON = [
    {
        FirstName : "Raul",
        LastName : "Lewis",
        DOB : new Date("1963-08-19"),
        Address : "1719 University Street, Seattle, WA, 98109",
        GovId : "LEWISR261LL",
        GovIdType : "Driver License"
    },
    {
        FirstName : "Brent",
        LastName : "Logan",
        DOB : new Date("1967-07-03"),
        Address : "43 Stockert Hollow Road, Everett, WA, 98203",
        GovId : "LOGANB486CG",
        GovIdType : "Driver License"
    },
    {
        FirstName : "Alexis",
        LastName : "Pena",
        DOB : new Date("1974-02-10"),
        Address : "4058 Melrose Street, Spokane Valley, WA, 99206",
        GovId : "744 849 301",
        GovIdType : "SSN"
    },
    {
        FirstName : "Melvin",
        LastName : "Parker",
        DOB : new Date("1976-05-22"),
        Address : "4362 Ryder Avenue, Seattle, WA, 98101",
        GovId : "P626-168-229-765",
        GovIdType : "Passport"
    },
    {
        FirstName : "Salvatore",
        LastName : "Spencer",
        DOB : new Date("1997-11-15"),
        Address : "4450 Honeysuckle Lane, Seattle, WA, 98101",
        GovId : "S152-780-97-415-0",
        GovIdType : "Passport"
    }
];
export const VEHICLE = [
    {
```

```
        VIN : "1N4AL11D75C109151",
        Type : "Sedan",
        Year : 2011,
        Make : "Audi",
        Model : "A5",
        Color : "Silver"
    },
    {
        VIN : "KM8SRDHF6EU074761",
        Type : "Sedan",
        Year : 2015,
        Make : "Tesla",
        Model : "Model S",
        Color : "Blue"
    },
    {
        VIN : "3HGGK5G53FM761765",
        Type : "Motorcycle",
        Year : 2011,
        Make : "Ducati",
        Model : "Monster 1200",
        Color : "Yellow"
    },
    {
        VIN : "1HVBBAANXWH544237",
        Type : "Semi",
        Year : 2009,
        Make : "Ford",
        Model : "F 150",
        Color : "Black"
    },
    {
        VIN : "1C4RJFAG0FC625797",
        Type : "Sedan",
        Year : 2019,
        Make : "Mercedes",
        Model : "CLK 350",
        Color : "White"
    }
];
export const VEHICLE_REGISTRATION = [
    {
        VIN : "1N4AL11D75C109151",
        LicensePlateNumber : "LEWISR261LL",
```

```
        State : "WA",
        City : "Seattle",
        ValidFromDate : new Date("2017-08-21"),
        ValidToDate : new Date("2020-05-11"),
        PendingPenaltyTicketAmount : new Decimal(9025, -2),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "KM8SRDHF6EU074761",
        LicensePlateNumber : "CA762X",
        State : "WA",
        City : "Kent",
        PendingPenaltyTicketAmount : new Decimal(13075, -2),
        ValidFromDate : new Date("2017-09-14"),
        ValidToDate : new Date("2020-06-25"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "3HGGK5G53FM761765",
        LicensePlateNumber : "CD820Z",
        State : "WA",
        City : "Everett",
        PendingPenaltyTicketAmount : new Decimal(44230, -2),
        ValidFromDate : new Date("2011-03-17"),
        ValidToDate : new Date("2021-03-24"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "1HVBBAANXWH544237",
        LicensePlateNumber : "LS477D",
        State : "WA",
        City : "Tacoma",
        PendingPenaltyTicketAmount : new Decimal(4220, -2),
        ValidFromDate : new Date("2011-10-26"),
        ValidToDate : new Date("2023-09-25"),
```

```
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "1C4RJFAG0FC625797",
        LicensePlateNumber : "TH393F",
        State : "WA",
        City : "Olympia",
        PendingPenaltyTicketAmount : new Decimal(3045, -2),
        ValidFromDate : new Date("2013-09-02"),
        ValidToDate : new Date("2024-03-19"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    }
];
```

2. Util.ts - ion-js パッケージからインポートするユーティリティモジュールで、Amazon Ion データを変換、解析、および出力するヘルパー関数を提供します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { GetBlockResponse, GetDigestResponse, ValueHolder } from "aws-sdk/
clients/qldb";
import {
    Decimal,
    decodeUtf8,
    dom,
    IonTypes,
    makePrettyWriter,
    makeReader,
    Reader,
    Timestamp,
    toBase64,
    Writer
} from "ion-js";

import { error } from "./LogUtil";



/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given BlockResponse.
 * @param blockResponse The BlockResponse to convert to string.
 * @returns The string representation of the supplied BlockResponse.
 */
export function blockResponseToString(blockResponse: GetBlockResponse): string {
    let stringBuilder: string = "";
    if (blockResponse.Block.IonText) {
        stringBuilder = stringBuilder + "Block: " + blockResponse.Block.IonText +
 ", ";
    }
    if (blockResponse.Proof.IonText) {
        stringBuilder = stringBuilder + "Proof: " + blockResponse.Proof.IonText;
    }
    stringBuilder = "{" + stringBuilder + "}";
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
```

```
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given GetDigestResponse.
 * @param digestResponse The GetDigestResponse to convert to string.
 * @returns The string representation of the supplied GetDigestResponse.
 */
export function digestResponseToString(digestResponse: GetDigestResponse): string
 {
    let stringBuilder: string = "";
    if (digestResponse.Digest) {
        stringBuilder += "Digest: " + JSON.stringify(toBase64(<Uint8Array>
 digestResponse.Digest)) + ", ";
    }
    if (digestResponse.DigestTipAddress.IonText) {
        stringBuilder += "DigestTipAddress: " +
 digestResponse.DigestTipAddress.IonText;
    }
    stringBuilder = "{" + stringBuilder + "}";
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * Get the document IDs from the given table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName The table name to query.
 * @param field A field to query.
 * @param value The key of the given field.
 * @returns Promise which fulfills with the document ID as a string.
 */
export async function getDocumentId(
    txn: TransactionExecutor,
    tableName: string,
    field: string,
    value: string
): Promise<string> {
    const query: string = `SELECT id FROM ${tableName} AS t BY id WHERE t.
${field} = ?`;
```

```
    let documentId: string = undefined;
    await txn.execute(query, value).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to retrieve document ID using ${value}.`);
        }
        documentId = resultList[0].get("id").stringValue();

    }).catch((err: any) => {
        error(`Error getting documentId: ${err}`);
    });
    return documentId;
}

/**
 * Sleep for the specified amount of time.
 * @param ms The amount of time to sleep in milliseconds.
 * @returns Promise which fulfills with void.
 */
export function sleep(ms: number): Promise<void> {
    return new Promise(resolve => setTimeout(resolve, ms));
}

/**
 * Find the value of a given path in an Ion value. The path should contain a blob
 value.
 * @param value The Ion value that contains the journal block attributes.
 * @param path The path to a certain attribute.
 * @returns Uint8Array value of the blob, or null if the attribute cannot be
 found in the Ion value
 *                  or is not of type Blob
 */
export function getBlobValue(value: dom.Value, path: string): Uint8Array | null {
    const attribute: dom.Value = value.get(path);
    if (attribute !== null && attribute.getType() === IonTypes.BLOB) {
        return attribute.uInt8ArrayValue();
    }
    return null;
}

/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given ValueHolder.
 * @param valueHolder The ValueHolder to convert to string.
```

```
 * @returns The string representation of the supplied ValueHolder.
 */
export function valueHolderToString(valueHolder: ValueHolder): string {
    const stringBuilder: string = `{ IonText: ${valueHolder.IonText}}`;
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * Converts a given value to Ion using the provided writer.
 * @param value The value to convert to Ion.
 * @param ionWriter The Writer to pass the value into.
 * @throws Error: If the given value cannot be converted to Ion.
 */
export function writeValueAsIon(value: any, ionWriter: Writer): void {
    switch (typeof value) {
        case "string":
            ionWriter.writeString(value);
            break;
        case "boolean":
            ionWriter.writeBoolean(value);
            break;
        case "number":
                ionWriter.writeInt(value);
                break;
        case "object":
            if (Array.isArray(value)) {
                // Object is an array.
                ionWriter.stepIn(IonTypes.LIST);

                for (const element of value) {
                    writeValueAsIon(element, ionWriter);
                }

                ionWriter.stepOut();
            } else if (value instanceof Date) {
                // Object is a Date.
                ionWriter.writeTimestamp(Timestamp.parse(value.toISOString()));
            } else if (value instanceof Decimal) {
                // Object is a Decimal.
                ionWriter.writeDecimal(value);
            } else if (value === null) {
```

```
                ionWriter.writeNull(IonTypes.NULL);
            } else {
                // Object is a struct.
                ionWriter.stepIn(IonTypes.STRUCT);

                for (const key of Object.keys(value)) {
                    ionWriter.writeFieldName(key);
                    writeValueAsIon(value[key], ionWriter);
                }
                ionWriter.stepOut();
            }
            break;
        default:
            throw new Error(`Cannot convert to Ion for type: ${(typeof
 value)}.`);
    }
}
```

> **Note**
>
> getDocumentId 関数は、システムによって割り当てられたドキュメント ID をテーブルから返すクエリを実行します。詳細については、BY 句を使用したドキュメント ID のクエリの実行 を参照してください。

2. 次のプログラム (InsertDocument.ts) を使用してサンプルデータをテーブルに挿入します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
```

```
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { DRIVERS_LICENSE, PERSON, VEHICLE, VEHICLE_REGISTRATION } from "./model/
SampleData";
import {
    DRIVERS_LICENSE_TABLE_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Insert the given list of documents into a table in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to insert documents into.
 * @param documents List of documents to insert.
 * @returns Promise which fulfills with a {@linkcode Result} object.
 */
export async function insertDocument(
    txn: TransactionExecutor,
    tableName: string,
    documents: object[]
): Promise<Result> {
    const statement: string = `INSERT INTO ${tableName} ?`;
    const result: Result = await txn.execute(statement, documents);
    return result;
}

/**
 * Handle the insertion of documents and updating PersonIds all in a single
 transaction.
```

```
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @returns Promise which fulfills with void.
 */
async function updateAndInsertDocuments(txn: TransactionExecutor): Promise<void> {
    log("Inserting multiple documents into the 'Person' table...");
    const documentIds: Result = await insertDocument(txn, PERSON_TABLE_NAME,
 PERSON);

    const listOfDocumentIds: dom.Value[] = documentIds.getResultList();
    log("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
 'VehicleRegistration'...");
    updatePersonId(listOfDocumentIds);

    log("Inserting multiple documents into the remaining tables...");
    await Promise.all([
        insertDocument(txn, DRIVERS_LICENSE_TABLE_NAME, DRIVERS_LICENSE),
        insertDocument(txn, VEHICLE_REGISTRATION_TABLE_NAME, VEHICLE_REGISTRATION),
        insertDocument(txn, VEHICLE_TABLE_NAME, VEHICLE)
    ]);
}

/**
 * Update the PersonId value for DriversLicense records and the PrimaryOwner value
 for VehicleRegistration records.
 * @param documentIds List of document IDs.
 */
export function updatePersonId(documentIds: dom.Value[]): void {
    documentIds.forEach((value: dom.Value, i: number) => {
        const documentId: string = value.get("documentId").stringValue();
        DRIVERS_LICENSE[i].PersonId = documentId;
        VEHICLE_REGISTRATION[i].Owners.PrimaryOwner.PersonId = documentId;
    });
}

/**
 * Insert documents into a table in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await updateAndInsertDocuments(txn);
        });
```

```
    } catch (e) {
        error(`Unable to insert documents: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> - このプログラムは、パラメータ化された値を渡して execute 関数を呼び出す方法を示しています。実行する PartiQL ステートメントに加えて、データパラメータを渡すことができます。ステートメント文字列の変数プレースホルダーとして疑問符 (?) を使用します。
> - INSERT ステートメントが成功すると、 挿入された各ドキュメントの id が返されます。

3. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/InsertDocument.js
```

次に、SELECT ステートメントを使用すると、vehicle-registration 台帳のテーブルからデータを読み取ることができます。ステップ 4: 台帳のテーブルにクエリを実行する に進みます。

ステップ 4: 台帳のテーブルにクエリを実行する

Amazon QLDB 台帳にテーブルを作成し、データをロードした後は、クエリを実行して、挿入した車両登録データを確認できます。QLDB は PartiQL をクエリ言語として使用し、Amazon Ion をドキュメント指向のデータモデルとして使用します。

PartiQL は、Ion で動作するように拡張されたオープンソースの SQL 互換のクエリ言語です。PartiQL を使用すると、使い慣れた SQL 演算子を使用してデータを挿入、クエリ、および管理できます。Amazon Ion は JSON のスーパーセットです。Ion はオープンソースのドキュメントベースのデータ形式であり、構造化データ、半構造化データ、およびネストされたデータを柔軟に保存および処理できます。

このステップでは、SELECT ステートメントを使用して、vehicle-registration 台帳のテーブルからデータを読み取ります。

> **Warning**
>
> インデックス付きルックアップなしで QLDB でクエリを実行すると、完全なテーブルスキャンが呼び出されます。PartiQL は SQL 互換であるため、このようなクエリをサポートしています。ただし、QLDB の本番環境のユースケースではテーブルスキャンを実行しないでください。テーブルスキャンより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する必要があります (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`)。詳細については、「クエリパフォーマンスの最適化」を参照してください。

テーブルのクエリを実行するには

1. 次のプログラム (FindVehicles.ts) を使用して台帳の人物に登録されているすべての車両をクエリします。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { PERSON } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";
import { prettyPrintResultList } from "./ScanTable";

/**
 * Query 'Vehicle' and 'VehicleRegistration' tables using a unique document ID in
 one transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param govId The owner's government ID.
 * @returns Promise which fulfills with void.
 */
async function findVehiclesForOwner(txn: TransactionExecutor, govId: string):
 Promise<void> {
    const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME, "GovId",
 govId);
    const query: string = "SELECT Vehicle FROM Vehicle INNER JOIN
 VehicleRegistration AS r " +
                          "ON Vehicle.VIN = r.VIN WHERE
 r.Owners.PrimaryOwner.PersonId = ?";

    await txn.execute(query, documentId).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        log(`List of vehicles for owner with GovId: ${govId}`);
        prettyPrintResultList(resultList);
    });
}

/**
 * Find all vehicles registered under a person.
 * @returns Promise which fulfills with void.
 */
```

```
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await findVehiclesForOwner(txn, PERSON[0].GovId);
        });
    } catch (e) {
        error(`Error getting vehicles for owner: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> まず、このプログラムでは、GovId LEWISR261LL を使用してドキュメントの Person テーブルに対してクエリを実行し、id メタデータフィールドを取得します。
> 次に、このドキュメント id を外部キーとして使用して、PrimaryOwner.PersonId によって VehicleRegistration テーブルをクエリします。また、VehicleRegistration が VIN フィールドの Vehicle テーブルと結合されます。

2. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/FindVehicles.js
```

vehicle-registration 台帳のテーブルのドキュメントの変更方法については、「ステップ 5: 台帳内のドキュメントを変更する」を参照してください。

ステップ 5: 台帳内のドキュメントを変更する

操作するデータを用意できたところで、Amazon QLDB で vehicle-registration 台帳のドキュメントに変更を加えることができます。このステップでは、次のコード例により、データ操作言語 (DML) ステートメントを実行する方法を示します。これらのステートメントは、ある車両の主所有者を更新し、別の車両に第二所有者を追加します。

ドキュメントに変更を加えるには

1. 次のプログラム (TransferVehicleOwnership.ts) を使用して、台帳で VIN が 1N4AL11D75C109151 の車両の主所有者を更新します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { PERSON, VEHICLE } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";

/**
 * Query a driver's information using the given ID.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
```

```
 * @param documentId The unique ID of a document in the Person table.
 * @returns Promise which fulfills with an Ion value containing the person.
 */
export async function findPersonFromDocumentId(txn: TransactionExecutor,
 documentId: string): Promise<dom.Value> {
    const query: string = "SELECT p.* FROM Person AS p BY pid WHERE pid = ?";

    let personId: dom.Value;
    await txn.execute(query, documentId).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to find person with ID: ${documentId}.`);
        }
        personId = resultList[0];
    });
    return personId;
}

/**
 * Find the primary owner for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN to find primary owner for.
 * @returns Promise which fulfills with an Ion value containing the primary owner.
 */
export async function findPrimaryOwnerForVehicle(txn: TransactionExecutor, vin:
 string): Promise<dom.Value> {
    log(`Finding primary owner for vehicle with VIN: ${vin}`);
    const query: string = "SELECT Owners.PrimaryOwner.PersonId FROM
 VehicleRegistration AS v WHERE v.VIN = ?";

    let documentId: string = undefined;
    await txn.execute(query, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to retrieve document ID using ${vin}.`);
        }
        const PersonIdValue: dom.Value = resultList[0].get("PersonId");
        if (PersonIdValue === null) {
            throw new Error(`Expected field name PersonId not found.`);
        }
        documentId = PersonIdValue.stringValue();
    });
    return findPersonFromDocumentId(txn, documentId);
}
```

```
/**
 * Update the primary owner for a vehicle using the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN for the vehicle to operate on.
 * @param documentId New PersonId for the primary owner.
 * @returns Promise which fulfills with void.
 */
async function updateVehicleRegistration(txn: TransactionExecutor, vin: string,
 documentId: string): Promise<void> {
    const statement: string = "UPDATE VehicleRegistration AS r SET
 r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?";

    log(`Updating the primary owner for vehicle with VIN: ${vin}...`);
    await txn.execute(statement, documentId, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error("Unable to transfer vehicle, could not find
 registration.");
        }
        log(`Successfully transferred vehicle with VIN ${vin} to new owner.`);
    });
}

/**
 * Validate the current owner of the given vehicle and transfer its ownership to a
 new owner in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN of the vehicle to transfer ownership of.
 * @param currentOwner The GovId of the current owner of the vehicle.
 * @param newOwner The GovId of the new owner of the vehicle.
 */
export async function validateAndUpdateRegistration(
    txn: TransactionExecutor,
    vin: string,
    currentOwner: string,
    newOwner: string
): Promise<void> {
    const primaryOwner: dom.Value = await findPrimaryOwnerForVehicle(txn, vin);
    const govIdValue: dom.Value = primaryOwner.get("GovId");
    if (govIdValue !== null && govIdValue.stringValue() !== currentOwner) {
        log("Incorrect primary owner identified for vehicle, unable to transfer.");
    }
    else {
```

```
        const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME,
 "GovId", newOwner);
        await updateVehicleRegistration(txn, vin, documentId);
        log("Successfully transferred vehicle ownership!");
    }
}

/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();

        const vin: string = VEHICLE[0].VIN;
        const previousOwnerGovId: string = PERSON[0].GovId;
        const newPrimaryOwnerGovId: string = PERSON[1].GovId;

        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await validateAndUpdateRegistration(txn, vin, previousOwnerGovId,
 newPrimaryOwnerGovId);
        });
    } catch (e) {
        error(`Unable to connect and run queries: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

2. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/TransferVehicleOwnership.js
```

3. 次のプログラム (AddSecondaryOwner.ts) を使用して、台帳で VIN が KM8SRDHF6EU074761 の車両に第二所有者を追加します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
```

```
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { PERSON, VEHICLE_REGISTRATION } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";
import { prettyPrintResultList } from "./ScanTable";

/**
 * Add a secondary owner into 'VehicleRegistration' table for a particular VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN of the vehicle to query.
 * @param secondaryOwnerId The secondary owner's person ID.
 * @returns Promise which fulfills with void.
 */
export async function addSecondaryOwner(
    txn: TransactionExecutor,
    vin: string,
    secondaryOwnerId: string
```

```
): Promise<void> {
    log(`Inserting secondary owner for vehicle with VIN: ${vin}`);
    const query: string =
        `FROM VehicleRegistration AS v WHERE v.VIN = ? INSERT INTO
 v.Owners.SecondaryOwners VALUE ?`;

    const personToInsert = {PersonId: secondaryOwnerId};
    await txn.execute(query, vin, personToInsert).then(async (result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        log("VehicleRegistration Document IDs which had secondary owners added: ");
        prettyPrintResultList(resultList);
    });
}

/**
 * Query for a document ID with a government ID.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param governmentId The government ID to query with.
 * @returns Promise which fulfills with the document ID as a string.
 */
export async function getDocumentIdByGovId(txn: TransactionExecutor, governmentId:
 string): Promise<string> {
    const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME, "GovId",
 governmentId);
    return documentId;
}

/**
 * Check whether a driver has already been registered for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN of the vehicle to query.
 * @param secondaryOwnerId The secondary owner's person ID.
 * @returns Promise which fulfills with a boolean.
 */
export async function isSecondaryOwnerForVehicle(
    txn: TransactionExecutor,
    vin: string,
    secondaryOwnerId: string
): Promise<boolean> {
    log(`Finding secondary owners for vehicle with VIN: ${vin}`);
    const query: string = "SELECT Owners.SecondaryOwners FROM VehicleRegistration
 AS v WHERE v.VIN = ?";

    let doesExist: boolean = false;
```

```
    await txn.execute(query, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();

        resultList.forEach((value: dom.Value) => {
            const secondaryOwnersList: dom.Value[] =
 value.get("SecondaryOwners").elements();

            secondaryOwnersList.forEach((secondaryOwner) => {
                const personId: dom.Value = secondaryOwner.get("PersonId");
                if (personId !== null &&  personId.stringValue() ===
 secondaryOwnerId) {
                    doesExist = true;
                }
            });
        });
    });
    return doesExist;
}

/**
 * Finds and adds secondary owners for a vehicle.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        const vin: string = VEHICLE_REGISTRATION[1].VIN;
        const govId: string = PERSON[0].GovId;

        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            const documentId: string = await getDocumentIdByGovId(txn, govId);

            if (await isSecondaryOwnerForVehicle(txn, vin, documentId)) {
                log(`Person with ID ${documentId} has already been added as a
 secondary owner of this vehicle.`);
            } else {
                await addSecondaryOwner(txn, vin, documentId);
            }
        });

        log("Secondary owners successfully updated.");
    } catch (e) {
        error(`Unable to add secondary owner: ${e}`);
```

```
    }
}

if (require.main === module) {
    main();
}
```

4. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/AddSecondaryOwner.js
```

これらの変更を vehicle-registration 台帳で確認するには、「ステップ 6: ドキュメントのリビジョン履歴を表示する」を参照してください。

ステップ 6: ドキュメントのリビジョン履歴を表示する

前のステップで車両の登録データを変更すると、登録されているすべての所有者とその他の更新されたフィールドの履歴に対してクエリを実行できます。このステップでは、vehicle-registration 台帳の VehicleRegistration テーブルに含まれているドキュメントのリビジョン履歴のクエリを実行します。

リビジョン履歴を表示するには

1. 次のプログラム (QueryHistory.ts) を使用して、VIN が 1N4AL11D75C109151 の VehicleRegistration ドキュメントのリビジョン履歴をクエリします。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { VEHICLE_REGISTRATION } from "./model/SampleData";
import { VEHICLE_REGISTRATION_TABLE_NAME } from "./qldb/Constants";
import { prettyPrintResultList } from "./ScanTable";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";

/**
 * Find previous primary owners for the given VIN in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN to find previous primary owners for.
 * @returns Promise which fulfills with void.
 */
async function previousPrimaryOwners(txn: TransactionExecutor, vin: string):
 Promise<void> {
    const documentId: string = await getDocumentId(txn,
 VEHICLE_REGISTRATION_TABLE_NAME, "VIN", vin);
    const todaysDate: Date = new Date();
    // set todaysDate back one minute to ensure end time is in the past
    // by the time the request reaches our backend
    todaysDate.setMinutes(todaysDate.getMinutes() - 1);
    const threeMonthsAgo: Date = new Date(todaysDate);
    threeMonthsAgo.setMonth(todaysDate.getMonth() - 3);

    const query: string =
        `SELECT data.Owners.PrimaryOwner, metadata.version FROM history ` +
        `(${VEHICLE_REGISTRATION_TABLE_NAME}, \`${threeMonthsAgo.toISOString()}\`,
 \`${todaysDate.toISOString()}\`) ` +
        `AS h WHERE h.metadata.id = ?`;
```

```
    await txn.execute(query, documentId).then((result: Result) => {
        log(`Querying the 'VehicleRegistration' table's history using VIN:
 ${vin}.`);
        const resultList: dom.Value[] = result.getResultList();
        prettyPrintResultList(resultList);
    });
}

/**
 * Query a table's history for a particular set of documents.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        const vin: string = VEHICLE_REGISTRATION[0].VIN;
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await previousPrimaryOwners(txn, vin);
        });
    } catch (e) {
        error(`Unable to query history to find previous owners: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

Note

- 以下の構文で組み込みの「履歴関数」のクエリを実行することで、ドキュメントのリビジョン履歴を表示できます。

```
SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
[ WHERE h.metadata.id = 'id' ]
```

- start-time および end-time はいずれもオプションです。これらは、バックティック (`...`) で示すことができる Amazon Ion リテラル値です。詳細については、「Amazon QLDB での PartiQL による Ion のクエリ」を参照してください。

- ベストプラクティスとして、履歴クエリは日付範囲 (start-time および end-time) とドキュメント ID (`metadata.id`) の両方で修飾します。QLDB はトランザクション内の SELECT クエリを処理します。これらは、トランザクションタイムアウト制限が適用されます。

  QLDB 履歴はドキュメント ID によってインデックス付けされるため、現時点では追加の履歴インデックスを作成することはできません。開始時刻と終了時刻を含む履歴クエリでは、日付範囲修飾のメリットが得られます。

2. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/QueryHistory.js
```

`vehicle-registration` 台帳のドキュメントリビジョンを暗号的に検証するには、「ステップ 7: 台帳内のドキュメントを検証する」に進みます。

ステップ 7: 台帳内のドキュメントを検証する

Amazon QLDB では、SHA-256 の暗号的ハッシュを使用して、台帳のジャーナルのドキュメントの整合性を効率的に検証できます。検証と暗号的ハッシュが QLDB でどのように機能するかについては、「Amazon QLDB でのデータ検証」を参照してください。

このステップでは、`vehicle-registration` 台帳の `VehicleRegistration` テーブルのドキュメントリビジョンを確認します。まず、ダイジェストをリクエストします。ダイジェストは出力ファイルとして返され、台帳の変更履歴全体の署名として機能します。次に、そのダイジェストに関連するリビジョンの証明をリクエストします。この証明を使用して、すべての検証チェックに合格すると、リビジョンの整合性が検証されます。

ドキュメントのリビジョンを検証するには

1. 以下の `.ts` ファイルを確認します。これらのファイルには、検証に必要な QLDB オブジェクトが含まれています。

   1. `BlockAddress.ts`

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { ValueHolder } from "aws-sdk/clients/qldb";
import { dom, IonTypes } from "ion-js";

export class BlockAddress {
    _strandId: string;
    _sequenceNo: number;

    constructor(strandId: string, sequenceNo: number) {
        this._strandId = strandId;
        this._sequenceNo = sequenceNo;
    }
}

/**
 * Convert a block address from an Ion value into a ValueHolder.
 * Shape of the ValueHolder must be: {'IonText': "{strandId: <"strandId">,
 sequenceNo: <sequenceNo>}"}
 * @param value The Ion value that contains the block address values to convert.
 * @returns The ValueHolder that contains the strandId and sequenceNo.
 */
export function blockAddressToValueHolder(value: dom.Value): ValueHolder {
    const blockAddressValue : dom.Value = getBlockAddressValue(value);
    const strandId: string = getStrandId(blockAddressValue);
```

```
    const sequenceNo: number = getSequenceNo(blockAddressValue);
    const valueHolder: string = `{strandId: "${strandId}", sequenceNo:
 ${sequenceNo}}`;
    const blockAddress: ValueHolder = {IonText: valueHolder};
    return blockAddress;
}

/**
 * Helper method that to get the Metadata ID.
 * @param value The Ion value.
 * @returns The Metadata ID.
 */
export function getMetadataId(value: dom.Value): string {
    const metaDataId: dom.Value = value.get("id");
    if (metaDataId === null) {
        throw new Error(`Expected field name id, but not found.`);
    }
    return metaDataId.stringValue();
}

/**
 * Helper method to get the Sequence No.
 * @param value The Ion value.
 * @returns The Sequence No.
 */
export function getSequenceNo(value : dom.Value): number {
    const sequenceNo: dom.Value = value.get("sequenceNo");
    if (sequenceNo === null) {
        throw new Error(`Expected field name sequenceNo, but not found.`);
    }
    return sequenceNo.numberValue();
}

/**
 * Helper method to get the Strand ID.
 * @param value The Ion value.
 * @returns The Strand ID.
 */
export function getStrandId(value: dom.Value): string {
    const strandId: dom.Value = value.get("strandId");
    if (strandId === null) {
        throw new Error(`Expected field name strandId, but not found.`);
    }
    return strandId.stringValue();
```

```
}

export function getBlockAddressValue(value: dom.Value) : dom.Value {
    const type = value.getType();
    if (type !== IonTypes.STRUCT) {
        throw new Error(`Unexpected format: expected struct, but got IonType:
 ${type.name}`);
    }
    const blockAddress: dom.Value = value.get("blockAddress");
    if (blockAddress == null) {
        throw new Error(`Expected field name blockAddress, but not found.`);
    }
    return blockAddress;
}
```

2. Verifier.ts

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Digest, ValueHolder } from "aws-sdk/clients/qldb";
import { createHash } from "crypto";
```

```
import { dom, toBase64 } from "ion-js";

import { getBlobValue } from "./Util";

const HASH_LENGTH: number = 32;
const UPPER_BOUND: number = 8;

/**
 * Build the candidate digest representing the entire ledger from the Proof
 hashes.
 * @param proof The Proof object.
 * @param leafHash The revision hash to pair with the first hash in the Proof
 hashes list.
 * @returns The calculated root hash.
 */
function buildCandidateDigest(proof: ValueHolder, leafHash: Uint8Array):
 Uint8Array {
    const parsedProof: Uint8Array[] = parseProof(proof);
    const rootHash: Uint8Array = calculateRootHashFromInternalHash(parsedProof,
 leafHash);
    return rootHash;
}

/**
 * Combine the internal hashes and the leaf hash until only one root hash
 remains.
 * @param internalHashes An array of hash values.
 * @param leafHash The revision hash to pair with the first hash in the Proof
 hashes list.
 * @returns The root hash constructed by combining internal hashes.
 */
function calculateRootHashFromInternalHash(internalHashes: Uint8Array[],
 leafHash: Uint8Array): Uint8Array {
    const rootHash: Uint8Array = internalHashes.reduce(joinHashesPairwise,
 leafHash);
    return rootHash;
}

/**
 * Compare two hash values by converting each Uint8Array byte, which is unsigned
 by default,
 * into a signed byte, assuming they are little endian.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
```

```
 * @returns Zero if the hash values are equal, otherwise return the difference of
 the first pair of non-matching bytes.
 */
function compareHashValues(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_LENGTH || hash2.length !== HASH_LENGTH) {
        throw new Error("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of array to concatenate, in the order provided.
 * @returns The concatenated array.
 */
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}

/**
 * Flip a single random bit in the given hash value.
 * This method is intended to be used for purpose of demonstrating the QLDB
 verification features only.
 * @param original The hash value to alter.
 * @returns The altered hash with a single random bit changed.
 */
export function flipRandomBit(original: any): Uint8Array {
    if (original.length === 0) {
```

```
        throw new Error("Array cannot be empty!");
    }
    const bytePos: number = Math.floor(Math.random() * original.length);
    const bitShift: number = Math.floor(Math.random() * UPPER_BOUND);
    const alteredHash: Uint8Array = original;

    alteredHash[bytePos] = alteredHash[bytePos] ^ (1 << bitShift);
    return alteredHash;
}

/**
 * Take two hash values, sort them, concatenate them, and generate a new hash
 value from the concatenated values.
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The concatenated array of hashes.
 */
export function joinHashesPairwise(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }
    let concat: Uint8Array;
    if (compareHashValues(h1, h2) < 0) {
        concat = concatenate(h1, h2);
    } else {
        concat = concatenate(h2, h1);
    }
    const hash = createHash('sha256');
    hash.update(concat);
    const newDigest: Uint8Array = hash.digest();
    return newDigest;
}

/**
 * Parse the Block object returned by QLDB and retrieve block hash.
 * @param valueHolder A structure containing an Ion string value.
 * @returns The block hash.
 */
export function parseBlock(valueHolder: ValueHolder): Uint8Array {
    const block: dom.Value = dom.load(valueHolder.IonText);
    const blockHash: Uint8Array = getBlobValue(block, "blockHash");
```

```
    return blockHash;
}

/**
 * Parse the Proof object returned by QLDB into an iterator.
 * The Proof object returned by QLDB is a dictionary like the following:
 * {'IonText': '[{{<hash>}},{{<hash>}}]'}
 * @param valueHolder A structure containing an Ion string value.
 * @returns A list of hash values.
 */
function parseProof(valueHolder: ValueHolder): Uint8Array[] {
    const proofs : dom.Value = dom.load(valueHolder.IonText);
    return proofs.elements().map(proof => proof.uInt8ArrayValue());
}

/**
 *  Verify document revision against the provided digest.
 * @param documentHash The SHA-256 value representing the document revision to be
 verified.
 * @param digest The SHA-256 hash value representing the ledger digest.
 * @param proof The Proof object retrieved from GetRevision.getRevision.
 * @returns If the document revision verifies against the ledger digest.
 */
export function verifyDocument(documentHash: Uint8Array, digest: Digest, proof:
 ValueHolder): boolean {
    const candidateDigest = buildCandidateDigest(proof, documentHash);
    return (toBase64(<Uint8Array> digest) === toBase64(candidateDigest));
}
```

2. 2 つの .ts プログラム (GetDigest.ts および GetRevision.ts) を使用して、次の手順を実行します。

   - vehicle-registration 台帳に新しいダイジェストをリクエストします。
   - VehicleRegistration テーブルの VIN が 1N4AL11D75C109151 のドキュメントについて、各リビジョンの証明をリクエストします。
   - 返されたダイジェストと証明を使用して、ダイジェストを再計算することで、リビジョンを検証します。

   GetDigest.ts プログラムには、次のコードが含まれています。

```
/*
```

```
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QLDB } from "aws-sdk";
import { GetDigestRequest, GetDigestResponse } from "aws-sdk/clients/qldb";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { digestResponseToString } from "./qldb/Util";

/**
 * Get the digest of a ledger's journal.
 * @param ledgerName Name of the ledger to operate on.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a GetDigestResponse.
 */
export async function getDigestResult(ledgerName: string, qldbClient: QLDB):
 Promise<GetDigestResponse> {
    const request: GetDigestRequest = {
        Name: ledgerName
    };
    const result: GetDigestResponse = await
 qldbClient.getDigest(request).promise();
```

```
    return result;
}

/**
 * This is an example for retrieving the digest of a particular ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        log(`Retrieving the current digest for ledger: ${LEDGER_NAME}.`);
        const digest: GetDigestResponse = await getDigestResult(LEDGER_NAME,
 qldbClient);
        log(`Success. Ledger digest: \n${digestResponseToString(digest)}.`);
    } catch (e) {
        error(`Unable to get a ledger digest: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> getDigest 関数を使用して、台帳のジャーナルの現在のティップを含むダイジェストをリクエストします。ジャーナルのティップとは、QLDB がリクエストを受けた時点でコミット済みの最新のブロックのことです。

GetRevision.ts プログラムには、次のコードが含まれています。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
```

```
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { QLDB } from "aws-sdk";
import { Digest, GetDigestResponse, GetRevisionRequest, GetRevisionResponse,
 ValueHolder } from "aws-sdk/clients/qldb";
import { dom, toBase64 } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { getDigestResult } from './GetDigest';
import { VEHICLE_REGISTRATION } from "./model/SampleData"
import { blockAddressToValueHolder, getMetadataId } from './qldb/BlockAddress';
import { LEDGER_NAME } from './qldb/Constants';
import { error, log } from "./qldb/LogUtil";
import { getBlobValue, valueHolderToString } from "./qldb/Util";
import { flipRandomBit, verifyDocument } from "./qldb/Verifier";

/**
 * Get the revision data object for a specified document ID and block address.
 * Also returns a proof of the specified revision for verification.
 * @param ledgerName Name of the ledger containing the document to query.
 * @param documentId Unique ID for the document to be verified, contained in the
 committed view of the document.
 * @param blockAddress The location of the block to request.
 * @param digestTipAddress The latest block location covered by the digest.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a GetRevisionResponse.
 */
async function getRevision(
    ledgerName: string,
    documentId: string,
```

```
    blockAddress: ValueHolder,
    digestTipAddress: ValueHolder,
    qldbClient: QLDB
): Promise<GetRevisionResponse> {
    const request: GetRevisionRequest = {
        Name: ledgerName,
        BlockAddress: blockAddress,
        DocumentId: documentId,
        DigestTipAddress: digestTipAddress
    };
    const result: GetRevisionResponse = await
 qldbClient.getRevision(request).promise();
    return result;
}

/**
 * Query the table metadata for a particular vehicle for verification.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN to query the table metadata of a specific registration with.
 * @returns Promise which fulfills with a list of Ion values that contains the
 results of the query.
 */
export async function lookupRegistrationForVin(txn: TransactionExecutor, vin:
 string): Promise<dom.Value[]> {
    log(`Querying the 'VehicleRegistration' table for VIN: ${vin}...`);
    let resultList: dom.Value[];
    const query: string = "SELECT blockAddress, metadata.id FROM
 _ql_committed_VehicleRegistration WHERE data.VIN = ?";

    await txn.execute(query, vin).then(function(result) {
      resultList = result.getResultList();
    });
    return resultList;
}

/**
 * Verify each version of the registration for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param ledgerName The ledger to get the digest from.
 * @param vin VIN to query the revision history of a specific registration with.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 * @throws Error: When verification fails.
 */
```

```
export async function verifyRegistration(
    txn: TransactionExecutor,
    ledgerName: string,
    vin: string,
    qldbClient: QLDB
): Promise<void> {
    log(`Let's verify the registration with VIN = ${vin}, in ledger =
 ${ledgerName}.`);
    const digest: GetDigestResponse = await getDigestResult(ledgerName,
 qldbClient);
    const digestBytes: Digest = digest.Digest;
    const digestTipAddress: ValueHolder = digest.DigestTipAddress;

    log(
        `Got a ledger digest: digest tip address = \n
${valueHolderToString(digestTipAddress)},
        digest = \n${toBase64(<Uint8Array> digestBytes)}.`
    );
    log(`Querying the registration with VIN = ${vin} to verify each version of the
 registration...`);
    const resultList: dom.Value[] = await lookupRegistrationForVin(txn, vin);
    log("Getting a proof for the document.");

    for (const result of resultList) {
        const blockAddress: ValueHolder =  blockAddressToValueHolder(result);
        const documentId: string = getMetadataId(result);

        const revisionResponse: GetRevisionResponse = await getRevision(
            ledgerName,
            documentId,
            blockAddress,
            digestTipAddress,
            qldbClient
        );

        const revision: dom.Value = dom.load(revisionResponse.Revision.IonText);
        const documentHash: Uint8Array = getBlobValue(revision, "hash");
        const proof: ValueHolder = revisionResponse.Proof;
        log(`Got back a proof: ${valueHolderToString(proof)}.`);

        let verified: boolean = verifyDocument(documentHash, digestBytes, proof);
        if (!verified) {
           throw new Error("Document revision is not verified.");
        } else {
```

```
            log("Success! The document is verified.");
        }
        const alteredDocumentHash: Uint8Array = flipRandomBit(documentHash);

        log(
            `Flipping one bit in the document's hash and assert that the document
 is NOT verified.
            The altered document hash is: ${toBase64(alteredDocumentHash)}`
        );
        verified = verifyDocument(alteredDocumentHash, digestBytes, proof);

        if (verified) {
            throw new Error("Expected altered document hash to not be verified
 against digest.");
        } else {
            log("Success! As expected flipping a bit in the document hash causes
 verification to fail.");
        }
        log(`Finished verifying the registration with VIN = ${vin} in ledger =
 ${ledgerName}.`);
    }
}

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        const qldbDriver: QldbDriver = getQldbDriver();

        const registration = VEHICLE_REGISTRATION[0];
        const vin: string = registration.VIN;

        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await verifyRegistration(txn, LEDGER_NAME, vin, qldbClient);
        });
    } catch (e) {
        error(`Unable to verify revision: ${e}`);
    }
}

if (require.main === module) {
```

```
    main();
}
```

> **Note**
>
> getRevision 関数から指定したドキュメントリビジョンの証明が返されると、このプログラムはクライアント側 API を使用してそのリビジョンを検証します。

3. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/GetRevision.js
```

vehicle-registration 台帳を使用する必要がなくなった場合は、「ステップ 8 (オプション): リソースをクリーンアップする」に進みます。

ステップ 8 (オプション): リソースをクリーンアップする

vehicle-registration 台帳は引き続き使用できます。ただし、不要になった場合は、削除することをお勧めします。

台帳を削除するには

1. 次のプログラム (DeleteLedger.ts) を使用して vehicle-registration 台帳とその内容全体を削除します。

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { isResourceNotFoundException } from "amazon-qldb-driver-nodejs";
import { AWSError, QLDB } from "aws-sdk";
import { DeleteLedgerRequest, DescribeLedgerRequest } from "aws-sdk/clients/qldb";

import { setDeletionProtection } from "./DeletionProtection";
import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { sleep } from "./qldb/Util";

const LEDGER_DELETION_POLL_PERIOD_MS = 20000;

/**
 * Send a request to QLDB to delete the specified ledger.
 * @param ledgerName Name of the ledger to be deleted.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */
export async function deleteLedger(ledgerName: string, qldbClient: QLDB):
 Promise<void> {
    log(`Attempting to delete the ledger with name: ${ledgerName}`);
    const request: DeleteLedgerRequest = {
        Name: ledgerName
    };
    await qldbClient.deleteLedger(request).promise();
    log("Success.");
}

/**
 * Wait for the ledger to be deleted.
 * @param ledgerName Name of the ledger to be deleted.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */
```

```
export async function waitForDeleted(ledgerName: string, qldbClient: QLDB):
 Promise<void> {
    log("Waiting for the ledger to be deleted...");
    const request: DescribeLedgerRequest = {
        Name: ledgerName
    };
    let isDeleted: boolean = false;
    while (true) {
        await qldbClient.describeLedger(request).promise().catch((error: AWSError)
 => {
            if (isResourceNotFoundException(error)) {
                isDeleted = true;
                log("Success. Ledger is deleted.");
            }
        });
        if (isDeleted) {
            break;
        }
        log("The ledger is still being deleted. Please wait...");
        await sleep(LEDGER_DELETION_POLL_PERIOD_MS);
    }
}

/**
 * Delete a ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        await setDeletionProtection(LEDGER_NAME, qldbClient, false);
        await deleteLedger(LEDGER_NAME, qldbClient);
        await waitForDeleted(LEDGER_NAME, qldbClient);
    } catch (e) {
        error(`Unable to delete the ledger: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> 台帳に対して削除保護が有効になっている場合は、QLDB API を使用して台帳を削除する前に、まず無効にする必要があります。

2. トランスパイルされたプログラムを実行するには、次のコマンドを入力します。

```
node dist/DeleteLedger.js
```

Amazon QLDB Python チュートリアル

このチュートリアルのサンプルアプリケーション実装では、AWS SDK for Python (Boto3) と Amazon QLDB ドライバーを使用して QLDB 台帳を作成し、サンプルデータを移入します。

このチュートリアルを進めるときは、管理 API のオペレーションについて「AWS SDK for Python (Boto3) API リファレンス」の「QLDB low-level client」を参照できます。トランザクションデータのオペレーションについては、「Python 用 QLDB ドライバー API リファレンス」を参照してください。

> **Note**
>
> 該当する場合、一部のチュートリアルステップでは、サポートされているメジャーバージョンの Python 用 QLDB ドライバーに対して異なるコマンドまたはコード例があります。

トピック

- ステップ 8 (オプション): リソースをクリーンアップする

Amazon QLDB Python サンプルアプリケーションのインストール

このセクションでは、ステップバイステップの Python チュートリアル用に提供されている Amazon QLDB サンプルアプリケーションをインストールして実行する方法について説明します。このサンプルアプリケーションのユースケースは、車両登録に関する完全な履歴情報を追跡する自動車部門 (DMV) データベースです。

Python 用の DMV サンプルアプリケーションは、GitHub リポジトリ aws-samples/amazon-qldb-dmv-sample-python でオープンソースとして公開されています。

前提条件

開始する前に、Python 用 QLDB ドライバーの「前提条件」を完了していることを確認します。これには、Python をインストールし、次の操作を行うことが含まれます。

1. AWS にサインアップする。
2. QLDB の適切なアクセス許可を持つユーザーを作成します。
3. 開発に必要なプログラムへのアクセスを提供します。

このチュートリアルのすべての手順を完了するには、QLDB API を介して台帳リソースへのフル管理アクセス権が必要です。

インストール

サンプルアプリケーションをインストールするには

1. 次の `pip` コマンドを入力して、GitHub からサンプルアプリケーションをクローン作成してインストールします。

   3.x

   ```
   pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-python.git
   ```

   2.x

   ```
   pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-
   python.git@v1.0.0
   ```

サンプルアプリケーションは、このチュートリアルからの完全なソースコードと、その依存関係 (Python ドライバーや AWS SDK for Python (Boto3) を含む) をパッケージ化しています。

2. コマンドラインでコードの実行を開始する前に、現在の作業ディレクトリを pyqldbsamples パッケージがインストールされている場所に切り替えます。次のコマンドを入力します。

```
cd $(python -c "import pyqldbsamples; print(pyqldbsamples.__path__[0])")
```

3. ステップ 1: 新しい台帳を作成する に進み、チュートリアルを開始して台帳を作成します。

ステップ 1: 新しい台帳を作成する

このステップでは、vehicle-registration という名前の新しい Amazon QLDB 台帳を作成します。

新しい台帳を作成するには

1. 次のファイル (constants.py) を確認します。このファイルには、このチュートリアル内の他のすべてのプログラムで使用される定数値が含まれています。

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
class Constants:
    """
    Constant values used throughout this tutorial.
    """
    LEDGER_NAME = "vehicle-registration"

    VEHICLE_REGISTRATION_TABLE_NAME = "VehicleRegistration"
    VEHICLE_TABLE_NAME = "Vehicle"
    PERSON_TABLE_NAME = "Person"
    DRIVERS_LICENSE_TABLE_NAME = "DriversLicense"

    LICENSE_NUMBER_INDEX_NAME = "LicenseNumber"
    GOV_ID_INDEX_NAME = "GovId"
    VEHICLE_VIN_INDEX_NAME = "VIN"
    LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber"
    PERSON_ID_INDEX_NAME = "PersonId"

    JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-tutorial-journal-export"
    USER_TABLES = "information_schema.user_tables"
    S3_BUCKET_ARN_TEMPLATE = "arn:aws:s3:::"
    LEDGER_NAME_WITH_TAGS = "tags"

    RETRY_LIMIT = 4
```

2. 次のプログラム (create_ledger.py) を使用して、vehicle-registration という名前の台帳を作成します。

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
```

```
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO
from time import sleep

from boto3 import client

from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

LEDGER_CREATION_POLL_PERIOD_SEC = 10
ACTIVE_STATE = "ACTIVE"


def create_ledger(name):
    """
    Create a new ledger with the specified name.

    :type name: str
    :param name: Name for the ledger to be created.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info("Let's create the ledger named: {}...".format(name))
    result = qldb_client.create_ledger(Name=name, PermissionsMode='ALLOW_ALL')
    logger.info('Success. Ledger state: {}.'.format(result.get('State')))
    return result


def wait_for_active(name):
    """
```

```
    Wait for the newly created ledger to become active.

    :type name: str
    :param name: The ledger to check on.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info('Waiting for ledger to become active...')
    while True:
        result = qldb_client.describe_ledger(Name=name)
        if result.get('State') == ACTIVE_STATE:
            logger.info('Success. Ledger is active and ready to use.')
            return result
        logger.info('The ledger is still creating. Please wait...')
        sleep(LEDGER_CREATION_POLL_PERIOD_SEC)


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create a ledger and wait for it to be active.
    """
    try:
        create_ledger(ledger_name)
        wait_for_active(ledger_name)
    except Exception as e:
        logger.exception('Unable to create the ledger!')
        raise e


if __name__ == '__main__':
    main()
```

> **Note**
>
> - `create_ledger` の呼び出しで、台帳名とアクセス許可モードを指定する必要があります。台帳データのセキュリティを最大化する、STANDARD のアクセス許可モードの使用を推奨します。
> - 台帳を作成すると、デフォルトで削除保護が有効になります。これは QLDB の機能であり、ユーザーによって台帳が削除されるのを防ぎます。QLDB API または AWS

Command Line Interface (AWS CLI) を使用して、台帳作成時に削除保護を無効にすることもできます。

- 必要に応じて、台帳にアタッチするタグを指定することもできます。

3. このプログラムを実行するには、次のコマンドを入力します。

```
python create_ledger.py
```

新しい台帳への接続を確認するには、「ステップ 2: 台帳への接続をテストする」に進みます。

ステップ 2: 台帳への接続をテストする

このステップでは、トランザクションデータ API エンドポイントを使用して Amazon QLDB の `vehicle-registration` 台帳に接続できることを確認します。

台帳への接続をテストするには

1. 次のプログラム (`connect_to_ledger.py`) を使用して、`vehicle-registration` 台帳へのデータセッション接続を作成します。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from botocore.exceptions import ClientError

from pyqldb.driver.qldb_driver import QldbDriver
from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_qldb_driver(ledger_name=Constants.LEDGER_NAME, region_name=None,
 endpoint_url=None, boto3_session=None):
    """
    Create a QLDB driver for executing transactions.

    :type ledger_name: str
    :param ledger_name: The QLDB ledger name.

    :type region_name: str
    :param region_name: See [1].

    :type endpoint_url: str
    :param endpoint_url: See [1].

    :type boto3_session: :py:class:`boto3.session.Session`
    :param boto3_session: The boto3 session to create the client with (see [1]).

    :rtype: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :return: A QLDB driver object.

    [1]: `Boto3 Session.client Reference <https://
boto3.amazonaws.com/v1/documentation/api/latest/reference/core/
session.html#boto3.session.Session.client>`.
    """
    qldb_driver = QldbDriver(ledger_name=ledger_name, region_name=region_name,
 endpoint_url=endpoint_url,
```

```
                                boto3_session=boto3_session)
    return qldb_driver


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Connect to a given ledger using default settings.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            logger.info('Listing table names ')
            for table in driver.list_tables():
                logger.info(table)
    except ClientError as ce:
        logger.exception('Unable to list tables.')
        raise ce


if __name__ == '__main__':
    main()
```

> **Note**
>
> - 台帳に対してデータトランザクションを実行するには、QLDB ドライバーオブジェクトを作成して特定の台帳に接続する必要があります。これは、前のステップで台帳の作成に使用した `qldb_client` オブジェクトとは異なるクライアントオブジェクトです。前のクライアントは、「Amazon QLDB API リファレンス」に示されている管理 API オペレーションの実行にのみ使用されます。
> - このドライバーオブジェクトを作成するときに台帳名を指定する必要があります。

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
```

```
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from botocore.exceptions import ClientError

from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_qldb_driver(ledger_name=Constants.LEDGER_NAME, region_name=None,
 endpoint_url=None, boto3_session=None):
    """
    Create a QLDB driver for creating sessions.

    :type ledger_name: str
    :param ledger_name: The QLDB ledger name.

    :type region_name: str
    :param region_name: See [1].

    :type endpoint_url: str
    :param endpoint_url: See [1].

    :type boto3_session: :py:class:`boto3.session.Session`
```

```
    :param boto3_session: The boto3 session to create the client with (see [1]).

    :rtype: :py:class:`pyqldb.driver.pooled_qldb_driver.PooledQldbDriver`
    :return: A pooled QLDB driver object.

    [1]: `Boto3 Session.client Reference <https://
boto3.amazonaws.com/v1/documentation/api/latest/reference/core/
session.html#boto3.session.Session.client>`.
    """
    qldb_driver = PooledQldbDriver(ledger_name=ledger_name,
 region_name=region_name, endpoint_url=endpoint_url,
                                   boto3_session=boto3_session)
    return qldb_driver


def create_qldb_session():
    """
    Retrieve a QLDB session object.

    :rtype: :py:class:`pyqldb.session.pooled_qldb_session.PooledQldbSession`
    :return: A pooled QLDB session object.
    """
    qldb_session = pooled_qldb_driver.get_session()
    return qldb_session


pooled_qldb_driver = create_qldb_driver()


if __name__ == '__main__':
    """
    Connect to a session for a given ledger using default settings.
    """
    try:
        qldb_session = create_qldb_session()
        logger.info('Listing table names ')
        for table in qldb_session.list_tables():
            logger.info(table)
    except ClientError:
        logger.exception('Unable to create session.')
```

> **Note**
>
> - 台帳に対してデータトランザクションを実行するには、QLDB ドライバーオブジェクトを作成して特定の台帳に接続する必要があります。これは、前のステップで台帳の作成に使用した `qldb_client` オブジェクトとは異なるクライアントオブジェクトです。前のクライアントは、「Amazon QLDB API リファレンス」に示されている管理 API オペレーションの実行にのみ使用されます。
> - まず、プールされた QLDB ドライバーオブジェクトを作成します。このドライバーを作成するときに台帳名を指定する必要があります。
> - その後、このプールされたドライバーオブジェクトからセッションを作成できます。

2. このプログラムを実行するには、次のコマンドを入力します。

```
python connect_to_ledger.py
```

`vehicle-registration` 台帳にテーブルを作成するには、「ステップ 3: テーブル、インデックス、およびサンプルデータを作成する」に進みます。

ステップ 3: テーブル、インデックス、およびサンプルデータを作成する

Amazon QLDB 台帳がアクティブになり、接続を受け入れると、車両、車両の所有者、登録情報に関するデータのテーブルを作成できるようになります。テーブルとインデックスを作成した後、これらにデータをロードできます。

このステップでは、`vehicle-registration` 台帳に 4 つのテーブルを作成します。

- `VehicleRegistration`
- `Vehicle`
- `Person`
- `DriversLicense`

以下のインデックスも作成します。

テーブル名	フィールド
VehicleRegistration	VIN
VehicleRegistration	LicensePlateNumber
Vehicle	VIN
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

サンプルデータを挿入するときは、まず Person テーブルにドキュメントを挿入します。次に、各 Person ドキュメントからシステムによって割り当てられた id を使用して、適切な VehicleRegistration および DriversLicense ドキュメントの対応するフィールドに入力します。

> **Tip**
>
> ベストプラクティスとして、外部キーには、ドキュメントにシステムによって割り当てられた id を使用します。一意の識別子 (車両の VIN など) 用にフィールドを定義することはできますが、実際のドキュメントの一意の識別子はその id です。このフィールドはドキュメントのメタデータに含まれており、コミット済みビュー (テーブルのシステム定義のビュー) でクエリを実行できます。
> QLDB におけるビューの詳細については、「重要な概念」を参照してください。メタデータの詳細については、「ドキュメントのメタデータのクエリの実行」を参照してください。

テーブルとインデックスを作成するには

1. 次のプログラム (create_table.py) を使用して前述のテーブルを作成します。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
```

```
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_table(driver, table_name):
    """
    Create a table with the specified name.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to create.

    :rtype: int
    :return: The number of changes to the database.
    """
```

```
    logger.info("Creating the '{}' table...".format(table_name))
    statement = 'CREATE TABLE {}'.format(table_name)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement))
    logger.info('{} table created successfully.'.format(table_name))
    return len(list(cursor))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create registrations, vehicles, owners, and licenses tables.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            create_table(driver, Constants.DRIVERS_LICENSE_TABLE_NAME)
            create_table(driver, Constants.PERSON_TABLE_NAME)
            create_table(driver, Constants.VEHICLE_TABLE_NAME)
            create_table(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME)
            logger.info('Tables created successfully.')
    except Exception as e:
        logger.exception('Errors creating tables.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
```

```
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_table(transaction_executor, table_name):
    """
    Create a table with the specified name using an Executor object.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to create.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Creating the '{}' table...".format(table_name))
    statement = 'CREATE TABLE {}'.format(table_name)
    cursor = transaction_executor.execute_statement(statement)
    logger.info('{} table created successfully.'.format(table_name))
    return len(list(cursor))


if __name__ == '__main__':
    """
```

```
    Create registrations, vehicles, owners, and licenses tables in a single
transaction.
    """
    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda x: create_table(x,
Constants.DRIVERS_LICENSE_TABLE_NAME) and
                                   create_table(x, Constants.PERSON_TABLE_NAME)
and
                                   create_table(x, Constants.VEHICLE_TABLE_NAME)
and
                                   create_table(x,
Constants.VEHICLE_REGISTRATION_TABLE_NAME),
                                   lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Tables created successfully.')
    except Exception:
        logger.exception('Errors creating tables.')
```

> **Note**
>
> このプログラムは、execute_lambda 関数を使用する方法を示しています。この例では、1 つの Lambda 式で複数の CREATE TABLE PartiQL ステートメントを実行します。この execute 関数は、暗黙的にトランザクションを開始し、Lambda ですべてのステートメントを実行して、トランザクションを自動コミットします。

2. このプログラムを実行するには、次のコマンドを入力します。

```
python create_table.py
```

3. 次のプログラム (create_index.py) を使用して前述のテーブルにインデックスを作成します。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_index(driver, table_name, index_attribute):
    """
    Create an index for a particular table.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to add indexes for.

    :type index_attribute: str
    :param index_attribute: Index to create on a single attribute.

    :rtype: int
    :return: The number of changes to the database.
```

```
    """
    logger.info("Creating index on '{}'...".format(index_attribute))
    statement = 'CREATE INDEX on {} ({})'.format(table_name, index_attribute)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement))
    return len(list(cursor))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create indexes on tables in a particular ledger.
    """
    logger.info('Creating indexes on all tables...')
    try:
        with create_qldb_driver(ledger_name) as driver:
            create_index(driver, Constants.PERSON_TABLE_NAME,
 Constants.GOV_ID_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_TABLE_NAME,
 Constants.VEHICLE_VIN_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.LICENSE_PLATE_NUMBER_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.VEHICLE_VIN_INDEX_NAME)
            create_index(driver, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.PERSON_ID_INDEX_NAME)
            create_index(driver, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.LICENSE_NUMBER_INDEX_NAME)
            logger.info('Indexes created successfully.')
    except Exception as e:
        logger.exception('Unable to create indexes.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_index(transaction_executor, table_name, index_attribute):
    """
    Create an index for a particular table.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to add indexes for.

    :type index_attribute: str
    :param index_attribute: Index to create on a single attribute.

    :rtype: int
```

```
    :return: The number of changes to the database.
    """
    logger.info("Creating index on '{}'...".format(index_attribute))
    statement = 'CREATE INDEX on {} ({})'.format(table_name, index_attribute)
    cursor = transaction_executor.execute_statement(statement)
    return len(list(cursor))


if __name__ == '__main__':
    """
    Create indexes on tables in a particular ledger.
    """
    logger.info('Creating indexes on all tables in a single transaction...')
    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda x: create_index(x,
 Constants.PERSON_TABLE_NAME,

 Constants.GOV_ID_INDEX_NAME)
                                   and create_index(x,
 Constants.VEHICLE_TABLE_NAME,

 Constants.VEHICLE_VIN_INDEX_NAME)
                                   and create_index(x,
 Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.LICENSE_PLATE_NUMBER_INDEX_NAME)
                                   and create_index(x,
 Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.VEHICLE_VIN_INDEX_NAME)
                                   and create_index(x,
 Constants.DRIVERS_LICENSE_TABLE_NAME,

 Constants.PERSON_ID_INDEX_NAME)
                                   and create_index(x,
 Constants.DRIVERS_LICENSE_TABLE_NAME,

 Constants.LICENSE_NUMBER_INDEX_NAME),
                                   lambda retry_attempt: logger.info('Retrying
 due to OCC conflict...'))
            logger.info('Indexes created successfully.')
    except Exception:
```

```
        logger.exception('Unable to create indexes.')
```

4. このプログラムを実行するには、次のコマンドを入力します。

```
python create_index.py
```

データをテーブルにロードするには

1. 次のファイル (sample_data.py) を確認します。このファイルには、vehicle-registration テーブルに挿入するサンプルデータが含まれています。このファイルは、Amazon Ion データを変換、解析、および出力するヘルパー関数を提供する amazon.ion パッケージからもインポートされます。

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
from datetime import datetime
from decimal import Decimal
from logging import basicConfig, getLogger, INFO

from amazon.ion.simple_types import IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict,
 IonPyFloat, IonPyInt, IonPyList, \
    IonPyNull, IonPySymbol, IonPyText, IonPyTimestamp
```

```
from amazon.ion.simpleion import dumps, loads

logger = getLogger(__name__)
basicConfig(level=INFO)
IonValue = (IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict, IonPyFloat, IonPyInt,
 IonPyList, IonPyNull, IonPySymbol,
            IonPyText, IonPyTimestamp)


class SampleData:
    """
    Sample domain objects for use throughout this tutorial.
    """
    DRIVERS_LICENSE = [
        {
            'PersonId': '',
            'LicenseNumber': 'LEWISR261LL',
            'LicenseType': 'Learner',
            'ValidFromDate': datetime(2016, 12, 20),
            'ValidToDate': datetime(2020, 11, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': 'LOGANB486CG',
            'LicenseType': 'Probationary',
            'ValidFromDate': datetime(2016, 4, 6),
            'ValidToDate': datetime(2020, 11, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': '744 849 301',
            'LicenseType': 'Full',
            'ValidFromDate': datetime(2017, 12, 6),
            'ValidToDate': datetime(2022, 10, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': 'P626-168-229-765',
            'LicenseType': 'Learner',
            'ValidFromDate': datetime(2017, 8, 16),
            'ValidToDate': datetime(2021, 11, 15)
        },
        {
            'PersonId': '',
```

```
            'LicenseNumber': 'S152-780-97-415-0',
            'LicenseType': 'Probationary',
            'ValidFromDate': datetime(2015, 8, 15),
            'ValidToDate': datetime(2021, 8, 21)
        }
    ]
    PERSON = [
        {
            'FirstName': 'Raul',
            'LastName': 'Lewis',
            'Address': '1719 University Street, Seattle, WA, 98109',
            'DOB': datetime(1963, 8, 19),
            'GovId': 'LEWISR261LL',
            'GovIdType': 'Driver License'
        },
        {
            'FirstName': 'Brent',
            'LastName': 'Logan',
            'DOB': datetime(1967, 7, 3),
            'Address': '43 Stockert Hollow Road, Everett, WA, 98203',
            'GovId': 'LOGANB486CG',
            'GovIdType': 'Driver License'
        },
        {
            'FirstName': 'Alexis',
            'LastName': 'Pena',
            'DOB': datetime(1974, 2, 10),
            'Address': '4058 Melrose Street, Spokane Valley, WA, 99206',
            'GovId': '744 849 301',
            'GovIdType': 'SSN'
        },
        {
            'FirstName': 'Melvin',
            'LastName': 'Parker',
            'DOB': datetime(1976, 5, 22),
            'Address': '4362 Ryder Avenue, Seattle, WA, 98101',
            'GovId': 'P626-168-229-765',
            'GovIdType': 'Passport'
        },
        {
            'FirstName': 'Salvatore',
            'LastName': 'Spencer',
            'DOB': datetime(1997, 11, 15),
            'Address': '4450 Honeysuckle Lane, Seattle, WA, 98101',
```

```
            'GovId': 'S152-780-97-415-0',
            'GovIdType': 'Passport'
        }
    ]
    VEHICLE = [
        {
            'VIN': '1N4AL11D75C109151',
            'Type': 'Sedan',
            'Year': 2011,
            'Make': 'Audi',
            'Model': 'A5',
            'Color': 'Silver'
        },
        {
            'VIN': 'KM8SRDHF6EU074761',
            'Type': 'Sedan',
            'Year': 2015,
            'Make': 'Tesla',
            'Model': 'Model S',
            'Color': 'Blue'
        },
        {
            'VIN': '3HGGK5G53FM761765',
            'Type': 'Motorcycle',
            'Year': 2011,
            'Make': 'Ducati',
            'Model': 'Monster 1200',
            'Color': 'Yellow'
        },
        {
            'VIN': '1HVBBAANXWH544237',
            'Type': 'Semi',
            'Year': 2009,
            'Make': 'Ford',
            'Model': 'F 150',
            'Color': 'Black'
        },
        {
            'VIN': '1C4RJFAG0FC625797',
            'Type': 'Sedan',
            'Year': 2019,
            'Make': 'Mercedes',
            'Model': 'CLK 350',
            'Color': 'White'
```

```
        }
    ]
    VEHICLE_REGISTRATION = [
        {
            'VIN': '1N4AL11D75C109151',
            'LicensePlateNumber': 'LEWISR261LL',
            'State': 'WA',
            'City': 'Seattle',
            'ValidFromDate': datetime(2017, 8, 21),
            'ValidToDate': datetime(2020, 5, 11),
            'PendingPenaltyTicketAmount': Decimal('90.25'),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        },
        {
            'VIN': 'KM8SRDHF6EU074761',
            'LicensePlateNumber': 'CA762X',
            'State': 'WA',
            'City': 'Kent',
            'PendingPenaltyTicketAmount': Decimal('130.75'),
            'ValidFromDate': datetime(2017, 9, 14),
            'ValidToDate': datetime(2020, 6, 25),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        },
        {
            'VIN': '3HGGK5G53FM761765',
            'LicensePlateNumber': 'CD820Z',
            'State': 'WA',
            'City': 'Everett',
            'PendingPenaltyTicketAmount': Decimal('442.30'),
            'ValidFromDate': datetime(2011, 3, 17),
            'ValidToDate': datetime(2021, 3, 24),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        },
        {
            'VIN': '1HVBBAANXWH544237',
```

```
            'LicensePlateNumber': 'LS477D',
            'State': 'WA',
            'City': 'Tacoma',
            'PendingPenaltyTicketAmount': Decimal('42.20'),
            'ValidFromDate': datetime(2011, 10, 26),
            'ValidToDate': datetime(2023, 9, 25),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        },
        {
            'VIN': '1C4RJFAG0FC625797',
            'LicensePlateNumber': 'TH393F',
            'State': 'WA',
            'City': 'Olympia',
            'PendingPenaltyTicketAmount': Decimal('30.45'),
            'ValidFromDate': datetime(2013, 9, 2),
            'ValidToDate': datetime(2024, 3, 19),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        }
    ]


def convert_object_to_ion(py_object):
    """
    Convert a Python object into an Ion object.

    :type py_object: object
    :param py_object: The object to convert.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyValue`
    :return: The converted Ion object.
    """
    ion_object = loads(dumps(py_object))
    return ion_object


def to_ion_struct(key, value):
    """
    Convert the given key and value into an Ion struct.
```

```
    :type key: str
    :param key: The key which serves as an unique identifier.

    :type value: str
    :param value: The value associated with a given key.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The Ion dictionary object.
    """
    ion_struct = dict()
    ion_struct[key] = value
    return loads(str(ion_struct))


def get_document_ids(transaction_executor, table_name, field, value):
    """
    Gets the document IDs from the given table.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: The table name to query.

    :type field: str
    :param field: A field to query.

    :type value: str
    :param value: The key of the given field.

    :rtype: list
    :return: A list of document IDs.
    """
    query = "SELECT id FROM {} AS t BY id WHERE t.{} = ?".format(table_name, field)
    cursor = transaction_executor.execute_statement(query,
 convert_object_to_ion(value))
    return list(map(lambda table: table.get('id'), cursor))


def get_document_ids_from_dml_results(result):
    """
    Return a list of modified document IDs as strings from DML results.
```

```
    :type result: :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :param result: The result set from DML operation.

    :rtype: list
    :return: List of document IDs.
    """
    ret_val = list(map(lambda x: x.get('documentId'), result))
    return ret_val


def print_result(cursor):
    """
    Pretty print the result set. Returns the number of documents in the result set.

    :type cursor: :py:class:`pyqldb.cursor.stream_cursor.StreamCursor`/
                  :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :param cursor: An instance of the StreamCursor or BufferedCursor class.

    :rtype: int
    :return: Number of documents in the result set.
    """
    result_counter = 0
    for row in cursor:
        # Each row would be in Ion format.
        print_ion(row)
        result_counter += 1
    return result_counter


def print_ion(ion_value):
    """
    Pretty print an Ion Value.

    :type ion_value: :py:class:`amazon.ion.simple_types.IonPySymbol`
    :param ion_value: Any Ion Value to be pretty printed.
    """
    logger.info(dumps(ion_value, binary=False, indent='  ',
 omit_version_marker=True))
```

> **Note**
>
> get_document_ids 関数は、システムによって割り当てられたドキュメント ID をテーブルから返すクエリを実行します。詳細については、「BY 句を使用したドキュメント ID のクエリの実行」を参照してください。

2. 次のプログラム (insert_document.py) を使用してサンプルデータをテーブルに挿入します。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion, SampleData,
 get_document_ids_from_dml_results
```

```
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def update_person_id(document_ids):
    """
    Update the PersonId value for DriversLicense records and the PrimaryOwner
 value for VehicleRegistration records.

    :type document_ids: list
    :param document_ids: List of document IDs.

    :rtype: list
    :return: Lists of updated DriversLicense records and updated
 VehicleRegistration records.
    """
    new_drivers_licenses = SampleData.DRIVERS_LICENSE.copy()
    new_vehicle_registrations = SampleData.VEHICLE_REGISTRATION.copy()
    for i in range(len(SampleData.PERSON)):
        drivers_license = new_drivers_licenses[i]
        registration = new_vehicle_registrations[i]
        drivers_license.update({'PersonId': str(document_ids[i])})
        registration['Owners']['PrimaryOwner'].update({'PersonId':
 str(document_ids[i])})
    return new_drivers_licenses, new_vehicle_registrations


def insert_documents(driver, table_name, documents):
    """
    Insert the given list of documents into a table in a single transaction.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to insert documents into.

    :type documents: list
    :param documents: List of documents to insert.

    :rtype: list
    :return: List of documents IDs for the newly inserted documents.
```

```
    """
    logger.info('Inserting some documents in the {}
 table...'.format(table_name))
    statement = 'INSERT INTO {} ?'.format(table_name)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement,

 convert_object_to_ion(documents)))
    list_of_document_ids = get_document_ids_from_dml_results(cursor)

    return list_of_document_ids


def update_and_insert_documents(driver):
    """
    Handle the insertion of documents and updating PersonIds.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.
    """
    list_ids = insert_documents(driver, Constants.PERSON_TABLE_NAME,
 SampleData.PERSON)

    logger.info("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
 'VehicleRegistration'...")
    new_licenses, new_registrations = update_person_id(list_ids)

    insert_documents(driver, Constants.VEHICLE_TABLE_NAME, SampleData.VEHICLE)
    insert_documents(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 new_registrations)
    insert_documents(driver, Constants.DRIVERS_LICENSE_TABLE_NAME, new_licenses)


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Insert documents into a table in a QLDB ledger.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            # An INSERT statement creates the initial revision of a document
 with a version number of zero.
            # QLDB also assigns a unique document identifier in GUID format as
 part of the metadata.
            update_and_insert_documents(driver)
```

```
            logger.info('Documents inserted successfully!')
    except Exception as e:
        logger.exception('Error inserting or updating documents.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion, SampleData,
 get_document_ids_from_dml_results
from pyqldbsamples.connect_to_ledger import create_qldb_session
```

```
logger = getLogger(__name__)
basicConfig(level=INFO)


def update_person_id(document_ids):
    """
    Update the PersonId value for DriversLicense records and the PrimaryOwner
 value for VehicleRegistration records.

    :type document_ids: list
    :param document_ids: List of document IDs.

    :rtype: list
    :return: Lists of updated DriversLicense records and updated
 VehicleRegistration records.
    """
    new_drivers_licenses = SampleData.DRIVERS_LICENSE.copy()
    new_vehicle_registrations = SampleData.VEHICLE_REGISTRATION.copy()
    for i in range(len(SampleData.PERSON)):
        drivers_license = new_drivers_licenses[i]
        registration = new_vehicle_registrations[i]
        drivers_license.update({'PersonId': str(document_ids[i])})
        registration['Owners']['PrimaryOwner'].update({'PersonId':
 str(document_ids[i])})
    return new_drivers_licenses, new_vehicle_registrations


def insert_documents(transaction_executor, table_name, documents):
    """
    Insert the given list of documents into a table in a single transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to insert documents into.

    :type documents: list
    :param documents: List of documents to insert.

    :rtype: list
    :return: List of documents IDs for the newly inserted documents.
    """
```

```
    logger.info('Inserting some documents in the {}
 table...'.format(table_name))
    statement = 'INSERT INTO {} ?'.format(table_name)
    cursor = transaction_executor.execute_statement(statement,
 convert_object_to_ion(documents))
    list_of_document_ids = get_document_ids_from_dml_results(cursor)

    return list_of_document_ids


def update_and_insert_documents(transaction_executor):
    """
    Handle the insertion of documents and updating PersonIds all in a single
 transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
    """
    list_ids = insert_documents(transaction_executor,
 Constants.PERSON_TABLE_NAME, SampleData.PERSON)

    logger.info("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
 'VehicleRegistration'...")
    new_licenses, new_registrations = update_person_id(list_ids)

    insert_documents(transaction_executor, Constants.VEHICLE_TABLE_NAME,
 SampleData.VEHICLE)
    insert_documents(transaction_executor,
 Constants.VEHICLE_REGISTRATION_TABLE_NAME, new_registrations)
    insert_documents(transaction_executor, Constants.DRIVERS_LICENSE_TABLE_NAME,
 new_licenses)


if __name__ == '__main__':
    """
    Insert documents into a table in a QLDB ledger.
    """
    try:
        with create_qldb_session() as session:
            # An INSERT statement creates the initial revision of a document
 with a version number of zero.
            # QLDB also assigns a unique document identifier in GUID format as
 part of the metadata.
```

```
            session.execute_lambda(lambda executor:
update_and_insert_documents(executor),
                                   lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Documents inserted successfully!')
    except Exception:
        logger.exception('Error inserting or updating documents.')
```

> **Note**
>
> - このプログラムは、パラメータ化された値を渡して execute_statement 関数を呼び出す方法を示しています。実行する PartiQL ステートメントに加えて、データパラメータを渡すことができます。ステートメント文字列の変数プレースホルダーとして疑問符 (?) を使用します。
> - INSERT ステートメントが成功すると、 挿入された各ドキュメントの id が返されます。

3. このプログラムを実行するには、次のコマンドを入力します。

```
python insert_document.py
```

次に、SELECT ステートメントを使用すると、vehicle-registration 台帳のテーブルからデータを読み取ることができます。ステップ 4: 台帳のテーブルにクエリを実行する に進みます。

ステップ 4: 台帳のテーブルにクエリを実行する

Amazon QLDB 台帳にテーブルを作成し、データをロードした後は、クエリを実行して、挿入した車両登録データを確認できます。QLDB は PartiQL をクエリ言語として使用し、Amazon Ion をドキュメント指向のデータモデルとして使用します。

PartiQL は、Ion で動作するように拡張されたオープンソースの SQL 互換のクエリ言語です。PartiQL を使用すると、使い慣れた SQL 演算子を使用してデータを挿入、クエリ、および管理できます。Amazon Ion は JSON のスーパーセットです。Ion はオープンソースのドキュメントベースのデータ形式であり、構造化データ、半構造化データ、およびネストされたデータを柔軟に保存および処理できます。

このステップでは、SELECT ステートメントを使用して、vehicle-registration 台帳のテーブルからデータを読み取ります。

> **Warning**
>
> インデックス付きルックアップなしで QLDB でクエリを実行すると、完全なテーブルスキャンが呼び出されます。PartiQL は SQL 互換であるため、このようなクエリをサポートしています。ただし、QLDB の本番環境のユースケースではテーブルスキャンを実行しないでください。テーブルスキャンより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する必要があります (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`)。詳細については、「クエリパフォーマンスの最適化」を参照してください。

テーブルのクエリを実行するには

1. 次のプログラム (find_vehicles.py) を使用して台帳の人物に登録されているすべての車両をクエリします。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def find_vehicles_for_owner(driver, gov_id):
    """
    Find vehicles registered under a driver using their government ID.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type gov_id: str
    :param gov_id: The owner's government ID.
    """
    document_ids = driver.execute_lambda(lambda executor:
 get_document_ids(executor, Constants.PERSON_TABLE_NAME,

 'GovId', gov_id))

    query = "SELECT Vehicle FROM Vehicle INNER JOIN VehicleRegistration AS r " \
            "ON Vehicle.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId = ?"

    for ids in document_ids:
        cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, ids))
        logger.info('List of Vehicles for owner with GovId:
 {}...'.format(gov_id))
        print_result(cursor)
```

```
def main(ledger_name=Constants.LEDGER_NAME):
    """
    Find all vehicles registered under a person.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            # Find all vehicles registered under a person.
            gov_id = SampleData.PERSON[0]['GovId']
            find_vehicles_for_owner(driver, gov_id)
    except Exception as e:
        logger.exception('Error getting vehicles for owner.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
```

```
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def find_vehicles_for_owner(transaction_executor, gov_id):
    """
    Find vehicles registered under a driver using their government ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type gov_id: str
    :param gov_id: The owner's government ID.
    """
    document_ids = get_document_ids(transaction_executor,
 Constants.PERSON_TABLE_NAME, 'GovId', gov_id)

    query = "SELECT Vehicle FROM Vehicle INNER JOIN VehicleRegistration AS r " \
            "ON Vehicle.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId = ?"

    for ids in document_ids:
        cursor = transaction_executor.execute_statement(query, ids)
        logger.info('List of Vehicles for owner with GovId:
 {}...'.format(gov_id))
        print_result(cursor)


if __name__ == '__main__':
    """
    Find all vehicles registered under a person.
    """
    try:
        with create_qldb_session() as session:
            # Find all vehicles registered under a person.
            gov_id = SampleData.PERSON[0]['GovId']
```

```
            session.execute_lambda(lambda executor:
find_vehicles_for_owner(executor, gov_id),
                                   lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
   except Exception:
       logger.exception('Error getting vehicles for owner.')
```

> **Note**
>
> まず、このプログラムでは、GovId LEWISR261LL を使用してドキュメントの Person テーブルに対してクエリを実行し、id メタデータフィールドを取得します。
> 次に、このドキュメント id を外部キーとして使用して、PrimaryOwner.PersonId によって VehicleRegistration テーブルをクエリします。また、VehicleRegistration が VIN フィールドの Vehicle テーブルと結合されます。

2. このプログラムを実行するには、次のコマンドを入力します。

```
python find_vehicles.py
```

vehicle-registration 台帳のテーブルのドキュメントの変更方法については、「ステップ 5: 台帳内のドキュメントを変更する」を参照してください。

ステップ 5: 台帳内のドキュメントを変更する

操作するデータを用意できたところで、Amazon QLDB で vehicle-registration 台帳のドキュメントに変更を加えることができます。このステップでは、次のコード例により、データ操作言語 (DML) ステートメントを実行する方法を示します。これらのステートメントは、ある車両の主所有者を更新し、別の車両に第二所有者を追加します。

ドキュメントに変更を加えるには

1. 次のプログラム (transfer_vehicle_ownership.py) を使用して、台帳で VIN が 1N4AL11D75C109151 の車両の主所有者を更新します。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.add_secondary_owner import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def find_person_from_document_id(transaction_executor, document_id):
    """
    Query a driver's information using the given ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
```

```
    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: The document ID required to query for the person.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    query = 'SELECT p.* FROM Person AS p BY pid WHERE pid = ?'
    cursor = transaction_executor.execute_statement(query, document_id)
    return next(cursor)


def find_primary_owner_for_vehicle(driver, vin):
    """
    Find the primary owner of a vehicle given its VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN to find primary owner for.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    logger.info('Finding primary owner for vehicle with VIN: {}.'.format(vin))
    query = "SELECT Owners.PrimaryOwner.PersonId FROM VehicleRegistration AS v
 WHERE v.VIN = ?"
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, convert_object_to_ion(vin)))
    try:
        return driver.execute_lambda(lambda executor:
 find_person_from_document_id(executor,

   next(cursor).get('PersonId')))
    except StopIteration:
        logger.error('No primary owner registered for this vehicle.')
        return None


def update_vehicle_registration(driver, vin, document_id):
    """
    Update the primary owner for a vehicle using the given VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
```

```
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN for the vehicle to operate on.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: New PersonId for the primary owner.

    :raises RuntimeError: If no vehicle registration was found using the given
 document ID and VIN.
    """
    logger.info('Updating the primary owner for vehicle with Vin:
 {}...'.format(vin))
    statement = "UPDATE VehicleRegistration AS r SET
 r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?"
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement, document_id,

 convert_object_to_ion(vin)))
    try:
        print_result(cursor)
        logger.info('Successfully transferred vehicle with VIN: {} to new
 owner.'.format(vin))
    except StopIteration:
        raise RuntimeError('Unable to transfer vehicle, could not find
 registration.')


def validate_and_update_registration(driver, vin, current_owner, new_owner):
    """
    Validate the current owner of the given vehicle and transfer its ownership
 to a new owner.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN of the vehicle to transfer ownership of.

    :type current_owner: str
    :param current_owner: The GovId of the current owner of the vehicle.

    :type new_owner: str
    :param new_owner: The GovId of the new owner of the vehicle.
```

```
    :raises RuntimeError: If unable to verify primary owner.
    """
    primary_owner = find_primary_owner_for_vehicle(driver, vin)
    if primary_owner is None or primary_owner['GovId'] != current_owner:
        raise RuntimeError('Incorrect primary owner identified for vehicle,
 unable to transfer.')

    document_ids = driver.execute_lambda(lambda executor:
 get_document_ids(executor, Constants.PERSON_TABLE_NAME,

 'GovId', new_owner))
    update_vehicle_registration(driver, vin, document_ids[0])


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Find primary owner for a particular vehicle's VIN.
    Transfer to another primary owner for a particular vehicle's VIN.
    """
    vehicle_vin = SampleData.VEHICLE[0]['VIN']
    previous_owner = SampleData.PERSON[0]['GovId']
    new_owner = SampleData.PERSON[1]['GovId']

    try:
        with create_qldb_driver(ledger_name) as driver:
            validate_and_update_registration(driver, vehicle_vin,
 previous_owner, new_owner)
    except Exception as e:
        logger.exception('Error updating VehicleRegistration.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.add_secondary_owner import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def find_person_from_document_id(transaction_executor, document_id):
    """
    Query a driver's information using the given ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: The document ID required to query for the person.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
```

```
    :return: The resulting document from the query.
    """
    query = 'SELECT p.* FROM Person AS p BY pid WHERE pid = ?'
    cursor = transaction_executor.execute_statement(query, document_id)
    return next(cursor)


def find_primary_owner_for_vehicle(transaction_executor, vin):
    """
    Find the primary owner of a vehicle given its VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: The VIN to find primary owner for.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    logger.info('Finding primary owner for vehicle with VIN: {}.'.format(vin))
    query = "SELECT Owners.PrimaryOwner.PersonId FROM VehicleRegistration AS v
 WHERE v.VIN = ?"
    cursor = transaction_executor.execute_statement(query,
 convert_object_to_ion(vin))
    try:
        return find_person_from_document_id(transaction_executor,
 next(cursor).get('PersonId'))
    except StopIteration:
        logger.error('No primary owner registered for this vehicle.')
        return None


def update_vehicle_registration(transaction_executor, vin, document_id):
    """
    Update the primary owner for a vehicle using the given VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: The VIN for the vehicle to operate on.
```

```
    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: New PersonId for the primary owner.

    :raises RuntimeError: If no vehicle registration was found using the given
 document ID and VIN.
    """
    logger.info('Updating the primary owner for vehicle with Vin:
 {}...'.format(vin))
    statement = "UPDATE VehicleRegistration AS r SET
 r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?"
    cursor = transaction_executor.execute_statement(statement, document_id,
 convert_object_to_ion(vin))
    try:
        print_result(cursor)
        logger.info('Successfully transferred vehicle with VIN: {} to new
 owner.'.format(vin))
    except StopIteration:
        raise RuntimeError('Unable to transfer vehicle, could not find
 registration.')


def validate_and_update_registration(transaction_executor, vin, current_owner,
 new_owner):
    """
    Validate the current owner of the given vehicle and transfer its ownership
 to a new owner in a single transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: The VIN of the vehicle to transfer ownership of.

    :type current_owner: str
    :param current_owner: The GovId of the current owner of the vehicle.

    :type new_owner: str
    :param new_owner: The GovId of the new owner of the vehicle.

    :raises RuntimeError: If unable to verify primary owner.
    """
    primary_owner = find_primary_owner_for_vehicle(transaction_executor, vin)
```

```
    if primary_owner is None or primary_owner['GovId'] != current_owner:
        raise RuntimeError('Incorrect primary owner identified for vehicle,
 unable to transfer.')

    document_id = next(get_document_ids(transaction_executor,
 Constants.PERSON_TABLE_NAME, 'GovId', new_owner))

    update_vehicle_registration(transaction_executor, vin, document_id)


if __name__ == '__main__':
    """
    Find primary owner for a particular vehicle's VIN.
    Transfer to another primary owner for a particular vehicle's VIN.
    """
    vehicle_vin = SampleData.VEHICLE[0]['VIN']
    previous_owner = SampleData.PERSON[0]['GovId']
    new_owner = SampleData.PERSON[1]['GovId']

    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda executor:
 validate_and_update_registration(executor, vehicle_vin,

     previous_owner, new_owner),
                                   retry_indicator=lambda retry_attempt:
 logger.info('Retrying due to OCC conflict...'))
    except Exception:
        logger.exception('Error updating VehicleRegistration.')
```

2. このプログラムを実行するには、次のコマンドを入力します。

```
python transfer_vehicle_ownership.py
```

3. 次のプログラム (add_secondary_owner.py) を使用して、台帳で VIN が KM8SRDHF6EU074761 の車両に第二所有者を追加します。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
```

```
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import to_ion_struct, get_document_ids,
 print_result, SampleData, \
    convert_object_to_ion
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def get_document_id_by_gov_id(driver, government_id):
    """
    Find a driver's person ID using the given government ID.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type government_id: str
    :param government_id: A driver's government ID.
```

```
    :rtype: list
    :return: A list of document IDs.
    """
    logger.info("Finding secondary owner's person ID using given government ID:
 {}.".format(government_id))
    return driver.execute_lambda(lambda executor: get_document_ids(executor,
 Constants.PERSON_TABLE_NAME, 'GovId',

 government_id))


def is_secondary_owner_for_vehicle(driver, vin, secondary_owner_id):
    """
    Check whether a secondary owner has already been registered for the given
 VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to query.

    :type secondary_owner_id: str
    :param secondary_owner_id: The secondary owner's person ID.

    :rtype: bool
    :return: If the driver has already been registered.
    """
    logger.info('Finding secondary owners for vehicle with VIN:
 {}...'.format(vin))
    query = 'SELECT Owners.SecondaryOwners FROM VehicleRegistration AS v WHERE
 v.VIN = ?'
    rows = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, convert_object_to_ion(vin)))

    for row in rows:
        secondary_owners = row.get('SecondaryOwners')
        person_ids = map(lambda owner: owner.get('PersonId').text,
 secondary_owners)
        if secondary_owner_id in person_ids:
            return True
    return False
```

```
def add_secondary_owner_for_vin(driver, vin, parameter):
    """
    Add a secondary owner into `VehicleRegistration` table for a particular VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to add a secondary owner for.

    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to
 Ion for filling in parameters of the
                      statement.
    """
    logger.info('Inserting secondary owner for vehicle with VIN:
 {}...'.format(vin))
    statement = "FROM VehicleRegistration AS v WHERE v.VIN = ? INSERT INTO
 v.Owners.SecondaryOwners VALUE ?"

    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement, convert_object_to_ion(vin),

 parameter))
    logger.info('VehicleRegistration Document IDs which had secondary owners
 added: ')
    print_result(cursor)


def register_secondary_owner(driver, vin, gov_id):
    """
    Register a secondary owner for a vehicle if they are not already registered.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to register a secondary owner for.

    :type gov_id: str
    :param gov_id: The government ID of the owner.
    """
    logger.info('Finding the secondary owners for vehicle with VIN:
 {}.'.format(vin))
```

```
    document_ids = get_document_id_by_gov_id(driver, gov_id)

    for document_id in document_ids:
        if is_secondary_owner_for_vehicle(driver, vin, document_id):
            logger.info('Person with ID {} has already been added as a secondary
 owner of this vehicle.'.format(gov_id))
        else:
            add_secondary_owner_for_vin(driver, vin, to_ion_struct('PersonId',
 document_id))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Finds and adds secondary owners for a vehicle.
    """
    vin = SampleData.VEHICLE[1]['VIN']
    gov_id = SampleData.PERSON[0]['GovId']
    try:
        with create_qldb_driver(ledger_name) as driver:
            register_secondary_owner(driver, vin, gov_id)
            logger.info('Secondary owners successfully updated.')
    except Exception as e:
        logger.exception('Error adding secondary owner.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
```

```
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import to_ion_struct, get_document_ids,
 print_result, SampleData, \
    convert_object_to_ion
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def get_document_id_by_gov_id(transaction_executor, government_id):
    """
    Find a driver's person ID using the given government ID.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
    :type government_id: str
    :param government_id: A driver's government ID.
    :rtype: list
    :return: A list of document IDs.
    """
    logger.info("Finding secondary owner's person ID using given government ID:
 {}.".format(government_id))
    return get_document_ids(transaction_executor, Constants.PERSON_TABLE_NAME,
 'GovId', government_id)
```

```
def is_secondary_owner_for_vehicle(transaction_executor, vin,
 secondary_owner_id):
    """
    Check whether a secondary owner has already been registered for the given
 VIN.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to query.
    :type secondary_owner_id: str
    :param secondary_owner_id: The secondary owner's person ID.
    :rtype: bool
    :return: If the driver has already been registered.
    """
    logger.info('Finding secondary owners for vehicle with VIN:
 {}...'.format(vin))
    query = 'SELECT Owners.SecondaryOwners FROM VehicleRegistration AS v WHERE
 v.VIN = ?'
    rows = transaction_executor.execute_statement(query,
 convert_object_to_ion(vin))

    for row in rows:
        secondary_owners = row.get('SecondaryOwners')
        person_ids = map(lambda owner: owner.get('PersonId').text,
 secondary_owners)
        if secondary_owner_id in person_ids:
            return True
    return False


def add_secondary_owner_for_vin(transaction_executor, vin, parameter):
    """
    Add a secondary owner into `VehicleRegistration` table for a particular VIN.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to add a secondary owner for.
    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to
 Ion for filling in parameters of the
                      statement.
    """
```

```
    logger.info('Inserting secondary owner for vehicle with VIN:
 {}...'.format(vin))
    statement = "FROM VehicleRegistration AS v WHERE v.VIN = '{}' INSERT INTO
 v.Owners.SecondaryOwners VALUE ?"\
        .format(vin)

    cursor = transaction_executor.execute_statement(statement, parameter)
    logger.info('VehicleRegistration Document IDs which had secondary owners
 added: ')
    print_result(cursor)


def register_secondary_owner(transaction_executor, vin, gov_id):
    """
    Register a secondary owner for a vehicle if they are not already registered.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to register a secondary owner for.
    :type gov_id: str
    :param gov_id: The government ID of the owner.
    """
    logger.info('Finding the secondary owners for vehicle with VIN:
 {}.'.format(vin))
    document_ids = get_document_id_by_gov_id(transaction_executor, gov_id)

    for document_id in document_ids:
        if is_secondary_owner_for_vehicle(transaction_executor, vin,
 document_id):
            logger.info('Person with ID {} has already been added as a secondary
 owner of this vehicle.'.format(gov_id))
        else:
            add_secondary_owner_for_vin(transaction_executor, vin,
 to_ion_struct('PersonId', document_id))


if __name__ == '__main__':
    """
    Finds and adds secondary owners for a vehicle.
    """
    vin = SampleData.VEHICLE[1]['VIN']
    gov_id = SampleData.PERSON[0]['GovId']
    try:
```

```
        with create_qldb_session() as session:
            session.execute_lambda(lambda executor:
register_secondary_owner(executor, vin, gov_id),
                                   lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Secondary owners successfully updated.')
    except Exception:
        logger.exception('Error adding secondary owner.')
```

4. このプログラムを実行するには、次のコマンドを入力します。

```
python add_secondary_owner.py
```

これらの変更を vehicle-registration 台帳で確認するには、「ステップ 6: ドキュメントのリビジョン履歴を表示する」を参照してください。

ステップ 6: ドキュメントのリビジョン履歴を表示する

前のステップで車両の登録データを変更した後、登録されているすべての所有者とその他の更新されたフィールドの履歴に対してクエリを実行できます。このステップでは、vehicle-registration 台帳の VehicleRegistration テーブルに含まれているドキュメントのリビジョン履歴のクエリを実行します。

リビジョン履歴を表示するには

1. 次のプログラム (query_history.py) を使用して、VIN が 1N4AL11D75C109151 の VehicleRegistration ドキュメントのリビジョン履歴をクエリします。

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
```

```
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime, timedelta
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import print_result, get_document_ids,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def format_date_time(date_time):
    """
    Format the given date time to a string.

    :type date_time: :py:class:`datetime.datetime`
    :param date_time: The date time to format.

    :rtype: str
    :return: The formatted date time.
    """
    return date_time.strftime('`%Y-%m-%dT%H:%M:%S.%fZ`')


def previous_primary_owners(driver, vin):
    """
    Find previous primary owners for the given VIN in a single transaction.
    In this example, query the `VehicleRegistration` history table to find all
 previous primary owners for a VIN.
```

```
    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN to find previous primary owners for.
    """
    person_ids = driver.execute_lambda(lambda executor:
 get_document_ids(executor,

 Constants.VEHICLE_REGISTRATION_TABLE_NAME,
                                                                    'VIN',
 vin))

    todays_date = datetime.utcnow() - timedelta(seconds=1)
    three_months_ago = todays_date - timedelta(days=90)
    query = 'SELECT data.Owners.PrimaryOwner, metadata.version FROM history({},
 {}, {}) AS h WHERE h.metadata.id = ?'.\
        format(Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 format_date_time(three_months_ago),
               format_date_time(todays_date))

    for ids in person_ids:
        logger.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}.".format(vin))
        cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, ids))
        if not (print_result(cursor)) > 0:
            logger.info('No modification history found within the given time
 frame for document ID: {}'.format(ids))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Query a table's history for a particular set of documents.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            vin = SampleData.VEHICLE_REGISTRATION[0]['VIN']
            previous_primary_owners(driver, vin)
            logger.info('Successfully queried history.')
    except Exception as e:
        logger.exception('Unable to query history to find previous owners.')
        raise e
```

```
if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime, timedelta
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import print_result, get_document_ids,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)
```

```
def format_date_time(date_time):
    """
    Format the given date time to a string.

    :type date_time: :py:class:`datetime.datetime`
    :param date_time: The date time to format.

    :rtype: str
    :return: The formatted date time.
    """
    return date_time.strftime('`%Y-%m-%dT%H:%M:%S.%fZ`')


def previous_primary_owners(transaction_executor, vin):
    """
    Find previous primary owners for the given VIN in a single transaction.
    In this example, query the `VehicleRegistration` history table to find all
 previous primary owners for a VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: VIN to find previous primary owners for.
    """
    person_ids = get_document_ids(transaction_executor,
 Constants.VEHICLE_REGISTRATION_TABLE_NAME, 'VIN', vin)

    todays_date = datetime.utcnow() - timedelta(seconds=1)
    three_months_ago = todays_date - timedelta(days=90)
    query = 'SELECT data.Owners.PrimaryOwner, metadata.version FROM history({},
 {}, {}) AS h WHERE h.metadata.id = ?'.\
        format(Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 format_date_time(three_months_ago),
               format_date_time(todays_date))

    for ids in person_ids:
        logger.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}.".format(vin))
        cursor = transaction_executor.execute_statement(query, ids)
        if not (print_result(cursor)) > 0:
```

```
            logger.info('No modification history found within the given time
 frame for document ID: {}'.format(ids))


if __name__ == '__main__':
    """
    Query a table's history for a particular set of documents.
    """
    try:
        with create_qldb_session() as session:
            vin = SampleData.VEHICLE_REGISTRATION[0]['VIN']
            session.execute_lambda(lambda lambda_executor:
 previous_primary_owners(lambda_executor, vin),
                                   lambda retry_attempt: logger.info('Retrying
 due to OCC conflict...'))
            logger.info('Successfully queried history.')
    except Exception:
        logger.exception('Unable to query history to find previous owners.')
```

Note

- 以下の構文で組み込みの「履歴関数」のクエリを実行することで、ドキュメントのリビジョン履歴を表示できます。

  ```
  SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
  [ WHERE h.metadata.id = 'id' ]
  ```

- start-time および end-time はいずれもオプションです。これらは、バックティック (`` `...` ``) で示すことができる Amazon Ion リテラル値です。詳細については、「Amazon QLDB での PartiQL による Ion のクエリ」を参照してください。
- ベストプラクティスとして、履歴クエリは日付範囲 (start-time および end-time) とドキュメント ID (`metadata.id`) の両方で修飾します。QLDB はトランザクション内の SELECT クエリを処理します。これらは、トランザクションタイムアウト制限が適用されます。

  QLDB 履歴はドキュメント ID によってインデックス付けされるため、現時点では追加の履歴インデックスを作成することはできません。開始時刻と終了時刻を含む履歴クエリでは、日付範囲修飾のメリットが得られます。

2. このプログラムを実行するには、次のコマンドを入力します。

```
python query_history.py
```

vehicle-registration 台帳のドキュメントリビジョンを暗号的に検証するには、「ステップ 7: 台帳内のドキュメントを検証する」に進みます。

ステップ 7: 台帳内のドキュメントを検証する

Amazon QLDB では、SHA-256 の暗号的ハッシュを使用して、台帳のジャーナルのドキュメントの整合性を効率的に検証できます。検証と暗号的ハッシュが QLDB でどのように機能するかについては、「Amazon QLDB でのデータ検証」を参照してください。

このステップでは、vehicle-registration 台帳の VehicleRegistration テーブルのドキュメントリビジョンを確認します。まず、ダイジェストをリクエストします。ダイジェストは出力ファイルとして返され、台帳の変更履歴全体の署名として機能します。次に、そのダイジェストに関連するリビジョンの証明をリクエストします。この証明を使用して、すべての検証チェックに合格すると、リビジョンの整合性が検証されます。

ドキュメントのリビジョンを検証するには

1. 以下の .py ファイルを確認します。これらのファイルには、検証に必要な QLDB オブジェクトと、QLDB レスポンスタイプを文字列に変換するヘルパー関数を提供するユーティリティモジュールが含まれています。

   1. block_address.py

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
```

```
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


def block_address_to_dictionary(ion_dict):
    """
    Convert a block address from IonPyDict into a dictionary.
    Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">,
 sequenceNo: <sequenceNo>}"}

    :type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
    :param ion_dict: The block address value to convert.

    :rtype: dict
    :return: The converted dict.
    """
    block_address = {'IonText': {}}
    if not isinstance(ion_dict, str):
        py_dict = '{{strandId: "{}", sequenceNo:
{}}}'.format(ion_dict['strandId'], ion_dict['sequenceNo'])
        ion_dict = py_dict
    block_address['IonText'] = ion_dict
    return block_address
```

2. verifier.py

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
```

```
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from array import array
from base64 import b64encode
from functools import reduce
from hashlib import sha256
from random import randrange

from amazon.ion.simpleion import loads

HASH_LENGTH = 32
UPPER_BOUND = 8


def parse_proof(value_holder):
    """
    Parse the Proof object returned by QLDB into an iterator.

    The Proof object returned by QLDB is a dictionary like the following:
    {'IonText': '[{{<hash>}},{{<hash>}}]'}

    :type value_holder: dict
    :param value_holder: A structure containing an Ion string value.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyList`
    :return: A list of hash values.
    """
    value_holder = value_holder.get('IonText')
    proof_list = loads(value_holder)
    return proof_list
```

```
def parse_block(value_holder):
    """
    Parse the Block object returned by QLDB and retrieve block hash.

    :type value_holder: dict
    :param value_holder: A structure containing an Ion string value.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyBytes`
    :return: The block hash.
    """
    value_holder = value_holder.get('IonText')
    block = loads(value_holder)
    block_hash = block.get('blockHash')
    return block_hash


def flip_random_bit(original):
    """
    Flip a single random bit in the given hash value.
    This method is used to demonstrate QLDB's verification features.

    :type original: bytes
    :param original: The hash value to alter.

    :rtype: bytes
    :return: The altered hash with a single random bit changed.
    """
    assert len(original) != 0, 'Invalid bytes.'

    altered_position = randrange(len(original))
    bit_shift = randrange(UPPER_BOUND)
    altered_hash = bytearray(original).copy()

    altered_hash[altered_position] = altered_hash[altered_position] ^ (1 <<
 bit_shift)
    return bytes(altered_hash)


def compare_hash_values(hash1, hash2):
    """
    Compare two hash values by converting them into byte arrays, assuming they
 are little endian.

    :type hash1: bytes
```

```
    :param hash1: The hash value to compare.

    :type hash2: bytes
    :param hash2: The hash value to compare.

    :rtype: int
    :return: Zero if the hash values are equal, otherwise return the difference
 of the first pair of non-matching bytes.
    """
    assert len(hash1) == HASH_LENGTH
    assert len(hash2) == HASH_LENGTH

    hash_array1 = array('b', hash1)
    hash_array2 = array('b', hash2)

    for i in range(len(hash_array1) - 1, -1, -1):
        difference = hash_array1[i] - hash_array2[i]
        if difference != 0:
            return difference
    return 0


def join_hash_pairwise(hash1, hash2):
    """
    Take two hash values, sort them, concatenate them, and generate a new hash
 value from the concatenated values.

    :type hash1: bytes
    :param hash1: Hash value to concatenate.

    :type hash2: bytes
    :param hash2: Hash value to concatenate.

    :rtype: bytes
    :return: The new hash value generated from concatenated hash values.
    """
    if len(hash1) == 0:
        return hash2
    if len(hash2) == 0:
        return hash1

    concatenated = hash1 + hash2 if compare_hash_values(hash1, hash2) < 0 else
 hash2 + hash1
    new_hash_lib = sha256()
```

```
    new_hash_lib.update(concatenated)
    new_digest = new_hash_lib.digest()
    return new_digest


def calculate_root_hash_from_internal_hashes(internal_hashes, leaf_hash):
    """
    Combine the internal hashes and the leaf hash until only one root hash
 remains.

    :type internal_hashes: map
    :param internal_hashes: An iterable over a list of hash values.

    :type leaf_hash: bytes
    :param leaf_hash: The revision hash to pair with the first hash in the Proof
 hashes list.

    :rtype: bytes
    :return: The root hash constructed by combining internal hashes.
    """
    root_hash = reduce(join_hash_pairwise, internal_hashes, leaf_hash)
    return root_hash


def build_candidate_digest(proof, leaf_hash):
    """
    Build the candidate digest representing the entire ledger from the Proof
 hashes.

    :type proof: dict
    :param proof: The Proof object.

    :type leaf_hash: bytes
    :param leaf_hash: The revision hash to pair with the first hash in the Proof
 hashes list.

    :rtype: bytes
    :return: The calculated root hash.
    """
    parsed_proof = parse_proof(proof)
    root_hash = calculate_root_hash_from_internal_hashes(parsed_proof, leaf_hash)
    return root_hash
```

```
def verify_document(document_hash, digest, proof):
    """
    Verify document revision against the provided digest.

    :type document_hash: bytes
    :param document_hash: The SHA-256 value representing the document revision to
 be verified.

    :type digest: bytes
    :param digest: The SHA-256 hash value representing the ledger digest.

    :type proof: dict
    :param proof: The Proof object retrieved
 from :func:`pyqldbsamples.get_revision.get_revision`.

    :rtype: bool
    :return: If the document revision verify against the ledger digest.
    """
    candidate_digest = build_candidate_digest(proof, document_hash)
    return digest == candidate_digest


def to_base_64(input):
    """
    Encode input in base64.

    :type input: bytes
    :param input: Input to be encoded.

    :rtype: string
    :return: Return input that has been encoded in base64.
    """
    encoded_value = b64encode(input)
    return str(encoded_value, 'UTF-8')
```

3. qldb_string_utils.py

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
```

```
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
from amazon.ion.simpleion import dumps, loads


def value_holder_to_string(value_holder):
    """
    Returns the string representation of a given `value_holder`.

    :type value_holder: dict
    :param value_holder: The `value_holder` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `value_holder`.
    """
    ret_val = dumps(loads(value_holder), binary=False, indent='  ',
 omit_version_marker=True)
    val = '{{ IonText: {}}}'.format(ret_val)
    return val


def block_response_to_string(block_response):
    """
    Returns the string representation of a given `block_response`.

    :type block_response: dict
    :param block_response: The `block_response` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `block_response`.
    """
```

```
    string = ''
    if block_response.get('Block', {}).get('IonText') is not None:
        string += 'Block: ' + value_holder_to_string(block_response['Block']
['IonText']) + ', '

    if block_response.get('Proof', {}).get('IonText') is not None:
        string += 'Proof: ' + value_holder_to_string(block_response['Proof']
['IonText'])

    return '{' + string + '}'


def digest_response_to_string(digest_response):
    """
    Returns the string representation of a given `digest_response`.

    :type digest_response: dict
    :param digest_response: The `digest_response` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `digest_response`.
    """
    string = ''
    if digest_response.get('Digest') is not None:
        string += 'Digest: ' + str(digest_response['Digest']) + ', '

    if digest_response.get('DigestTipAddress', {}).get('IonText') is not None:
        string += 'DigestTipAddress: ' +
 value_holder_to_string(digest_response['DigestTipAddress']['IonText'])

    return '{' + string + '}'
```

2. 2 つの .py プログラム (get_digest.py および get_revision.py) を使用して、次の手順を実行します。

   - vehicle-registration 台帳に新しいダイジェストをリクエストします。
   - VehicleRegistration テーブルの VIN が 1N4AL11D75C109151 のドキュメントについて、各リビジョンの証明をリクエストします。
   - 返されたダイジェストと証明を使用して、ダイジェストを再計算することで、リビジョンを検証します。

   get_digest.py プログラムには、次のコードが含まれています。

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.qldb.qldb_string_utils import digest_response_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')


def get_digest_result(name):
    """
    Get the digest of a ledger's journal.

    :type name: str
    :param name: Name of the ledger to operate on.
```

```
    :rtype: dict
    :return: The digest in a 256-bit hash value and a block address.
    """
    logger.info("Let's get the current digest of the ledger named {}".format(name))
    result = qldb_client.get_digest(Name=name)
    logger.info('Success. LedgerDigest:
 {}.'.format(digest_response_to_string(result)))
    return result


def main(ledger_name=Constants.LEDGER_NAME):
    """
    This is an example for retrieving the digest of a particular ledger.
    """
    try:
        get_digest_result(ledger_name)
    except Exception as e:
        logger.exception('Unable to get a ledger digest!')
        raise e


if __name__ == '__main__':
    main()
```

> **Note**
>
> get_digest_result 関数を使用して、台帳のジャーナルの現在のティップを含むダイジェストをリクエストします。ジャーナルのティップとは、QLDB がリクエストを受けた時点でコミット済みの最新のブロックのことです。

get_revision.py プログラムには、次のコードが含まれています。

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
```

```
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from amazon.ion.simpleion import loads
from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.get_digest import get_digest_result
from pyqldbsamples.model.sample_data import SampleData, convert_object_to_ion
from pyqldbsamples.qldb.block_address import block_address_to_dictionary
from pyqldbsamples.verifier import verify_document, flip_random_bit, to_base_64
from pyqldbsamples.connect_to_ledger import create_qldb_driver
from pyqldbsamples.qldb.qldb_string_utils import value_holder_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')


def get_revision(ledger_name, document_id, block_address, digest_tip_address):
    """
    Get the revision data object for a specified document ID and block address.
    Also returns a proof of the specified revision for verification.

    :type ledger_name: str
    :param ledger_name: Name of the ledger containing the document to query.
```

```
    :type document_id: str
    :param document_id: Unique ID for the document to be verified, contained in
 the committed view of the document.

    :type block_address: dict
    :param block_address: The location of the block to request.

    :type digest_tip_address: dict
    :param digest_tip_address: The latest block location covered by the digest.

    :rtype: dict
    :return: The response of the request.
    """
    result = qldb_client.get_revision(Name=ledger_name,
 BlockAddress=block_address, DocumentId=document_id,
                                      DigestTipAddress=digest_tip_address)
    return result


def lookup_registration_for_vin(driver, vin):
    """
    Query revision history for a particular vehicle for verification.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
 with.

    :rtype: :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :return: Cursor on the result set of the statement query.
    """
    logger.info("Querying the 'VehicleRegistration' table for VIN:
 {}...".format(vin))
    query = 'SELECT * FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?'
    return driver.execute_lambda(lambda txn: txn.execute_statement(query,
 convert_object_to_ion(vin)))


def verify_registration(driver, ledger_name, vin):
    """
    Verify each version of the registration for the given VIN.
```

```
    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type ledger_name: str
    :param ledger_name: The ledger to get digest from.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
with.

    :raises AssertionError: When verification failed.
    """
    logger.info("Let's verify the registration with VIN = {}, in ledger =
{}.".format(vin, ledger_name))
    digest = get_digest_result(ledger_name)
    digest_bytes = digest.get('Digest')
    digest_tip_address = digest.get('DigestTipAddress')

    logger.info('Got a ledger digest: digest tip address = {}, digest =
{}.'.format(
        value_holder_to_string(digest_tip_address.get('IonText')),
to_base_64(digest_bytes)))

    logger.info('Querying the registration with VIN = {} to verify each version
of the registration...'.format(vin))
    cursor = lookup_registration_for_vin(driver, vin)
    logger.info('Getting a proof for the document.')

    for row in cursor:
        block_address = row.get('blockAddress')
        document_id = row.get('metadata').get('id')

        result = get_revision(ledger_name, document_id,
block_address_to_dictionary(block_address), digest_tip_address)
        revision = result.get('Revision').get('IonText')
        document_hash = loads(revision).get('hash')

        proof = result.get('Proof')
        logger.info('Got back a proof: {}.'.format(proof))

        verified = verify_document(document_hash, digest_bytes, proof)
        if not verified:
            raise AssertionError('Document revision is not verified.')
        else:
```

```
            logger.info('Success! The document is verified.')

        altered_document_hash = flip_random_bit(document_hash)
        logger.info("Flipping one bit in the document's hash and assert that the
 document is NOT verified. "
                    "The altered document hash is:
 {}.".format(to_base_64(altered_document_hash)))
        verified = verify_document(altered_document_hash, digest_bytes, proof)
        if verified:
            raise AssertionError('Expected altered document hash to not be
 verified against digest.')
        else:
            logger.info('Success! As expected flipping a bit in the document
 hash causes verification to fail.')

        logger.info('Finished verifying the registration with VIN = {} in ledger
 = {}.'.format(vin, ledger_name))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Verify the integrity of a document revision in a QLDB ledger.
    """
    registration = SampleData.VEHICLE_REGISTRATION[0]
    vin = registration['VIN']
    try:
        with create_qldb_driver(ledger_name) as driver:
            verify_registration(driver, ledger_name, vin)
    except Exception as e:
        logger.exception('Unable to verify revision.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from amazon.ion.simpleion import loads
from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.get_digest import get_digest_result
from pyqldbsamples.model.sample_data import SampleData, convert_object_to_ion
from pyqldbsamples.qldb.block_address import block_address_to_dictionary
from pyqldbsamples.verifier import verify_document, flip_random_bit, to_base_64
from pyqldbsamples.connect_to_ledger import create_qldb_session
from pyqldbsamples.qldb.qldb_string_utils import value_holder_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')


def get_revision(ledger_name, document_id, block_address, digest_tip_address):
    """
    Get the revision data object for a specified document ID and block address.
    Also returns a proof of the specified revision for verification.

    :type ledger_name: str
```

```
    :param ledger_name: Name of the ledger containing the document to query.

    :type document_id: str
    :param document_id: Unique ID for the document to be verified, contained in
 the committed view of the document.

    :type block_address: dict
    :param block_address: The location of the block to request.

    :type digest_tip_address: dict
    :param digest_tip_address: The latest block location covered by the digest.

    :rtype: dict
    :return: The response of the request.
    """
    result = qldb_client.get_revision(Name=ledger_name,
 BlockAddress=block_address, DocumentId=document_id,
                                      DigestTipAddress=digest_tip_address)
    return result


def lookup_registration_for_vin(qldb_session, vin):
    """
    Query revision history for a particular vehicle for verification.

    :type qldb_session: :py:class:`pyqldb.session.qldb_session.QldbSession`
    :param qldb_session: An instance of the QldbSession class.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
 with.

    :rtype: :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :return: Cursor on the result set of the statement query.
    """
    logger.info("Querying the 'VehicleRegistration' table for VIN:
 {}...".format(vin))
    query = 'SELECT * FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?'
    parameters = [convert_object_to_ion(vin)]
    cursor = qldb_session.execute_statement(query, parameters)
    return cursor


def verify_registration(qldb_session, ledger_name, vin):
```

```
    """
    Verify each version of the registration for the given VIN.

    :type qldb_session: :py:class:`pyqldb.session.qldb_session.QldbSession`
    :param qldb_session: An instance of the QldbSession class.

    :type ledger_name: str
    :param ledger_name: The ledger to get digest from.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
with.

    :raises AssertionError: When verification failed.
    """
    logger.info("Let's verify the registration with VIN = {}, in ledger =
{}.".format(vin, ledger_name))
    digest = get_digest_result(ledger_name)
    digest_bytes = digest.get('Digest')
    digest_tip_address = digest.get('DigestTipAddress')

    logger.info('Got a ledger digest: digest tip address = {}, digest =
{}.'.format(
        value_holder_to_string(digest_tip_address.get('IonText')),
to_base_64(digest_bytes)))

    logger.info('Querying the registration with VIN = {} to verify each version
of the registration...'.format(vin))
    cursor = lookup_registration_for_vin(qldb_session, vin)
    logger.info('Getting a proof for the document.')

    for row in cursor:
        block_address = row.get('blockAddress')
        document_id = row.get('metadata').get('id')

        result = get_revision(ledger_name, document_id,
block_address_to_dictionary(block_address), digest_tip_address)
        revision = result.get('Revision').get('IonText')
        document_hash = loads(revision).get('hash')

        proof = result.get('Proof')
        logger.info('Got back a proof: {}.'.format(proof))

        verified = verify_document(document_hash, digest_bytes, proof)
```

```
        if not verified:
            raise AssertionError('Document revision is not verified.')
        else:
            logger.info('Success! The document is verified.')

        altered_document_hash = flip_random_bit(document_hash)
        logger.info("Flipping one bit in the document's hash and assert that the
 document is NOT verified. "
                    "The altered document hash is:
 {}.".format(to_base_64(altered_document_hash)))
        verified = verify_document(altered_document_hash, digest_bytes, proof)
        if verified:
            raise AssertionError('Expected altered document hash to not be
 verified against digest.')
        else:
            logger.info('Success! As expected flipping a bit in the document
 hash causes verification to fail.')

        logger.info('Finished verifying the registration with VIN = {} in ledger
 = {}.'.format(vin, ledger_name))


if __name__ == '__main__':
    """
    Verify the integrity of a document revision in a QLDB ledger.
    """
    registration = SampleData.VEHICLE_REGISTRATION[0]
    vin = registration['VIN']
    try:
        with create_qldb_session() as session:
            verify_registration(session, Constants.LEDGER_NAME, vin)
    except Exception:
        logger.exception('Unable to verify revision.')
```

> **Note**
>
> get_revision 関数から指定したドキュメントリビジョンの証明が返されると、このプログラムはクライアント側 API を使用してそのリビジョンを検証します。

3. このプログラムを実行するには、次のコマンドを入力します。

```
python get_revision.py
```

vehicle-registration 台帳を使用する必要がなくなった場合は、「ステップ 8 (オプション): リソースをクリーンアップする」に進みます。

ステップ 8 (オプション): リソースをクリーンアップする

vehicle-registration 台帳は引き続き使用できます。ただし、不要になった場合は、削除することをお勧めします。

台帳を削除するには

1. 次のプログラム (delete_ledger.py) を使用して vehicle-registration 台帳とその内容全体を削除します。

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO
from time import sleep
```

```
from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.describe_ledger import describe_ledger

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

LEDGER_DELETION_POLL_PERIOD_SEC = 20


def delete_ledger(ledger_name):
    """
    Send a request to QLDB to delete the specified ledger.

    :type ledger_name: str
    :param ledger_name: Name for the ledger to be deleted.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info('Attempting to delete the ledger with name:
 {}...'.format(ledger_name))
    result = qldb_client.delete_ledger(Name=ledger_name)
    logger.info('Success.')
    return result


def wait_for_deleted(ledger_name):
    """
    Wait for the ledger to be deleted.

    :type ledger_name: str
    :param ledger_name: The ledger to check on.
    """
    logger.info('Waiting for the ledger to be deleted...')
    while True:
        try:
            describe_ledger(ledger_name)
            logger.info('The ledger is still being deleted. Please wait...')
            sleep(LEDGER_DELETION_POLL_PERIOD_SEC)
        except qldb_client.exceptions.ResourceNotFoundException:
```

```
            logger.info('Success. The ledger is deleted.')
            break


def set_deletion_protection(ledger_name, deletion_protection):
    """
    Update an existing ledger's deletion protection.

    :type ledger_name: str
    :param ledger_name: Name of the ledger to update.

    :type deletion_protection: bool
    :param deletion_protection: Enable or disable the deletion protection.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info("Let's set deletion protection to {} for the ledger with name
 {}.".format(deletion_protection,

       ledger_name))
    result = qldb_client.update_ledger(Name=ledger_name,
 DeletionProtection=deletion_protection)
    logger.info('Success. Ledger updated: {}'.format(result))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Delete a ledger.
    """
    try:
        set_deletion_protection(ledger_name, False)
        delete_ledger(ledger_name)
        wait_for_deleted(ledger_name)
    except Exception as e:
        logger.exception('Unable to delete the ledger.')
        raise e


if __name__ == '__main__':
    main()
```

> **Note**
>
> 台帳に対して削除保護が有効になっている場合は、QLDB API を使用して台帳を削除する前に、まず無効にする必要があります。

delete_ledger.py ファイルは以下のプログラム (describe_ledger.py) にも依存しています。

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from boto3 import client

from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)
```

```
qldb_client = client('qldb')


def describe_ledger(ledger_name):
    """
    Describe a ledger.

    :type ledger_name: str
    :param ledger_name: Name of the ledger to describe.
    """
    logger.info('describe ledger with name: {}.'.format(ledger_name))
    result = qldb_client.describe_ledger(Name=ledger_name)
    result.pop('ResponseMetadata')
    logger.info('Success. Ledger description: {}.'.format(result))
    return result


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Describe a QLDB ledger.
    """
    try:
        describe_ledger(ledger_name)
    except Exception as e:
        logger.exception('Unable to describe a ledger.')
        raise e


if __name__ == '__main__':
    main()
```

2. このプログラムを実行するには、次のコマンドを入力します。

```
python delete_ledger.py
```

Amazon QLDB で Amazon Ion のデータ型を操作する

Amazon QLDB では、Amazon Ion 形式でデータを格納します。QLDB でデータを操作するには、サポートされているプログラミング言語の依存関係として、Ion ライブラリを使用する必要があります。

このセクションでは、ネイティブ型から同等のイオン型にデータを変換する方法と、その逆の方法について説明します。このリファレンスガイドでは、QLDB ドライバーを使用して QLDB 台帳内の Ion データを処理するコード例を示します。これには、Java、.NET (C#)、Go、Node.js (TypeScript)、Python のコード例が含まれています。

トピック

前提条件

次のコード例では、`ExampleTable` という名前のテーブルを持つアクティブ台帳に接続する QLDB ドライバーインスタンスがあることを前提としています。テーブルには、次の 8 つのフィールドを持つ既存のドキュメントが 1 つ含まれています。

- ExampleBool
- ExampleInt
- ExampleFloat
- ExampleDecimal
- ExampleTimestamp
- ExampleString
- ExampleBlob
- ExampleList

> **Note**
>
> ここでは参照目的で、各フィールドに格納されている型がその名前と一致すると仮定します。実際には、QLDB で、ドキュメントフィールドのスキーマまたはデータ型の定義が強制されることはありません。

Bool

次のコード例は、Ion ブール型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

boolean exampleBoolean = driver.execute((txn) -> {
    // Transforming a Java boolean to Ion
    boolean aBoolean = true;
    IonValue ionBool = ionSystem.newBool(aBoolean);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleBool = ?", ionBool);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleBool from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java boolean
        // Cast IonValue to IonBool first
        aBoolean = ((IonBool)ionValue).booleanValue();
    }

    // exampleBoolean is now the value fetched from QLDB
    return aBoolean;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...
```

```
IValueFactory valueFactory = new ValueFactory();

// Transforming a C# bool to Ion.
bool nativeBool  = true;
IIonValue ionBool = valueFactory.NewBool(nativeBool);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleBool = ?", ionBool);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleBool from ExampleTable");
});

bool? retrievedBool = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# bool.
    retrievedBool = ionValue.BoolValue;
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleBool, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aBool := true

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleBool = ?", aBool)
 if err != nil {
  return nil, err
 }
```

```
 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleBool FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult bool
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleBool is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonBoolean(driver: QldbDriver): Promise<boolean> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleBool = ?", true);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleBool FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Boolean
            const boolValue: boolean = ionValue.booleanValue();
            return boolValue;
        })
    );
}
```

Python

```
def update_and_query_ion_bool(txn):
    # QLDB can take in a Python bool
    a_bool = True

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleBool = ?", a_bool)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleBool FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python bool is a child class of Python bool
        a_bool = ion_value

    # example_bool is now the value fetched from QLDB
    return a_bool


example_bool = driver.execute_lambda(lambda txn: update_and_query_ion_bool(txn))
```

Int

次のコード例は、Ion 整数型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

int exampleInt = driver.execute((txn) -> {
    // Transforming a Java int to Ion
    int aInt = 256;
    IonValue ionInt = ionSystem.newInt(aInt);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleInt = ?", ionInt);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleInt from ExampleTable");
```

```
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java int
        // Cast IonValue to IonInt first
        aInt = ((IonInt)ionValue).intValue();
    }

    // exampleInt is now the value fetched from QLDB
    return aInt;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# int to Ion.
int nativeInt  = 256;
IIonValue ionInt = valueFactory.NewInt(nativeInt);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleInt = ?", ionInt);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleInt from ExampleTable");
});

int? retrievedInt = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# int.
    retrievedInt = ionValue.IntValue;
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleInt, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aInt := 256

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleInt = ?", aInt)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleInt FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult int
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleInt is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonInt(driver: QldbDriver): Promise<number> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
```

```
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleInt = ?", 256);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleInt FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Number
            const intValue: number = ionValue.numberValue();
            return intValue;
        })
    );
}
```

Python

```
def update_and_query_ion_int(txn):
    # QLDB can take in a Python int
    a_int = 256

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleInt = ?", a_int)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleInt FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python int is a child class of Python int
        a_int = ion_value

    # example_int is now the value fetched from QLDB
    return a_int


example_int = driver.execute_lambda(lambda txn: update_and_query_ion_int(txn))
```

浮動小数点

次のコード例は、Ion 浮動小数点型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

float exampleFloat = driver.execute((txn) -> {
    // Transforming a Java float to Ion
    float aFloat = 256;
    IonValue ionFloat = ionSystem.newFloat(aFloat);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleFloat from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java float
        // Cast IonValue to IonFloat first
        aFloat = ((IonFloat)ionValue).floatValue();
    }

    // exampleFloat is now the value fetched from QLDB
    return aFloat;
});
```

.NET

C# float 型の使用

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# float to Ion.
float nativeFloat = 256;
IIonValue ionFloat = valueFactory.NewFloat(nativeFloat);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
```

```
    await txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleFloat from ExampleTable");
});

float? retrievedFloat = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# float. We cast ionValue.DoubleValue to a float
    // but be cautious, this is a down-cast and can lose precision.
    retrievedFloat = (float)ionValue.DoubleValue;
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

C# double 型の使用

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# double to Ion.
double nativeDouble = 256;
IIonValue ionFloat = valueFactory.NewFloat(nativeDouble);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleFloat from ExampleTable");
});
```

```
double? retrievedDouble = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# double.
    retrievedDouble = ionValue.DoubleValue;
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleFloat, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aFloat := float32(256)

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", aFloat)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleFloat FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  // float64 would work as well
  var decodedResult float32
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }
```

```
  // exampleFloat is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonFloat(driver: QldbDriver): Promise<number> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleFloat = ?", 25.6);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleFloat FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Number
            const floatValue: number = ionValue.numberValue();
            return floatValue;
        })
    );
}
```

Python

```
def update_and_query_ion_float(txn):
    # QLDB can take in a Python float
    a_float = float(256)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleFloat = ?", a_float)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleFloat FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python float is a child class of Python float
        a_float = ion_value
```

```
    # example_float is now the value fetched from QLDB
    return a_float


example_float = driver.execute_lambda(lambda txn: update_and_query_ion_float(txn))
```

10 進数

次のコード例は、Ion 10 進数型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

double exampleDouble = driver.execute((txn) -> {
    // Transforming a Java double to Ion
    double aDouble = 256;
    IonValue ionDecimal = ionSystem.newDecimal(aDouble);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleDecimal = ?", ionDecimal);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleDecimal from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java double
        // Cast IonValue to IonDecimal first
        aDouble = ((IonDecimal)ionValue).doubleValue();
    }

    // exampleDouble is now the value fetched from QLDB
    return aDouble;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...
```

```
IValueFactory valueFactory = new ValueFactory();

// Transforming a C# decimal to Ion.
decimal nativeDecimal = 256.8723m;
IIonValue ionDecimal = valueFactory.NewDecimal(nativeDecimal);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleDecimal = ?", ionDecimal);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleDecimal from ExampleTable");
});

decimal? retrievedDecimal = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# decimal.
    retrievedDecimal = ionValue.DecimalValue;
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleDecimal, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aDecimal, err := ion.ParseDecimal("256")
 if err != nil {
  return nil, err
 }

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleDecimal = ?", aDecimal)
 if err != nil {
```

```
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleDecimal FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult ion.Decimal
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleDecimal is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonDecimal(driver: QldbDriver): Promise<Decimal> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Creating a Decimal value. Decimal is an Ion Type with high precision
            let ionDecimal: Decimal = dom.load("2.5d-6").decimalValue();
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleDecimal = ?",
 ionDecimal);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleDecimal FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Get the Ion Decimal
            ionDecimal = ionValue.decimalValue();
            return ionDecimal;
        })
```

```
    );
}
```

> **Note**
>
> また、`ionValue.numberValue()` を使用して、Ion 10 進数型の値を JavaScript の値に変換することもできます。`ionValue.numberValue()` を使用するとパフォーマンスは向上しますが、精度が低くなります。

Python

```
def update_and_query_ion_decimal(txn):
    # QLDB can take in a Python decimal
    a_decimal = Decimal(256)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleDecimal = ?", a_decimal)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleDecimal FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python decimal is a child class of Python decimal
        a_decimal = ion_value

    # example_decimal is now the value fetched from QLDB
    return a_decimal


example_decimal = driver.execute_lambda(lambda txn:
 update_and_query_ion_decimal(txn))
```

タイムスタンプ

次のコード例は、Ion タイムスタンプ型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

Date exampleDate = driver.execute((txn) -> {
    // Transforming a Java Date to Ion
    Date aDate = new Date();
    IonValue ionTimestamp = ionSystem.newUtcTimestamp(aDate);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleTimestamp = ?", ionTimestamp);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleTimestamp from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java Date
        // Cast IonValue to IonTimestamp first
        aDate = ((IonTimestamp)ionValue).dateValue();
    }

    // exampleDate is now the value fetched from QLDB
    return aDate;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using Amazon.IonDotnet;
...

IValueFactory valueFactory = new ValueFactory();

// Convert C# native DateTime to Ion.
DateTime nativeDateTime = DateTime.Now;

// First convert it to a timestamp object from Ion.
Timestamp timestamp = new Timestamp(nativeDateTime);
// Then convert to Ion timestamp.
IIonValue ionTimestamp = valueFactory.NewTimestamp(timestamp);
```

```
IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleTimestamp = ?", ionTimestamp);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleTimestamp from ExampleTable");
});

DateTime? retrievedDateTime = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# DateTime.
    retrievedDateTime = ionValue.TimestampValue.DateTimeValue;
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleTimestamp, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aTimestamp := time.Date(2006, time.May, 20, 12, 30, 0, 0, time.UTC)
 if err != nil {
  return nil, err
 }

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleTimestamp = ?", aTimestamp)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleTimestamp FROM ExampleTable")
 if err != nil {
```

```
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult ion.Timestamp
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleTimestamp is now the value fetched from QLDB
  return decodedResult.GetDateTime(), nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonTimestamp(driver: QldbDriver): Promise<Date> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            let exampleDateTime: Date = new Date(Date.now());
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleTimestamp = ?",
 exampleDateTime);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleTimestamp FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Date
            exampleDateTime = ionValue.timestampValue().getDate();
            return exampleDateTime;
        })
    );
}
```

Python

```
def update_and_query_ion_timestamp(txn):
    # QLDB can take in a Python timestamp
```

```
    a_datetime = datetime.now()

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleTimestamp = ?",
 a_datetime)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleTimestamp FROM
 ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python timestamp is a child class of Python datetime
        a_timestamp = ion_value

    # example_timestamp is now the value fetched from QLDB
    return a_timestamp


example_timestamp = driver.execute_lambda(lambda txn:
 update_and_query_ion_timestamp(txn))
```

文字列

次のコード例は、Ion 文字列型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

String exampleString = driver.execute((txn) -> {
    // Transforming a Java String to Ion
    String aString = "Hello world!";
    IonValue ionString = ionSystem.newString(aString);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleString = ?", ionString);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleString from ExampleTable");
    // Assume there is only one document in ExampleTable
```

```
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java String
        // Cast IonValue to IonString first
        aString = ((IonString)ionValue).stringValue();
    }

    // exampleString is now the value fetched from QLDB
    return aString;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Convert C# string to Ion.
String nativeString = "Hello world!";
IIonValue ionString = valueFactory.NewString(nativeString);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleString = ?", ionString);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleString from ExampleTable");
});

String retrievedString = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# string.
    retrievedString = ionValue.StringValue;
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleString, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aString := "Hello World!"

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleString = ?", aString)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleString FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult string
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleString is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonString(driver: QldbDriver): Promise<string> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
```

```
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleString = ?", "Hello
 World!");

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleString FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript String
            const stringValue: string = ionValue.stringValue();
            return stringValue;
        })
    );
}
```

Python

```
def update_and_query_ion_string(txn):
    # QLDB can take in a Python string
    a_string = "Hello world!"

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleString = ?", a_string)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleString FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python string is a child class of Python string
        a_string = ion_value

    # example_string is now the value fetched from QLDB
    return a_string


example_string = driver.execute_lambda(lambda txn: update_and_query_ion_string(txn))
```

blob

次のコード例は、Ion blob 型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

byte[] exampleBytes = driver.execute((txn) -> {
    // Transforming a Java byte array to Ion
    // Transform any arbitrary data to a byte array to store in QLDB
    String aString = "Hello world!";
    byte[] aByteArray = aString.getBytes();
    IonValue ionBlob = ionSystem.newBlob(aByteArray);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleBlob = ?", ionBlob);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleBlob from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java byte array
        // Cast IonValue to IonBlob first
        aByteArray = ((IonBlob)ionValue).getBytes();
    }

    // exampleBytes is now the value fetched from QLDB
    return aByteArray;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using System.Text;
...

IValueFactory valueFactory = new ValueFactory();

// Transform any arbitrary data to a byte array to store in QLDB.
string nativeString = "Hello world!";
```

```
byte[] nativeByteArray = Encoding.UTF8.GetBytes(nativeString);
// Transforming a C# byte array to Ion.
IIonValue ionBlob = valueFactory.NewBlob(nativeByteArray);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleBlob = ?", ionBlob);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleBlob from ExampleTable");
});

byte[] retrievedByteArray = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# byte array.
    retrievedByteArray = ionValue.Bytes().ToArray();
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleBlob, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aBlob := []byte("Hello World!")

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleBlob = ?", aBlob)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleBlob FROM ExampleTable")
```

```
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult []byte
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleBlob is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonBlob(driver: QldbDriver): Promise<Uint8Array> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            const enc = new TextEncoder();
            let blobValue: Uint8Array = enc.encode("Hello World!");
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleBlob = ?", blobValue);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleBlob FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to TypeScript Uint8Array
            blobValue = ionValue.uInt8ArrayValue();
            return blobValue;
        })
    );
}
```

Python

```
def update_and_query_ion_blob(txn):
```

```
    # QLDB can take in a Python byte array
    a_string = "Hello world!"
    a_byte_array = str.encode(a_string)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleBlob = ?", a_byte_array)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleBlob FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python blob is a child class of Python byte array
        a_blob = ion_value

    # example_blob is now the value fetched from QLDB
    return a_blob


example_blob = driver.execute_lambda(lambda txn: update_and_query_ion_blob(txn))
```

リスト

次のコード例は、Ion リスト型の処理方法を示しています。

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

List<Integer> exampleList = driver.execute((txn) -> {
    // Transforming a Java List to Ion
    List<Integer> aList = new ArrayList<>();
    // Add 5 Integers to the List for the sake of example
    for (int i = 0; i < 5; i++) {
        aList.add(i);
    }
    // Create an empty Ion List
    IonList ionList = ionSystem.newEmptyList();
    // Add the 5 Integers to the Ion List
    for (Integer i : aList) {
        // Convert each Integer to Ion ints first to add it to the Ion List
```

```
        ionList.add(ionSystem.newInt(i));
    }

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleList = ?", (IonValue) ionList);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleList from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Iterate through the Ion List to map it to a Java List
        List<Integer> intList = new ArrayList<>();
        for (IonValue ionInt : (IonList)ionValue) {
            // Convert the 5 Ion ints to Java Integers
            intList.add(((IonInt)ionInt).intValue());
        }
        aList = intList;
    }

    // exampleList is now the value fetched from QLDB
    return aList;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using System.Collections.Generic;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# list to Ion.
IIonValue ionList = valueFactory.NewEmptyList();
foreach (int i in new List<int> {0, 1, 2, 3, 4})
{
    // Convert to Ion int and add to Ion list.
    ionList.Add(valueFactory.NewInt(i));
}

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
```

```
    await txn.Execute("UPDATE ExampleTable SET ExampleList = ?", ionList);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleList from ExampleTable");
});


List<int> retrievedList = new List<int>();
await foreach (IIonValue ionValue in selectResult)
{
    // Iterate through the Ion List to map it to a C# list.
    foreach (IIonValue ionInt in ionValue)
    {
        retrievedList.Add(ionInt.IntValue);
    }
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
exampleList, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aList := []int{1, 2, 3, 4, 5}

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleList = ?", aList)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleList FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
```

```
 if result.Next(txn) {
  var decodedResult []int
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleList is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonList(driver: QldbDriver): Promise<number[]> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            let listOfNumbers: number[] = [1, 2, 3, 4, 5];
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleList = ?",
 listOfNumbers);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleList FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Get Ion List
            const ionList: dom.Value[] = ionValue.elements();
            // Iterate through the Ion List to map it to a JavaScript Array
            let intList: number[] = [];
            ionList.forEach(item => {
                // Transforming Ion to a TypeScript Number
                const intValue: number = item.numberValue();
                intList.push(intValue);
            });
            listOfNumbers = intList;
            return listOfNumbers;
        })
    );
}
```

Python

```
def update_and_query_ion_list(txn):
    # QLDB can take in a Python list
    a_list = list()
    for i in range(0, 5):
        a_list.append(i)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleList = ?", a_list)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleList FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python blob is a child class of Python list
        a_list = ion_value

    # example_list is now the value fetched from QLDB
    return a_list


example_list = driver.execute_lambda(lambda txn: update_and_query_ion_list(txn))
```

Struct

QLDB の `struct` データ型は、Ion のほかの型とは特に異なります。テーブルに挿入する最上位レベルのドキュメントは、`struct` 型でなければなりません。ドキュメントフィールドには、ネストされた `struct` を格納することもできます。

簡単にするために、次の例では、`ExampleString` と `ExampleInt` のフィールドのみがあるドキュメントを定義しています。

Java

```
class ExampleStruct {
    public String exampleString;
    public int exampleInt;

    public ExampleStruct(String exampleString, int exampleInt) {
```

```
        this.exampleString = exampleString;
        this.exampleInt = exampleInt;
    }
}

// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

ExampleStruct examplePojo = driver.execute((txn) -> {
    // Transforming a POJO to Ion
    ExampleStruct aPojo = new ExampleStruct("Hello world!", 256);
    // Create an empty Ion struct
    IonStruct ionStruct = ionSystem.newEmptyStruct();
    // Map the fields of the POJO to Ion values and put them in the Ion struct
    ionStruct.add("ExampleString", ionSystem.newString(aPojo.exampleString));
    ionStruct.add("ExampleInt", ionSystem.newInt(aPojo.exampleInt));

    // Insertion into QLDB
    txn.execute("INSERT INTO ExampleTable ?", ionStruct);

    // Fetching from QLDB
    Result result = txn.execute("SELECT * from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Map the fields of the Ion struct to Java values and construct a new POJO
        ionStruct = (IonStruct)ionValue;
        IonString exampleString = (IonString)ionStruct.get("ExampleString");
        IonInt exampleInt = (IonInt)ionStruct.get("ExampleInt");
        aPojo = new ExampleStruct(exampleString.stringValue(),
 exampleInt.intValue());
    }

    // examplePojo is now the document fetched from QLDB
    return aPojo;
});
```

Jackson ライブラリを使用して、データ型を Ion との間でマッピングすることもできます。このライブラリはほかの Ion データ型をサポートしていますが、この例では、`struct` 型に重点を置いています。

```
class ExampleStruct {
    public String exampleString;
```

```
    public int exampleInt;

    @JsonCreator
    public ExampleStruct(@JsonProperty("ExampleString") String exampleString,
                         @JsonProperty("ExampleInt") int exampleInt) {
        this.exampleString = exampleString;
        this.exampleInt = exampleInt;
    }

    @JsonProperty("ExampleString")
    public String getExampleString() {
        return this.exampleString;
    }

    @JsonProperty("ExampleInt")
    public int getExampleInt() {
        return this.exampleInt;
    }
}

// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();
// Instantiate an IonObjectMapper from the Jackson library
IonObjectMapper ionMapper = new IonValueMapper(ionSystem);

ExampleStruct examplePojo = driver.execute((txn) -> {
    // Transforming a POJO to Ion
    ExampleStruct aPojo = new ExampleStruct("Hello world!", 256);
    IonValue ionStruct;
    try  {
        // Use the mapper to convert Java objects into Ion
        ionStruct = ionMapper.writeValueAsIonValue(aPojo);
    } catch (IOException e) {
        // Wrap the exception and throw it for the sake of simplicity in this
 example
        throw new RuntimeException(e);
    }

    // Insertion into QLDB
    txn.execute("INSERT INTO ExampleTable ?", ionStruct);

    // Fetching from QLDB
    Result result = txn.execute("SELECT * from ExampleTable");
    // Assume there is only one document in ExampleTable
```

```
    for (IonValue ionValue : result)
    {
        // Use the mapper to convert Ion to Java objects
        try {
            aPojo = ionMapper.readValue(ionValue, ExampleStruct.class);
        } catch (IOException e) {
            // Wrap the exception and throw it for the sake of simplicity in this
 example
            throw new RuntimeException(e);
        }
    }

    // examplePojo is now the document fetched from QLDB
    return aPojo;
});
```

.NET

Amazon.QLDB.Driver.Serialization ライブラリを使用して、ネイティブ C# データ型を Ion にマッピングしたり、Ion からネイティブデータ型をマッピングしたりします。このライブラリはほかの Ion データ型をサポートしていますが、この例では、`struct` 型に重点を置いています。

```
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;
...

IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
    .WithLedger("vehicle-registration")
    // Add Serialization library
    .WithSerializer(new ObjectSerializer())
    .Build();

// Creating a C# POCO.
ExampleStruct exampleStruct = new ExampleStruct
{
    ExampleString = "Hello world!",
    ExampleInt = 256
};

IAsyncResult<ExampleStruct> selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
```

```
    await txn.Execute(txn.Query<Document>("UPDATE ExampleTable SET ExampleStruct
 = ?", exampleStruct));

    // Fetching from QLDB.
    return await txn.Execute(txn.Query<ExampleStruct>("SELECT VALUE ExampleStruct
 from ExampleTable"));
});

await foreach (ExampleStruct row in selectResult)
{
    Console.WriteLine(row.ExampleString);
    Console.WriteLine(row.ExampleInt);
}
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

または、Amazon.IonDotNet.Builders ライブラリを使用して Ion データ型を処理することもできます。

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Creating Ion struct.
IIonValue ionStruct = valueFactory.NewEmptyStruct();
ionStruct.SetField("ExampleString", valueFactory.NewString("Hello world!"));
ionStruct.SetField("ExampleInt", valueFactory.NewInt(256));

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleStruct = ?", ionStruct);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleStruct from ExampleTable");
});
```

```
string retrievedString = null;
int? retrievedInt = null;

await foreach (IIonValue ionValue in selectResult)
{
    retrievedString = ionValue.GetField("ExampleString").StringValue;
    retrievedInt = ionValue.GetField("ExampleInt").IntValue;
}
```

Go

```
exampleStruct, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aStruct := map[string]interface{} {
  "ExampleString": "Hello World!",
  "ExampleInt": 256,
 }

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleStruct = ?", aStruct)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleStruct FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult map[string]interface{}
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleStruct is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonStruct(driver: QldbDriver): Promise<any> {
    let exampleStruct: any = {stringValue: "Hello World!", intValue: 256};
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Inserting into QLDB
            await txn.execute("INSERT INTO ExampleTable ?", exampleStruct);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT * FROM
 ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // We can get all the keys of Ion struct and their associated values
            const ionFieldNames: string[] = ionValue.fieldNames();

            // Getting key and value of Ion struct to TypeScript String and Number
            const nativeStringVal: string =
 ionValue.get(ionFieldNames[0]).stringValue();
            const nativeIntVal: number =
 ionValue.get(ionFieldNames[1]).numberValue();
            // Alternatively, we can access to Ion struct fields, using their
 literal field names:
            //   const nativeStringVal = ionValue.get("stringValue").stringValue();
            //   const nativeIntVal = ionValue.get("intValue").numberValue();

            exampleStruct = {[ionFieldNames[0]]: nativeStringVal,
 [ionFieldNames[1]]: nativeIntVal};
            return exampleStruct;
        })
    );
}
```

Python

```
def update_and_query_ion_struct(txn):
    # QLDB can take in a Python struct
    a_struct = {"ExampleString": "Hello world!", "ExampleInt": 256}

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleStruct = ?", a_struct)
```

```
    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleStruct FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python struct is a child class of Python struct
        a_struct = ion_value

    # example_struct is now the value fetched from QLDB
    return a_struct


example_struct = driver.execute_lambda(lambda txn: update_and_query_ion_struct(txn))
```

null 値と動的型

QLDB では、オープンコンテンツがサポートされ、ドキュメントフィールドのスキーマやデータ型の定義が強制されることはありません。Ion null 値を QLDB ドキュメントに保存することもできます。前述したすべての例で、返される各データ型が既知であり、null ではないことを前提にしています。次の例は、データ型が不明な場合や、null の可能性がある場合に Ion を操作する方法を示しています。

Java

```
// Empty variables
String exampleString = null;
Integer exampleInt = null;

// Assume ionValue is some queried data from QLDB
IonValue ionValue = null;

// Check the value type and assign it to the variable if it is not null
if (ionValue.getType() == IonType.STRING) {
    if (ionValue.isNullValue()) {
        exampleString = null;
    } else {
        exampleString = ((IonString)ionValue).stringValue();
    }
} else if (ionValue.getType() == IonType.INT) {
    if (ionValue.isNullValue()) {
        exampleInt = null;
```

```
    } else {
        exampleInt = ((IonInt)ionValue).intValue();
    }
};

// Creating null values
IonSystem ionSystem = IonSystemBuilder.standard().build();

// A null value still has an Ion type
IonString ionString;
if (exampleString == null) {
    // Specifically a null string
    ionString = ionSystem.newNullString();
} else {
    ionString = ionSystem.newString(exampleString);
}

IonInt ionInt;
if (exampleInt == null) {
    // Specifically a null int
    ionInt = ionSystem.newNullInt();
} else {
    ionInt = ionSystem.newInt(exampleInt);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
// The above null values still have the types 'String' and 'Int'
IonValue specialNull = ionSystem.newNull();
if (specialNull.getType() == IonType.NULL) {
    // This is true!
}
if (specialNull.isNullValue()) {
    // This is also true!
}
if (specialNull.getType() == IonType.STRING || specialNull.getType() == IonType.INT)
 {
    // This is false!
}
```

.NET

```
// Empty variables.
```

```
string exampleString = null;
int? exampleInt = null;

// Assume ionValue is some queried data from QLDB.
IIonValue ionValue;

if (ionValue.Type() == IonType.String)
{
    exampleString = ionValue.StringValue;
}
else if (ionValue.Type() == IonType.Int)
{
    if (ionValue.IsNull)
    {
        exampleInt = null;
    }
    else
    {
        exampleInt = ionValue.IntValue;
    }
};

// Creating null values.
IValueFactory valueFactory = new ValueFactory();

// A null value still has an Ion type.
IIonValue ionString = valueFactory.NewString(exampleString);

IIonValue ionInt;
if (exampleInt == null)
{
    // Specifically a null int.
    ionInt = valueFactory.NewNullInt();
}
else
{
    ionInt = valueFactory.NewInt(exampleInt.Value);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
IIonValue specialNull = valueFactory.NewNull();
if (specialNull.Type() == IonType.Null) {
    // This is true!
```

```
}
if (specialNull.IsNull) {
    // This is also true!
}
```

Go

マーシャリングの制限

Go では、マーシャリング後にアンマーシャリングしても、nil 値の型が保持されません。nil 値を Ion null にマーシャリングしても、アンマーシャリングした Ion null は、nil ではなく、ゼロ値になります。

```
ionNull, err := ion.MarshalText(nil) // ionNull is set to ion null
if err != nil {
    return
}

var result int
err = ion.Unmarshal(ionNull, &result) // result unmarshals to 0
if err != nil {
    return
}
```

Node.js

```
// Empty variables
let exampleString: string;
let exampleInt: number;

// Assume ionValue is some queried data from QLDB
// Check the value type and assign it to the variable if it is not null
if (ionValue.getType() === IonTypes.STRING) {
    if (ionValue.isNull()) {
        exampleString = null;
    } else {
        exampleString = ionValue.stringValue();
    }
} else if (ionValue.getType() === IonTypes.INT) {
    if (ionValue.isNull()) {
        exampleInt = null;
    } else {
        exampleInt = ionValue.numberValue();
```

```
    }
}

// Creating null values
if (exampleString === null) {
    ionString = dom.load('null.string');
} else {
    ionString = dom.load.of(exampleString);
}

if (exampleInt === null) {
    ionInt = dom.load('null.int');
} else {
    ionInt = dom.load.of(exampleInt);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
// The above null values still have the types 'String' and 'Int'
specialNull: dom.Value = dom.load("null.null");
if (specialNull.getType() === IonType.NULL) {
    // This is true!
}
if (specialNull.getType() === IonType.STRING || specialNull.getType() ===
 IonType.INT) {
    // This is false!
}
```

Python

```
# Empty variables
example_string = None
example_int = None

# Assume ion_value is some queried data from QLDB
# Check the value type and assign it to the variable if it is not null
if ion_value.ion_type == IonType.STRING:
    if isinstance(ion_value, IonPyNull):
        example_string = None
    else:
        example_string = ion_value
elif ion_value.ion_type == IonType.INT:
    if isinstance(ion_value, IonPyNull):
```

```
        example_int = None
    else:
        example_int = ion_value

# Creating Ion null values
if example_string is None:
    # Specifically a null string
    ion_string = loads("null.string")
else:
    # QLDB can take in Python string
    ion_string = example_string
if example_int is None:
    # Specifically a null int
    ion_int = loads("null.int")
else:
    # QLDB can take in Python int
    ion_int = example_int


# Special case regarding null!
# There is a generic null type which has Ion type 'Null'.
# The above null values still have the types 'String' and 'Int'
special_null = loads("null.null")
if special_null.ion_type == IonType.NULL:
    # This is true!
if special_null.ion_type == IonType.STRING or special_null.ion_type == IonType.INT:
    # This is false!
```

JSON へのダウンコンバート

アプリケーションに JSON の互換性が必要な場合は、Amazon Ion データを JSON にダウンコンバートできます。ただし、JSON にないリッチ Ion 型をデータで使用している特定のケースでは、Ion を JSON に変換するとデータが劣化する可能性があります。

Ion から JSON への変換ルールの詳細については、「Amazon Ion Cookbook」(Amazon Ion クックブック) の「Down-converting to JSON」(JSON へのダウンコンバート) を参照してください。

Amazon QLDB でのデータと履歴の使用

以下のトピックでは、作成、読み取り、更新、削除 (CRUD) ステートメントの基本的な例を紹介します。これらのステートメントは、QLDB コンソールのPartiQL エディタ、または QLDB シェルを使用して手動で実行することができます。また、このガイドでは、台帳に変更を加えたときに、QLDB によってどのようにデータが処理されるか、そのプロセスについても一通り説明します。

QLDB は PartiQL クエリ言語をサポートしています。

QLDB ドライバーを使用して同様のステートメントをプログラムで実行する方法を示すコード例については、「ドライバーの開始方法」のチュートリアルを参照してください。

> **Tip**
>
> 以下は、QLDB で PartiQL を使用するためのヒントとベストプラクティスの簡単な要約です。
>
> - 同時実行性とトランザクション制限を理解する - SELECT クエリを含むすべてのステートメントはオプティミスティック同時実行制御 (OCC、Optimistic Concurrency Control) 競合およびトランザクション制限 (30 秒のトランザクションタイムアウトなど) の対象になります。
> - インデックスの使用 - 高基数インデックスを使用し、ターゲットとなるクエリを実行して、ステートメントを最適化し、すべてのテーブルスキャンを回避します。詳細については、「クエリパフォーマンスの最適化」を参照してください。
> - 等価述語の使用 - インデックス付きルックアップには等価演算子 (= または IN) が必要です。不等価演算子 (<、>、LIKE、BETWEEN) はインデックス付きルックアップの対象にならず、すべてのテーブルスキャンが実行されます。
> - 内部結合のみを使用する - QLDB は現在、内部結合のみをサポートしています。ベストプラクティスとして、結合するテーブルごとにインデックス付けされたフィールドで結合します。結合基準と等価述語の両方に高基数インデックスを選択します。

トピック

- BY 句を使用したドキュメント ID のクエリの実行
- ドキュメントの更新と削除
- リビジョン履歴のクエリの実行
- ドキュメントのリビジョンを秘匿化する
- クエリパフォーマンスの最適化
- PartiQL ステートメントの統計の取得
- システムカタログのクエリの実行
- テーブルの管理
- インデックスの管理
- Amazon QLDB で割り当てられる一意の ID

インデックスを持つテーブルの作成とドキュメントの挿入

Amazon QLDB 台帳を作成したら、最初に基本的な CREATE TABLE ステートメントを使用してテーブルを作成します。テーブルは QLDB ドキュメント で構成されます。これらは、Amazon Ion `struct` 形式のデータセットです。

トピック

- テーブルとインデックスの作成
- ドキュメントの挿入

テーブルとインデックスの作成

テーブルには名前空間のない単純な大文字と小文字が区別される名前を付けます。QLDB ではオープンコンテンツがサポートされており、スキーマは適用されないため、テーブル作成時に属性やデータ型を定義しません。

```
CREATE TABLE VehicleRegistration
```

```
CREATE TABLE Vehicle
```

`CREATE TABLE` ステートメントは、新しいテーブルのシステムによって割り当てられた ID を返します。QLDB では、システムによって割り当てられた ID はすべて汎用一意 ID (UUID、Universally Unique Identifier) であり、それぞれ Base62 エンコード文字列で表されます。

> **Note**
>
> オプションで、テーブルの作成中にテーブルリソースのタグを定義できます。この方法の詳細は、「作成時のテーブルのタグ付け」を参照してください。

また、テーブルにインデックスを作成して、クエリのパフォーマンスを最適化することもできます。

```
CREATE INDEX ON VehicleRegistration (VIN)
```

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

> **Important**
>
> ドキュメントを効率的に検索するには、インデックスが必要です。インデックスがないと、QLDB はドキュメントを読み取るときにテーブルスキャンを実行する必要があります。これにより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子 (= または IN) を使用する WHERE 述語句でステートメントを実行する必要があります。詳細については、「クエリパフォーマンスの最適化」を参照してください。

インデックスを作成する際には、以下の制約があることに注意してください。

- インデックスは 1 つのトップレベルフィールドでのみ作成できます。複合、ネスト、一意、および関数ベースのインデックスはサポートされていません。
- 任意の Ion データ型 (`list`、`struct` など) でインデックスを作成できます。ただし、Ion のデータ型に関わらず、Ion 値全体の等価によってインデックス付けされたルックアップのみを実行できます。例えば、`list` 型をインデックスとして使用すると、リスト内の 1 つの項目でインデックス付けされたルックアップは実行できません。
- クエリのパフォーマンスは、等価述語 (`WHERE indexedField = 123`、`WHERE indexedField IN (456, 789)` など) を使用する場合にのみ向上します。

QLDB では、クエリの述語で不等式はサポートされていません。そのため、範囲でフィルタリングされるスキャンは実装されていません。

- インデックス付きフィールドの名前は大文字と小文字の区別があり 128 文字以下で指定します。
- QLDB でのインデックス作成は非同期です。空でないテーブルでのインデックスの作成を完了するのにかかる時間は、テーブルサイズによって異なります。詳細については、「インデックスの管理」を参照してください。

ドキュメントの挿入

次に、テーブルにドキュメントを挿入できます。QLDB ドキュメントは Amazon Ion 形式で保存されます。次の PartiQL INSERT ステートメントには、Amazon QLDB コンソールの使用開始方法 で使用する車両登録のサンプルデータの一部が含まれます。

```
INSERT INTO VehicleRegistration
<< {
    'VIN' : '1N4AL11D75C109151',
    'LicensePlateNumber' : 'LEWISR261LL',
    'State' : 'WA',
    'City' : 'Seattle',
    'PendingPenaltyTicketAmount' : 90.25,
    'ValidFromDate' : `2017-08-21T`,
    'ValidToDate' : `2020-05-11T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId' : '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ { 'PersonId' : '5Ufgdlnj06gF5CWcOIu64s' } ]
    }
},
{
    'VIN' : 'KM8SRDHF6EU074761',
    'LicensePlateNumber' : 'CA762X',
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75,
    'ValidFromDate' : `2017-09-14T`,
    'ValidToDate' : `2020-06-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' },
        'SecondaryOwners' : []
    }
```

```
} >>
```

```
INSERT INTO Vehicle
<< {
    'VIN' : '1N4AL11D75C109151',
    'Type' : 'Sedan',
    'Year' : 2011,
    'Make' : 'Audi',
    'Model' : 'A5',
    'Color' : 'Silver'
} ,
{
    'VIN' : 'KM8SRDHF6EU074761',
    'Type' : 'Sedan',
    'Year' : 2015,
    'Make' : 'Tesla',
    'Model' : 'Model S',
    'Color' : 'Blue'
} >>
```

PartiQL 構文とセマンティクス

- フィールド名は一重引用符 (`'...'`) で囲みます。
- 文字列値も一重引用符 (`'...'`) で囲みます。
- タイムスタンプはバックティック (`` `...` ``) で囲みます。バックティックは、Ion リテラルを示すために使用できます。
- 整数と小数はリテラル値で、指定する必要はありません。

PartiQL の構文とセマンティクスの詳細については、「Amazon QLDB での PartiQL による Ion のクエリ」を参照してください。

INSERT ステートメントで、バージョン番号が 0 というドキュメントの最初のリビジョンが作成されます。各ドキュメントを一意に識別するために、QLDB によってドキュメント ID がメタデータの一部として割り当てられます。Insert ステートメントは、挿入された各ドキュメントの ID を返します。

> **Important**
>
> QLDB ではスキーマを適用しないため、何度でもテーブルに同じドキュメントを挿入できます。それぞれの挿入ステートメントでは、ジャーナルに個別にドキュメントのエントリを実行し、QLDB がそれぞれのドキュメントに一意の ID を割り当てます。

テーブルに挿入したドキュメントのクエリを実行する方法については、「データのクエリの実行」に進みます。

データのクエリの実行

ユーザービューは、自分のユーザーデータの中で最新の削除されていないリビジョンのみを返します。これが Amazon QLDB のデフォルトビューです。つまり、自分のデータのみにクエリを実行するときには、特別な限定詞は必要ありません。

次のクエリ例の構文とパラメータの詳細については、「Amazon QLDB PartiQL リファレンス」の「SELECT」を参照してください。

トピック

- 基本的なクエリ
- 射影とフィルタ
- Joins
- ネストされたデータ

基本的なクエリ

基本的な SELECT クエリでは、テーブルに挿入したドキュメントが返されます。

> **Warning**
>
> インデックス付きルックアップなしで QLDB でクエリを実行すると、完全なテーブルスキャンが呼び出されます。PartiQL は SQL 互換であるため、このようなクエリをサポートしています。ただし、QLDB の本番環境のユースケースではテーブルスキャンを実行しないでください。テーブルスキャンより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。

テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する必要があります (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`)。詳細については、「クエリパフォーマンスの最適化」を参照してください。

以下のクエリは、以前に インデックスを持つテーブルの作成とドキュメントの挿入 に挿入した車両登録ドキュメントの結果を示しています。結果の順序は一定ではなく、SELECT クエリごとに異なる場合があります。QLDB のクエリの結果の順序は変わる可能性があることに注意してください。

```
SELECT * FROM VehicleRegistration
WHERE LicensePlateNumber IN ('LEWISR261LL', 'CA762X')
```

```
{
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    PendingPenaltyTicketAmount: 90.25,
    ValidFromDate: 2017-08-21T,
    ValidToDate: 2020-05-11T,
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    }
},
{
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFromDate: 2017-09-14T,
    ValidToDate: 2020-06-25T,
    Owners: {
        PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
        SecondaryOwners: []
    }
}
```

```
SELECT * FROM Vehicle
```

```
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    VIN: "1N4AL11D75C109151",
    Type: "Sedan",
    Year: 2011,
    Make: "Audi",
    Model: "A5",
    Color: "Silver"
},
{
    VIN: "KM8SRDHF6EU074761",
    Type: "Sedan",
    Year: 2015,
    Make: "Tesla",
    Model: "Model S",
    Color: "Blue"
}
```

> **Important**
>
> PartiQL では、一重引用符を使用して、データ操作言語 (DML) またはクエリステートメントで文字列であることを示します。ただし、QLDB コンソールおよび QLDB シェルはクエリ結果を Amazon Ion テキスト形式で返すため、二重引用符で囲まれた文字列が表示されます。この構文により、PartiQL クエリ言語で SQL 互換性を維持し、Amazon Ion テキスト形式で JSON 互換性を維持できます。

射影とフィルタ

射影 (ターゲットの SELECT) およびその他の標準フィルタ (WHERE 句) を使用できます。次のクエリは、VehicleRegistration テーブルからドキュメントフィールドのサブセットを返します。以下の基準で車両をフィルタリングします。

- 文字列フィルタ: シアトルで登録されています。
- 10 進数フィルタ: 金額が 100.0 未満の支払い保留中の反則切符がある。
- 日付フィルタ: 登録日は 2019 年 9 月 4 日以降有効である。

```
SELECT r.VIN, r.PendingPenaltyTicketAmount, r.Owners
FROM VehicleRegistration AS r
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
AND r.City = 'Seattle' --string
AND r.PendingPenaltyTicketAmount < 100.0 --decimal
AND r.ValidToDate >= `2019-09-04T` --timestamp with day precision
```

```
{
    VIN: "1N4AL11D75C109151",
    PendingPenaltyTicketAmount: 90.25,
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    }
}
```

Joins

内部結合クエリを記述することもできます。以下の例では、登録車両の属性と共にすべての登録ドキュメントを返す暗黙的な内部結合クエリを示しています。

```
SELECT * FROM VehicleRegistration AS r, Vehicle AS v
WHERE r.VIN = v.VIN
AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    PendingPenaltyTicketAmount: 90.25,
    ValidFromDate: 2017-08-21T,
    ValidToDate: 2020-05-11T,
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    },
    Type: "Sedan",
    Year: 2011,
    Make: "Audi",
    Model: "A5",
```

```
    Color: "Silver"
},
{
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFromDate: 2017-09-14T,
    ValidToDate: 2020-06-25T,
    Owners: {
        PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
        SecondaryOwners: []
    },
    Type: "Sedan",
    Year: 2015,
    Make: "Tesla",
    Model: "Model S",
    Color: "Blue"
}
```

また、以下のように、明示的な構文で同じ内部結合クエリを記述することもできます。

```
SELECT * FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

ネストされたデータ

QLDB で PartiQL を使用すると、ドキュメント内のネストしたデータをクエリできます。以下の例では、ネストされたデータを平坦化する相関サブクエリを示しています。ここで @ 文字は構文的に省略可能です。ただしこの文字は、Owners という名前の別のコレクション (存在する場合) ではなく、VehicleRegistration 内の Owners 構造を必要とすることを明示的に示しています。

```
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    VIN: "1N4AL11D75C109151",
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
},
{
    VIN: "KM8SRDHF6EU074761",
    SecondaryOwners: []
}
```

以下に示しているのは、ネストされたデータを射影する SELECT リスト内のサブクエリと、内部結合を示しています。

```
SELECT
    v.Make,
    v.Model,
    (SELECT VALUE o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM
    VehicleRegistration AS r, Vehicle AS v
WHERE
    r.VIN = v.VIN AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    Make: "Audi",
    Model: "A5",
    PrimaryOwner: ["294jJ3YUoH1IEEm8GSabOs"]
},
{
    Make: "Tesla",
    Model: "Model S",
    PrimaryOwner: ["IN7MvYtUjkp1GMZu0F6CG9"]
}
```

以下では、VehicleRegistration ドキュメントについて、Owners.SecondaryOwners リスト内の各個人の PersonId とインデックス番号 (序数) を返します。

```
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = '1N4AL11D75C109151'
```

```
{
    PersonId: "5Ufgdlnj06gF5CWcOIu64s",
```

```
    owner_idx: 0
}
```

ドキュメントのメタデータのクエリを実行する方法については、「ドキュメントのメタデータのクエリの実行」に進みます。

ドキュメントのメタデータのクエリの実行

INSERT ステートメントで、バージョン番号が 0 というドキュメントの最初のリビジョンが作成されます。各ドキュメントを一意に識別するために、Amazon QLDB によってドキュメント ID がメタデータの一部として割り当てられます。

QLDB は、ドキュメント ID とバージョン番号に加えて、各ドキュメントの他のシステム生成メタデータをテーブルに保存します。このメタデータには、トランザクション情報、ジャーナル属性、およびドキュメントのハッシュ値が含まれます。

システムによって割り当てられた ID はすべて汎用一意 ID (UUID、Universally Unique Identifier) であり、それぞれ Base62 エンコード文字列で表されます。詳細については、「Amazon QLDB で割り当てられる一意の ID」を参照してください。

トピック

- コミット済みビュー
- コミットされたビューとユーザービューへの参加

コミット済みビュー

コミット済みビューのクエリを実行して、ドキュメントのメタデータにアクセスできます。このビューでは、システムで定義したテーブルからドキュメントを返します。このテーブルはユーザーテーブルに直接対応するものです。これには、自分のデータもシステムによって生成されたメタデータも、最後にコミットされ削除されていないリビジョンで含まれます。このビューのクエリを実行するには、クエリ内のテーブル名にプレフィックス _ql_committed_ を追加します。(プレフィックス _ql_ は、システムオブジェクトの QLDB に割り当てられます。)

```
SELECT * FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

このクエリの出力では、以前に インデックスを持つテーブルの作成とドキュメントの挿入 に挿入したデータを使用して、それぞれの削除されていないドキュメントの最新リビジョンのシステムコン

テンツを表示します。システムドキュメントには、metadata フィールドのネストされたメタデータ、data フィールドのネストされたユーザーデータがあります。

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14
    },
    hash:{{wCsmM6qD4STxz0WYmE+47nZvWtcCz9D6zNtCiM5GoWg=}},
    data:{
        VIN: "1N4AL11D75C109151",
        LicensePlateNumber: "LEWISR261LL",
        State: "WA",
        City: "Seattle",
        PendingPenaltyTicketAmount: 90.25,
        ValidFromDate: 2017-08-21T,
        ValidToDate: 2020-05-11T,
        Owners: {
            PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
            SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
        }
    },
    metadata:{
        id:"3Qv67yjXEwB9SjmvkuG6Cp",
        version:0,
        txTime:2019-06-05T20:53:321d-3Z,
        txId:"HgXAkLjAtV0HQ4lNYdzX60"
    }
},
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14
    },
    hash:{{wPuwH60TtcCvg/23BFp+redRXuCALkbDihkEvCX22Jk=}},
    data:{
        VIN: "KM8SRDHF6EU074761",
        LicensePlateNumber: "CA762X",
        State: "WA",
        City: "Kent",
        PendingPenaltyTicketAmount: 130.75,
        ValidFromDate: 2017-09-14T,
        ValidToDate: 2020-06-25T,
```

```
        Owners: {
            PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
            SecondaryOwners: []
        }
    },
    metadata:{
        id:"JOzfB3lWqGU727mpPeWyxg",
        version:0,
        txTime:2019-06-05T20:53:321d-3Z,
        txId:"HgXAkLjAtV0HQ4lNYdzX60"
    }
}
```

コミット済みビューフィールド

- blockAddress: ドキュメントリビジョンがコミットされた台帳のジャーナルのブロックの場所。暗号検証に使用するアドレスには、以下の 2 つのフィールドがあります。
  - strandId - ブロックを含むジャーナルストランドの一意の ID。
  - sequenceNo: ストランド内でブロックの場所を指定するインデックス番号。

  > **Note**
  >
  > この例にあるドキュメントにはどちらにも、sequenceNo が同じ blockAddress が含まれています。これらのドキュメントは、1 回のトランザクションで (この場合には、1 つのステートメントに) 挿入されたため、同じブロックにコミットされています。

- hash: ドキュメントリビジョンを一意に表す SHA-256 Ion ハッシュ値。ハッシュは data フィールドと metadata フィールドを対象とし、暗号検証に使用できます。
- data – ドキュメントのユーザーデータ属性。

  リビジョンを秘匿化すると、この data 構造は dataHash フィールドに置き換えられ、その値は削除された data 構造の Ion ハッシュになります。
- metadata: ドキュメントのメタデータ属性。
  - id: ドキュメントのシステムによって割り当てられた一意の ID。
  - version – ドキュメントのバージョン番号。これは、ドキュメントリビジョンごとに増えていく、0 から始まる整数です。
  - txTime: ジャーナルにドキュメントリビジョンがコミットされたときのタイムスタンプ。
  - txId: ドキュメントリビジョンをコミットしたトランザクションの一意の ID。

コミットされたビューとユーザービューへの参加

コミットされたビュー内のテーブルをユーザービューのテーブルと結合するクエリを記述できます。例えば、あるテーブルのドキュメント id を別のテーブルのユーザー定義フィールドと結合することができます。

次のクエリは、PersonId の DriversLicense と Person という名前の 2 つのテーブルをドキュメント id のフィールドとそれぞれ結合します。後者にはコミット済みビューを使用します。

```
SELECT * FROM DriversLicense AS d INNER JOIN _ql_committed_Person AS p
ON d.PersonId = p.metadata.id
WHERE p.metadata.id = '1CWScY2qHYI9G88C2SjvtH'
```

デフォルトのユーザービューでドキュメント ID フィールドのクエリを実行する方法については、「BY 句を使用したドキュメント ID のクエリの実行」に進みます。

BY 句を使用したドキュメント ID のクエリの実行

一意の識別子 (車両の VIN など) 用にフィールドを定義することはできますが、実際のドキュメントの一意の識別子は、「ドキュメントの挿入」で説明されているとおり、id メタデータのフィールドになります。このため、id フィールドを使用して、各テーブル間の関係を作成できます。

ドキュメントの id フィールドは、コミット済みビューでのみ直接アクセスできますが、BY 句を使用すれば、デフォルトのユーザービューに射影することもできます。例として、以下のクエリとその結果を参照してください。

```
SELECT r_id, r.VIN, r.LicensePlateNumber, r.State, r.City, r.Owners
FROM VehicleRegistration AS r BY r_id
WHERE r_id = '3Qv67yjXEwB9SjmvkuG6Cp'
```

```
{
    r_id: "3Qv67yjXEwB9SjmvkuG6Cp",
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
```

```
    }
}
```

このクエリで、r_id は、BY キーワードを使用して FROM 句で宣言されているユーザー定義のエイリアスです。この r_id エイリアスは、クエリの結果セットで各ドキュメントの id メタデータフィールドにバインドされます。SELECT 句とユーザービューのクエリの WHERE 句にこのエイリアスを使用できます。

ただし、他のメタデータの属性にアクセスするには、コミット済みビューでクエリを実行する必要があります。

ドキュメント ID での結合

あるテーブルのドキュメント id を別のテーブルのユーザー定義フィールド内の外部キーとして使用しているとします。BY 句を使用すると、これらのフィールドに 2 つのテーブルの内部結合クエリを記述できます (前のトピックの「コミットされたビューとユーザービューへの参加」と同様)。

次の例は、DriversLicense の Person と PersonId という名前の 2 つのテーブルをドキュメント id のフィールドとそれぞれ結合します。後者には BY 句を使用します。

```
SELECT * FROM DriversLicense AS d INNER JOIN Person AS p BY pid
ON d.PersonId = pid
WHERE pid = '1CWScY2qHYI9G88C2SjvtH'
```

テーブル内のドキュメントを変更する方法については、「ドキュメントの更新と削除」に進みます。

ドキュメントの更新と削除

Amazon QLDB のドキュメントリビジョンは、一意のドキュメント ID によって識別される一連のドキュメントの 1 つのバージョンを表す Amazon Ion 構造です。すべてのリビジョンには、ユーザーデータとシステム生成メタデータの両方を含む、ドキュメントの完全なデータセットが含まれます。各リビジョンは、ドキュメント ID と 0 から始まるバージョン番号を組み合わせてそれぞれ識別されます。

ドキュメントを更新すると、QLDB によって、同じドキュメント ID と増加したバージョン番号を持つ新しいリビジョンが作成されます。ドキュメントのライフサイクルは、テーブルから削除されたときに終了します。つまり、同じドキュメント ID のドキュメントリビジョンを再度作成することはできません。

ドキュメントのリビジョン

たとえば、次のステートメントに新しい車両登録を挿入し、登録都市をアップデートしてから、登録を削除します。その結果、3 つのドキュメントリビジョンが出来上がります。

```
INSERT INTO VehicleRegistration
{
    'VIN' : '1HVBBAANXWH544237',
    'LicensePlateNumber' : 'LS477D',
    'State' : 'WA',
    'City' : 'Tacoma',
    'PendingPenaltyTicketAmount' : 42.20,
    'ValidFromDate' : `2011-10-26T`,
    'ValidToDate' : `2023-09-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': 'KmA3XPKKFqYCP2zhR3d0Ho' },
        'SecondaryOwners' : []
    }
}
```

> **Note**
>
> Insert ステートメントと他の DML ステートメントは、影響を受ける各ドキュメントの ID を返します。次のトピックの履歴機能に必要なため、続行する前に、この ID を保存してください。次のクエリを使用してドキュメント ID を検索することもできます。
>
> ```
> SELECT r_id FROM VehicleRegistration AS r BY r_id
> WHERE r.VIN = '1HVBBAANXWH544237'
> ```

```
UPDATE VehicleRegistration AS r
SET r.City = 'Bellevue'
WHERE r.VIN = '1HVBBAANXWH544237'
```

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

これらの DML ステートメントの構文の例と詳細については、「Amazon QLDB PartiQL リファレンス」の「UPDATE」と「DELETE」を参照してください。

ドキュメント内の特定の要素を挿入および削除するには、FROM キーワードで始まる UPDATE ステートメントまたはその他の DML ステートメントを使用できます。詳細と例については、「FROM (INSERT、REMOVE、または SET)リファレンス」を参照してください。

ドキュメントを削除した後は、コミット済みビューまたはユーザービューでそのドキュメントのクエリを実行できなくなります。組み込みの履歴関数を使用してこのドキュメントのリビジョン履歴のクエリを実行する方法については、「リビジョン履歴のクエリの実行」に進みます。

リビジョン履歴のクエリの実行

Amazon QLDB は、テーブル内にあるすべてのドキュメントの履歴全体を保存します。組み込みの履歴関数のクエリを実行することで、「ドキュメントの更新と削除」で前に挿入、更新、削除された車両登録ドキュメントの 3 つのリビジョンすべてを表示できます。

トピック

- 履歴関数
- 履歴クエリの例

履歴関数

QLDB の履歴関数は、テーブルのシステム定義ビューからリビジョンを返す PartiQL の拡張機能です。そのため、お客様のデータも、コミット済みビューと同じスキーマにあるその関連メタデータも両方含まれます。

[Syntax] (構文)

```
SELECT * FROM history( table_name | 'table_id' [, `start-time` [, `end-time` ] ] ) AS h
[ WHERE h.metadata.id = 'id' ]
```

引数

table_name | '***table_id***'

テーブル名またはテーブル ID のいずれか。テーブル名は PartiQL 識別子であり、二重引用符で囲んでも、引用符で囲まなくてもかまいません。テーブル ID は文字列リテラルであり、単一引用符で囲む必要があります。テーブル ID の使用の詳細については、「非アクティブなテーブルの履歴に対するクエリの実行」を参照してください。

`start-time`、`end-time`

(オプション) リビジョンがアクティブだった時間範囲を指定します。これらのパラメータでは、リビジョンがトランザクションのジャーナルにコミットされた時間範囲を指定しません。

開始時刻と終了時刻は、バックティック (`` `...` ``) で示すことができる Ion タイムスタンプリテラルです。詳細については、「Amazon QLDB での PartiQL による Ion のクエリ」を参照してください。

これらの時間パラメータには、次の動作があります。

- 開始時刻と終了時刻は、両方とも含まれます。両方とも、ISO 8601 の日時形式、協定世界時 (UTC) にしてください。
- 開始時刻は、過去の任意の日付で 終了時刻以前の時刻に設定してください。
- 終了時刻は、協定世界時 (UTC) における現在の日時よりも前の時刻にしてください。
- 開始時刻を指定し、終了時刻を指定しない場合、クエリではデフォルトで終了時刻が現在の日付と時刻に設定されます。どちらも指定しない場合、クエリでは履歴全体が返されます。

'id'

(オプション) リビジョン履歴のクエリを実行するドキュメント ID。一重引用符を使用して表記されます。

> **Tip**
>
> ベストプラクティスとして、履歴クエリは日付範囲 (start-time および end-time) とドキュメント ID (`metadata.id`) の両方で修飾します。QLDB では、すべての `SELECT` クエリはトランザクションで処理され、トランザクションタイムアウト制限の対象になります。
> 履歴クエリでは、テーブルで作成するインデックスは使用されません。QLDB 履歴はドキュメント ID によってのみインデックス付けされるため、現時点では追加の履歴インデックスを作成することはできません。開始時刻と終了時刻を含む履歴クエリでは、日付範囲修飾のメリットが得られます。

履歴クエリの例

車両登録ドキュメントの履歴のクエリを実行するために、「ドキュメントの更新と削除」で保存しておいた `id` を使用します。例えば、次の履歴クエリは、`2019-06-05T00:00:00Z` と

2019-06-05T23:59:59Z の間アクティブだったドキュメント ID ADR2Ll1fGsU4Jr4EqTdnQF のリビジョンを返します。

> **Note**
>
> 開始時刻と終了時刻のパラメータでは、リビジョンがトランザクション内のジャーナルにコミットされた時間範囲を指定しないことに注意してください。例えば、リビジョンが 2019-06-05T00:00:00Z 以前にコミットされ、その開始時刻を過ぎてもアクティブなままだった場合、この例のクエリは結果にそのリビジョンを返します。

必ず、id、開始時刻、終了時刻を適した独自の値に置き換えてください。

```
SELECT * FROM history(VehicleRegistration, `2019-06-05T00:00:00Z`,
 `2019-06-05T23:59:59Z`) AS h
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF' --replace with your id
```

クエリ結果は以下のようになります。

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14
    },
    hash:{{B2wYwrHKOWsmIBmxUgPRrTx9lv36tMlod2xVvWNiTbo=}},
    data: {
        VIN: "1HVBBAANXWH544237",
        LicensePlateNumber: "LS477D",
        State: "WA",
        City: "Tacoma",
        PendingPenaltyTicketAmount: 42.20,
        ValidFromDate: 2011-10-26T,
        ValidToDate: 2023-09-25T,
        Owners: {
            PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
            SecondaryOwners: []
        }
    },
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:0,
        txTime:2019-06-05T20:53:321d-3Z,
```

```
        txId:"HgXAkLjAtV0HQ4lNYdzX60"
    }
},
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:17
    },
    hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuMOmFCj8pEd6Mp0=}},
    data: {
        VIN: "1HVBBAANXWH544237",
        LicensePlateNumber: "LS477D",
        State: "WA",
        PendingPenaltyTicketAmount: 42.20,
        ValidFromDate: 2011-10-26T,
        ValidToDate: 2023-09-25T,
        Owners: {
            PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
            SecondaryOwners: []
        },
        City: "Bellevue"
    },
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:1,
        txTime:2019-06-05T21:01:442d-3Z,
        txId:"9cArhIQV5xf5Tf5vtsPwPq"
    }
},
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:19
    },
    hash:{{7bm5DUwpqJFGrmZpb7h9wAxtvggYLPcXq+LAobi9fDg=}},
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:2,
        txTime:2019-06-05T21:03:76d-3Z,
        txId:"9GslbtDtpVHAgYghR5FXbZ"
    }
}
```

出力には、メタデータの属性が含まれ、それぞれのアイテムが変更された日時、また関連するトランザクション内容の詳細を確認できます。このデータで、以下を確認できます。

- ドキュメントはシステムによって割り当てられた id: ADR2Ll1fGsU4Jr4EqTdnQF で個々に識別されます。これは Base62 でエンコードされた文字列で表される UUID です。
- INSERT ステートメントで、ドキュメントの最初のリビジョン (バージョン0) を作成します。
- その後、更新のたびに新しいリビジョンが作成され、同じドキュメント id にバージョン番号が増えていきます。
- txId フィールドは各リビジョンをコミットしたトランザクションを示し、txTime は各リビジョンがコミットされた日時を示します。
- DELETE ステートメントでは新しいリビジョンが作成されますが、これはドキュメントの最後のリビジョンになります。この最終的なリビジョンには、メタデータのみが含まれます。

リビジョンを完全に削除する方法については、ドキュメントのリビジョンを秘匿化する に進んでください。

ドキュメントのリビジョンを秘匿化する

Amazon QLDB では、DELETE ステートメントは、削除済みとしてマークする新しいリビジョンを作成することによってのみドキュメントを論理的に削除します。QLDB テーブルの履歴にある使用頻度の低いドキュメントリビジョンを完全に削除できるデータの秘匿化オペレーションもサポートしています。

> **Note**
>
> 2021 年 7 月 22 日より前に作成された台帳は、現時点では秘匿化の対象にはなりません。台帳の作成時間は Amazon QLDB コンソールで確認できます。

秘匿化オペレーションでは、指定されたリビジョンのユーザーデータのみが削除され、ジャーナルシーケンスとドキュメントメタデータは変更されません。これにより、台帳の全体的なデータの整合性が維持されます。

QLDB でのデータの秘匿化を開始する前に、「Amazon QLDB PartiQL リファレンス」の「秘匿化に関する考慮事項と制約事項」を必ず確認してください。

トピック

秘匿化ストアドプロシージャ

REDACT_REVISION ストアドプロシージャを使用して、台帳内の使用頻度の低い個別のリビジョンを完全に削除できます。このストアドプロシージャは、インデックス付きストレージとジャーナルストレージの両方で、指定されたリビジョンのユーザーデータをすべて削除します。ただし、ジャーナルシーケンスと、ドキュメント ID やハッシュなどのドキュメントメタデータは変更されません。この操作を元に戻すことはできません。

指定されたドキュメントリビジョンは、履歴上使用頻度の低いリビジョンでなければなりません。ドキュメントの最新の有効なリビジョンは、秘匿化の対象にはなりません。

複数のリビジョンを秘匿化するには、リビジョンごとに 1 回ストアドプロシージャを実行する必要があります。トランザクションごとに 1 つのリビジョンを秘匿化できます。

[Syntax] (構文)

```
EXEC REDACT_REVISION `block-address`, 'table-id', 'document-id'
```

引数

`` `block-address` ``

秘匿化されたドキュメントリビジョンのジャーナルブロックの場所。アドレスは、strandId と sequenceNo という 2 つのフィールドを含む Amazon Ion 構造です。

これはバックスティックで示される Ion リテラル値です。例:

```
`{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`
```

`'table-id'`

一重引用符で囲まれた、秘匿化するドキュメントのリビジョンがあったテーブルの一意の ID。

'*document-id*'

一重引用符で囲まれた、秘匿化対象のリビジョンの一意のドキュメント ID。

秘匿化が完了したかどうかの確認

このストアドプロシージャを実行して秘匿化リクエストを送信すると、QLDB はデータの秘匿化を非同期的に処理します。完了すると、リビジョン内の (data 構造によって表される) ユーザーデータは完全に削除されます。秘匿化リクエストが完了したかどうかを確認するには、次のいずれかを使用します。

- ジャーナルエクスポート
- ジャーナルストリーム
- ゲットブロック API オペレーション
- ゲットリビジョン API オペレーション
- 履歴関数 – 注: ジャーナルで秘匿化が完了した後、履歴クエリに秘匿化の結果が表示されるまでに時間がかかることがあります。非同期の秘匿化が完了すると、一部のリビジョンが他のリビジョンよりも先に秘匿化されるかもしれませんが、履歴クエリでは最終的に完了した結果が表示されます。

リビジョンの秘匿化が完了すると、リビジョンの data 構造は新しい dataHash フィールドに置き換えられます。このフィールドの値は、次の例に示されるように、削除された data 構造の Ion ハッシュです。その結果、台帳は全体的なデータ整合性を維持し、既存の検証 API オペレーションを通じて暗号的に検証できる状態を維持します。検証の詳細については、「Amazon QLDB でのデータ検証」を参照してください。

秘匿化の例

以前に「リビジョン履歴のクエリの実行」で確認した車両登録書類を考えてみましょう。2 番目のリビジョン (version:1) を秘匿化するとします。次のクエリ例は、秘匿化前のリビジョンを示しています。クエリ結果では、秘匿化される data 構造が*#####*で強調表示されます。

```
SELECT * FROM history(VehicleRegistration) AS h
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF' --replace with your id
AND h.metadata.version = 1
```

```
{
```

```
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:17
    },
    hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuMOmFCj8pEd6Mp0=}},
    data: {
        VIN: "1HVBBAANXWH544237",
        LicensePlateNumber: "LS477D",
        State: "WA",
        PendingPenaltyTicketAmount: 42.20,
        ValidFromDate: 2011-10-26T,
        ValidToDate: 2023-09-25T,
        Owners: {
            PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
            SecondaryOwners: []
        },
        City: "Bellevue"
    },
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:1,
        txTime:2019-06-05T21:01:442d-3Z,
        txId:"9cArhIQV5xf5Tf5vtsPwPq"
    }
}
```

この値は REDACT_REVISION ストアドプロシージャに渡す必要があるため、クエリ結果の blockAddress に注意してください。次に、次のようにシステムカタログをクエリして、VehicleRegistration テーブルの一意の ID を見つけます。

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

このテーブル ID をドキュメント ID とブロックアドレスとともに使用して REDACT_REVISION を実行します。テーブル ID とドキュメント ID は一重引用符で囲む必要がある文字列リテラルで、ブロックアドレスはバックティックスで囲まれた Ion リテラルです。必要に応じて、これらの引数を独自の値に置き換えてください。

```
EXEC REDACT_REVISION `{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`,
 '5PLf9SXwndd63lPaSIa0O6', 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

> **Tip**
>
> QLDB コンソールまたは QLDB シェルでテーブル ID またはドキュメント ID (または任意の文字列リテラル値) をクエリすると、戻り値は二重引用符で囲まれます。ただし、REDACT_REVISION ストアドプロシージャのテーブル ID 引数とドキュメント ID 引数を指定する場合は、値を一重引用符で囲む必要があります。
> これは、ステートメントを PartiQL 形式で記述しても、QLDB は Amazon Ion 形式で結果を返すからです。QLDB での PartiQL の構文とセマンティクスの詳細については、「PartiQL での Ion のクエリ」を参照してください。

有効な秘匿化リクエストでは、秘匿化中のドキュメントリビジョンを表す Ion 構造体が次のように返されます。

```
{
  blockAddress: {
    strandId: "JdxjkR9bSYB5jMHWcI464T",
    sequenceNo: 17
  },
  tableId: "5PLf9SXwndd63lPaSIa0O6",
  documentId: "ADR2Ll1fGsU4Jr4EqTdnQF",
  version: 1
}
```

このストアドプロシージャを実行すると、QLDB は秘匿化リクエストを非同期的に処理します。秘匿化が完了すると、data 構造は完全に削除され、新しい *dataHash* フィールドに置き換えられます。このフィールドの値は、次のとおり、削除された data 構造の Ion ハッシュです。

> **Note**
>
> この dataHash の例は情報提供のみを目的として、実際に計算されたハッシュ値ではありません。

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:17
    },
```

```
    hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuMOmFCj8pEd6Mp0=}},
    dataHash: {{s83jd7sfhsdfhksj7hskjdfjfpIPP/DP2hvionas2d4=}},
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:1,
        txTime:2019-06-05T21:01:442d-3Z,
        txId:"9cArhIQV5xf5Tf5vtsPwPq"
    }
}
```

アクティブなリビジョンの削除と秘匿化

アクティブなドキュメントのリビジョン (つまり、各ドキュメントの削除されていない最新のリビジョン) は、データの秘匿化の対象にはなりません。アクティブなリビジョンを秘匿化するには、まずそのリビジョンを更新または削除する必要があります。これにより、以前にアクティブだったリビジョンが履歴に移動し、秘匿化が可能になります。

ユースケースでドキュメント全体を削除済みとしてマークする必要がある場合は、まず DELETE ステートメントを使用します。例えば、次のステートメントは VIN が 1HVBBAANXWH544237 の VehicleRegistration ドキュメントを論理的に削除します。

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

次に、前述のように、削除前の前のリビジョンを秘匿化します。必要であれば、以前のリビジョンを個別に秘匿化することもできます。

ユースケースでドキュメントをアクティブのままにしておく必要がある場合は、まず UPDATE または FROM ステートメントを使用して、秘匿化するフィールドを隠したり削除したりします。このプロセスの説明は、以下のセクションに記載されています。

リビジョン内の特定のフィールドを秘匿化する

QLDB は、ドキュメントリビジョン内の特定のフィールドの秘匿化をサポートしていません。そのためには、まず UPDATE-REMOVE ステートメントまたは FROM-REMOVE ステートメントを使用して、既存のフィールドをリビジョンから削除します。例えば、次のステートメントは VIN が 1HVBBAANXWH544237 の LicensePlateNumber フィールドを VehicleRegistration ドキュメントから削除します。

```
UPDATE VehicleRegistration AS r
```

```
REMOVE r.LicensePlateNumber
WHERE r.VIN = '1HVBBAANXWH544237'
```

次に、前述のように、削除の前に以前のリビジョンを秘匿化します。必要であれば、この削除済みフィールドを含む以前のリビジョンを個別に秘匿化することもできます。

クエリを最適化する方法については、「クエリパフォーマンスの最適化」に進みます。

クエリパフォーマンスの最適化

Amazon QLDB は、高性能なオンライントランザクション処理 (OLTP) ワークロードのニーズに対応することを目的としています。つまり、SQL のようなクエリ機能をサポートしているにもかかわらず、QLDB は特定のクエリパターンのセットに最適化されます。これらのクエリパターンを操作するには、アプリケーションとそのデータモデルを設計することが重要です。それ以外の場合は、テーブルが大きくなるにつれて、クエリのレイテンシー、トランザクションのタイムアウト、同時実行の競合など、重大なパフォーマンスの問題が発生します。

このセクションでは、QLDB のクエリ制約を説明し、これらの制約を考慮した最適なクエリを記述するためのガイダンスを提供します。

トピック

- トランザクションタイムアウト制限
- 同時実行の競合
- 最適なクエリパターン
- 回避すべきクエリパターン
- パフォーマンスのモニタリング

トランザクションタイムアウト制限

QLDB では、すべての PartiQL ステートメント (すべての SELECT クエリを含む) はトランザクションで処理され、トランザクションタイムアウト制限の対象になります。トランザクションは、コミットされるまでに最大 30 秒間実行できます。この制限を超えると、QLDB はトランザクションに対して行われたすべての作業を拒否し、トランザクションを実行するセッションを破棄します。この制限は、サービスのクライアントがトランザクションを開始し、トランザクションをコミットまたはキャンセルしないことで、セッションがリークするのを防ぎます。

同時実行の競合

QLDB では、オプティミスティック同時実行制御 (OCC) を使用して同時実行制御が実行されます。最適でないクエリは、OCC の競合が増える可能性もあります。OCC の詳細については、「Amazon QLDB 同時実行モデル」を参照してください。

最適なクエリパターン

ベストプラクティスとして、インデックス付きフィールドまたはドキュメント ID でフィルタリングする WHERE 述語句を使用してステートメントを実行することをお勧めします。QLDB では、ドキュメントを効率的に検索するために、インデックス付きフィールドに等価演算子 (= または IN) が必要です。

以下に、ユーザービューの最適なクエリパターンの例を示します。

```
--Indexed field (VIN) lookup using the = operator
SELECT * FROM VehicleRegistration
WHERE VIN = '1N4AL11D75C109151'

--Indexed field (VIN) AND non-indexed field (City) lookup
SELECT * FROM VehicleRegistration
WHERE VIN = '1N4AL11D75C109151' AND City = 'Seattle'

--Indexed field (VIN) lookup using the IN operator
SELECT * FROM VehicleRegistration
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')

--Document ID (r_id) lookup using the BY clause
SELECT * FROM VehicleRegistration BY r_id
WHERE r_id = '3Qv67yjXEwB9SjmvkuG6Cp'
```

これらのパターンに従わないクエリはテーブル全体のスキャンを呼び出します。テーブルスキャンは、大きなテーブルのクエリや大きな結果セットを返すクエリに対してトランザクションタイムアウトを引き起こす可能性があります。さらに、競合するトランザクションと OCC の競合につながる可能性もあります。

高基数インデックス

高基数値を含むフィールドのインデックスを作成することをお勧めします。例えば、VehicleRegistration テーブル内の VIN と LicensePlateNumber のフィールドは、一意にすることを意図したインデックス付きフィールドです。

ステータスコード、住所の都道府県、郵便番号などの低基数フィールドのインデックス作成は避けてください。このようなフィールドのインデックスを作成すると、クエリによって、トランザクションのタイムアウトや、意図しない OCC の競合が発生する可能性が高くなる大きな結果セットが生成される可能性があります。

コミット済みビュークエリ

コミット済みビューで実行するクエリは、ユーザービュークエリと同じ最適化ガイドラインに従います。テーブル上に作成するインデックスは、コミット済みビューのクエリにも使用されます。

履歴関数クエリ

履歴関数クエリでは、テーブルで作成するインデックスは使用されません。QLDB 履歴はドキュメント ID によってのみインデックス付けされるため、現時点では追加の履歴インデックスを作成することはできません。

ベストプラクティスとして、履歴クエリを日付範囲 (開始時刻および終了時刻) とドキュメント ID (`metadata.id`) の両方で修飾します。開始時刻と終了時刻を含む履歴クエリでは、日付範囲修飾のメリットが得られます。

内部結合クエリ

内部結合クエリでは、少なくとも結合の右側にあるテーブルのインデックス付きフィールドを含む結合条件を使用します。結合インデックスがない場合、結合クエリは複数のテーブルスキャンを呼び出します。結合の左側のテーブルのすべてのドキュメントに対して、クエリは右側のテーブルを完全にスキャンします。ベストプラクティスは、少なくとも 1 つのテーブルに `WHERE` 等価述語を指定することに加えて、結合している各テーブルにインデックス付けされているフィールドに結合することです。

例えば、次のクエリでは、`VehicleRegistration` と `Vehicle` のテーブルをそれぞれの `VIN` フィールドに結合します。その両方にインデックスが付けられます。このクエリには、`VehicleRegistration.VIN` の等価述語もあります。

```
SELECT * FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

結合クエリの結合基準と等価述語の両方に高基数インデックスを選択します。

回避すべきクエリパターン

次に、QLDB の大きいテーブルでは適切に拡張されない最適ではないステートメントの例をいくつか示します。クエリは最終的にトランザクションタイムアウトになるため、時間の経過とともに増加するテーブルについては、これらのタイプのクエリに依存しないことを強くお勧めします。テーブルにはサイズの異なるドキュメントが含まれているため、インデックス付けされていないクエリに対して正確な制限を定義することは困難です。

```
--No predicate clause
SELECT * FROM Vehicle

--COUNT() is not an optimized function
SELECT COUNT(*) FROM Vehicle

--Low-cardinality predicate
SELECT * FROM Vehicle WHERE Color = 'Silver'

--Inequality (>) does not qualify for indexed lookup
SELECT * FROM Vehicle WHERE "Year" > 2019

--Inequality (LIKE)
SELECT * FROM Vehicle WHERE VIN LIKE '1N4AL%'

--Inequality (BETWEEN)
SELECT SUM(PendingPenaltyTicketAmount) FROM VehicleRegistration
WHERE ValidToDate BETWEEN `2020-01-01T` AND `2020-07-01T`

--No predicate clause
DELETE FROM Vehicle

--No document id, and no date range for the history() function
SELECT * FROM history(Vehicle)
```

一般的に、QLDB の本番ユースケースに、次のタイプのクエリパターンを実行することはお勧めしません。

- オンライン分析処理 (OLAP) クエリ
- 述語句のない探索的クエリ
- レポート作成クエリ
- テキスト検索

代わりに、分析ユースケースに、最適化された目的別データベースサービスへのデータのストリーミングをすることをお勧めします。例えば、QLDB データを Amazon OpenSearch Service にストリーム配信すると、ドキュメントに対して全文検索機能を提供することができます。このユースケースを示すサンプルアプリケーションについては、GitHub リポジトリ aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python を参照してください。QLDB ストリーミングの詳細については、「Amazon QLDB からのジャーナルデータのストリーミング」を参照してください。

パフォーマンスのモニタリング

QLDB ドライバーは、ステートメントの結果オブジェクトで消費された I/O 使用量とタイミング情報を提供します。これらのメトリクスを使用して、非効率な PartiQL ステートメントを特定できます。詳細については、「PartiQL ステートメントの統計の取得」を参照してください。

また、Amazon CloudWatch を使用して、データオペレーションに対する台帳のパフォーマンスを追跡することもできます。指定された `LedgerName` と `CommandType` に対する `CommandLatency` メトリクスをモニタリングします。詳細については、「Amazon によるモニタリング CloudWatch」を参照してください。QLDB がコマンドを使用してデータオペレーションを管理する方法については、「ドライバーによるセッション管理」を参照してください。

PartiQL ステートメントの統計の取得

Amazon QLDB は、より効率的な PartiQL ステートメントを実行して、QLDB の使用を最適化する際に役立つステートメント実行の統計を提供します。QLDB は、ステートメントの結果と共にこれらの統計を返します。これには、消費された I/O 使用量とサーバー側の処理時間を定量化するメトリクスが含まれており、非効率的なステートメントを特定するために使用できます。

この機能は現在、すべてのサポートされている言語において、QLDB コンソール、QLDB シェル、最新バージョンの QLDB ドライバーの PartiQL エディタで使用できます。コンソールでクエリ履歴のステートメント統計を表示することもできます。

トピック

- I/O 使用量
- タイミング情報

I/O 使用量

I/O 使用量のメトリクスは、読み取り I/O リクエストの数を示します。読み取り I/O リクエストの数が予想よりも多い場合は、インデックスがないなど、ステートメントが最適化されていないことを示します。前のトピック「クエリパフォーマンスの最適化」の「最適なクエリパターン」を確認することをお勧めします。

> **Note**
>
> 空でないテーブルで `CREATE INDEX` ステートメントを実行した場合、I/O 使用量メトリクスには、同期インデックス作成呼び出しの読み取りリクエストのみが含まれます。
> QLDB は、テーブル内の既存のドキュメントのインデックスを非同期的に構築します。これらの非同期読み取りリクエストは、ステートメント結果からの I/O 使用量メトリクスには含まれません。非同期読み取りリクエストは個別に課金され、インデックスの構築が完了した後に読み取り I/O の合計に追加されます。

QLDB コンソールの使用

QLDB コンソールを使用してステートメントの読み取り I/O 使用量を取得するには、次のステップを実行します。

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [PartiQL エディタ] を選択します。
3. 台帳のドロップダウンリストから台帳を選択します。
4. クエリエディタウィンドウで、選択したステートメントを入力し、[Run] (実行) を選択します。以下はクエリの例です。

   ```
   SELECT * FROM testTable WHERE firstName = 'Jim'
   ```

   ステートメントを実行するには、キーボードショートカット Ctrl+Enter (Windows の場合) または Cmd+Return (macOS の場合) を使用することもできます。キーボードショートカットの詳細については、「PartiQL エディタのキーボードショートカット」を参照してください。
5. クエリエディタウィンドウの下に、クエリ結果にはステートメントによって行われた読み取りリクエストの数である read I/Os が含まれます。

次のステップを実行して、クエリ履歴の読み取り I/O を表示することもできます。

1. ナビゲーションペインで、[PartiQL エディタ] の [最近のクエリ] を選択します。
2. [Read I/Os] (読み取り I/O) 列には、各ステートメントによって行われた読み取りリクエストの数が表示されます。

QLDB ドライバーの使用

QLDB ドライバーを使用してステートメントの I/O 使用量を取得するには、結果のストリームカーソルまたはバッファ済みカーソルの `getConsumedIOs` オペレーションを呼び出します。

次のコード例は、読み取り I/O をステートメント結果のストリームカーソルから取得する方法を示します。

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.IOUsage;
import software.amazon.qldb.Result;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

driver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);

    for (IonValue ionValue : result) {
        // User code here to handle results
    }

    IOUsage ioUsage = result.getConsumedIOs();
    long readIOs = ioUsage.getReadIOs();
});
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
```

```
// This is one way of creating Ion values. We can also use a ValueFactory.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/
driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionFirstName = IonLoader.Default.Load("Jim");

await driver.Execute(async txn =>
{
    IAsyncResult result = await txn.Execute("SELECT * FROM testTable WHERE firstName
 = ?", ionFirstName);

    // Iterate through stream cursor to accumulate read IOs.
    await foreach (IIonValue ionValue in result)
    {
        // User code here to handle results.
        // Warning: It is bad practice to rely on results within a lambda block,
 unless
        // it is to check the state of a result. This is because lambdas are
 retryable.
    }

    var ioUsage = result.GetConsumedIOs();
    var readIOs = ioUsage?.ReadIOs;
});
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

driver.Execute(context.Background(), func(txn qldbdriver.Transaction) (interface{},
 error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?", "Jim")
```

```
    if err != nil {
        panic(err)
    }

    for result.Next(txn) {
        // User code here to handle results
    }

    ioUsage := result.GetConsumedIOs()
    readIOs := *ioUsage.GetReadIOs()
    fmt.Println(readIOs)
    return nil,nil
})
```

Node.js

```
import { IOUsage, ResultReadable, TransactionExecutor } from "amazon-qldb-driver-
nodejs";

await driver.executeLambda(async (txn: TransactionExecutor) => {
    const result: ResultReadable = await txn.executeAndStreamResults("SELECT * FROM
 testTable WHERE firstName = ?", "Jim");

    for await (const chunk of result) {
        // User code here to handle results
    }

    const ioUsage: IOUsage = result.getConsumedIOs();
    const readIOs: number = ioUsage.getReadIOs();
});
```

Python

```
def get_read_ios(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM testTable WHERE
 firstName = ?", "Jim")

    for row in cursor:
        # User code here to handle results
        pass

    consumed_ios = cursor.get_consumed_ios()
```

```
    read_ios = consumed_ios.get('ReadIOs')

qldb_driver.execute_lambda(lambda txn: get_read_ios(txn))
```

次のコード例は、読み取り I/O をステートメント結果のバッファ済みカーソルから取得する方法を示します。これは、ExecuteStatement と FetchPage リクエストから読み取り I/O の合計を返します。

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.IOUsage;
import software.amazon.qldb.Result;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

Result result = driver.execute(txn -> {
    return txn.execute("SELECT * FROM testTable WHERE firstName = ?", ionFirstName);
});

IOUsage ioUsage = result.getConsumedIOs();
long readIOs = ioUsage.getReadIOs();
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);
});

var ioUsage = result.GetConsumedIOs();
```

```
var readIOs = ioUsage?.ReadIOs;
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err :=  txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 "Jim")
    if err != nil {
        return nil, err
    }
    return txn.BufferResult(result)
})

if err != nil {
    panic(err)
}

qldbResult := result.(*qldbdriver.BufferedResult)
ioUsage := qldbResult.GetConsumedIOs()
readIOs := *ioUsage.GetReadIOs()
fmt.Println(readIOs)
```

Node.js

```
import { IOUsage, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";

const result: Result = await driver.executeLambda(async (txn: TransactionExecutor)
 => {
    return await txn.execute("SELECT * FROM testTable WHERE firstName = ?", "Jim");
```

```
});

const ioUsage: IOUsage = result.getConsumedIOs();
const readIOs: number = ioUsage.getReadIOs();
```

Python

```
cursor = qldb_driver.execute_lambda(
    lambda txn: txn.execute_statement("SELECT * FROM testTable WHERE firstName = ?",
 "Jim"))

consumed_ios = cursor.get_consumed_ios()
read_ios = consumed_ios.get('ReadIOs')
```

> **Note**
>
> ストリームカーソルは、結果セットをページ分割するため、ステートフルです。したがって、getConsumedIOs と getTimingInformation オペレーションは、呼び出す時間以降の蓄積されたメトリクスを返します。
> バッファ済みカーソルは、結果セットをメモリにバッファリングし、蓄積されたメトリクスの合計を返します。

タイミング情報

タイミング情報メトリクスは、サーバー側の処理時間をミリ秒単位で表します。サーバー側の処理時間は、QLDB がステートメントの処理に費やす期間として定義されます。これには、ネットワーク呼び出しや一時停止にかかる時間は含まれません。このメトリクスは、QLDB サービス側の処理時間とクライアント側の処理時間を区別します。

QLDB コンソールの使用

QLDB コンソールを使用してステートメントのタイミング情報を取得するには、次のステップを実行します。

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [PartiQL エディタ] を選択します。
3. 台帳のドロップダウンリストから台帳を選択します。

4. クエリエディタウィンドウで、選択したステートメントを入力し、[Run] (実行) を選択します。以下はクエリの例です。

```
SELECT * FROM testTable WHERE firstName = 'Jim'
```

   ステートメントを実行するには、キーボードショートカット Ctrl+Enter (Windows の場合) または Cmd+Return (macOS の場合) を使用することもできます。キーボードショートカットの詳細については、「PartiQL エディタのキーボードショートカット」を参照してください。

5. クエリエディタウィンドウの下のクエリ結果には、QLDB がステートメントリクエストを受信してからレスポンスを送信するまでの時間であるサーバー側のレイテンシーが含まれます。これはクエリ時間全体のサブセットです。

次のステップを実行して、クエリ履歴のタイミング情報を表示することもできます。

1. ナビゲーションペインで、[PartiQL エディタ] の [最近のクエリ] を選択します。
2. [実行時間 (ms)] 列には、各ステートメントのこのタイミング情報が表示されます。

QLDB ドライバーの使用

QLDB ドライバーを使用してステートメントのタイミング情報を取得するには、結果のストリームカーソルまたはバッファ済みカーソルの `getTimingInformation` オペレーションを呼び出します。

次のコード例は、処理時間をステートメント結果のストリームカーソルから取得する方法を示します。

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.Result;
import software.amazon.qldb.TimingInformation;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

driver.execute(txn -> {
```

```
    Result result = txn.execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);

    for (IonValue ionValue : result) {
        // User code here to handle results
    }

    TimingInformation timingInformation = result.getTimingInformation();
    long processingTimeMilliseconds =
 timingInformation.getProcessingTimeMilliseconds();
});
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

await driver.Execute(async txn =>
{
    IAsyncResult result = await txn.Execute("SELECT * FROM testTable WHERE firstName
 = ?", ionFirstName);

    // Iterate through stream cursor to accumulate processing time.
    await foreach(IIonValue ionValue in result)
    {
        // User code here to handle results.
        // Warning: It is bad practice to rely on results within a lambda block,
 unless
        // it is to check the state of a result. This is because lambdas are
 retryable.
    }

    var timingInformation = result.GetTimingInformation();
    var processingTimeMilliseconds = timingInformation?.ProcessingTimeMilliseconds;
});
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

driver.Execute(context.Background(), func(txn qldbdriver.Transaction) (interface{},
 error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?", "Jim")

    if err != nil {
        panic(err)
    }

    for result.Next(txn) {
        // User code here to handle results
    }

    timingInformation := result.GetTimingInformation()
    processingTimeMilliseconds := *timingInformation.GetProcessingTimeMilliseconds()
    fmt.Println(processingTimeMilliseconds)
    return nil, nil
})
```

Node.js

```
import { ResultReadable, TimingInformation, TransactionExecutor } from "amazon-qldb-
driver-nodejs";

await driver.executeLambda(async (txn: TransactionExecutor) => {
    const result: ResultReadable = await txn.executeAndStreamResults("SELECT * FROM
 testTable WHERE firstName = ?", "Jim");

    for await (const chunk of result) {
```

```
        // User code here to handle results
    }

    const timingInformation: TimingInformation = result.getTimingInformation();
    const processingTimeMilliseconds: number =
 timingInformation.getProcessingTimeMilliseconds();
});
```

Python

```
def get_processing_time_milliseconds(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM testTable WHERE
 firstName = ?", "Jim")

    for row in cursor:
        # User code here to handle results
        pass

    timing_information = cursor.get_timing_information()
    processing_time_milliseconds =
 timing_information.get('ProcessingTimeMilliseconds')

qldb_driver.execute_lambda(lambda txn: get_processing_time_milliseconds(txn))
```

次のコード例は、処理時間をステートメント結果のバッファ済みカーソルから取得する方法を示します。これは、ExecuteStatement と FetchPage リクエストから処理時間の合計を返します。

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.Result;
import software.amazon.qldb.TimingInformation;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

Result result = driver.execute(txn -> {
    return txn.execute("SELECT * FROM testTable WHERE firstName = ?", ionFirstName);
});
```

```
TimingInformation timingInformation = result.getTimingInformation();
long processingTimeMilliseconds = timingInformation.getProcessingTimeMilliseconds();
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);
});

var timingInformation = result.GetTimingInformation();
var processingTimeMilliseconds = timingInformation?.ProcessingTimeMilliseconds;
```

> **Note**
>
> 同期コードに変換するには、await と async のキーワードを削除して、IAsyncResult 型を IResult に変更します。

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err :=  txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 "Jim")
    if err != nil {
        return nil, err
    }
```

```
    return txn.BufferResult(result)
})

if err != nil {
    panic(err)
}

qldbResult := result.(*qldbdriver.BufferedResult)
timingInformation := qldbResult.GetTimingInformation()
processingTimeMilliseconds := *timingInformation.GetProcessingTimeMilliseconds()
fmt.Println(processingTimeMilliseconds)
```

Node.js

```
import { Result, TimingInformation, TransactionExecutor } from "amazon-qldb-driver-
nodejs";

const result: Result = await driver.executeLambda(async (txn: TransactionExecutor)
 => {
    return await txn.execute("SELECT * FROM testTable WHERE firstName = ?", "Jim");
});

const timingInformation: TimingInformation = result.getTimingInformation();
const processingTimeMilliseconds: number =
 timingInformation.getProcessingTimeMilliseconds();
```

Python

```
cursor = qldb_driver.execute_lambda(
    lambda txn: txn.execute_statement("SELECT * FROM testTable WHERE firstName = ?",
 "Jim"))

timing_information = cursor.get_timing_information()
processing_time_milliseconds = timing_information.get('ProcessingTimeMilliseconds')
```

> **Note**
>
> ストリームカーソルは、結果セットをページ分割するため、ステートフルです。したがって、getConsumedIOs と getTimingInformation オペレーションは、呼び出す時間以降の蓄積されたメトリクスを返します。

バッファ済みカーソルは、結果セットをメモリにバッファリングし、蓄積されたメトリクスの合計を返します。

システムカタログをクエリする方法については、システムカタログのクエリの実行 に進みます。

システムカタログのクエリの実行

Amazon QLDB 台帳で作成する各テーブルには、システムによって割り当てられた一意の ID があります。システムカタログテーブル `information_schema.user_tables` のクエリを実行することで、テーブルの ID、インデックスのリスト、およびその他のメタデータを見つけることができます。

システムによって割り当てられた ID はすべて汎用一意 ID (UUID、Universally Unique Identifier) であり、それぞれ Base62 エンコード文字列で表されます。詳細については、「Amazon QLDB で割り当てられる一意の ID」を参照してください。

以下の例では、`VehicleRegistration` テーブルのメタデータ属性を返すクエリの結果を示しています。

```
SELECT * FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

```
{
    tableId: "5PLf9SXwndd63lPaSIa0O6",
    name: "VehicleRegistration",
    indexes: [
        { indexId: "Djg2nt0yIs2GY0T29Kud1z", expr: "[VIN]", status: "ONLINE" },
        { indexId: "4tPW3fUhaVhDinRgKRLhGU", expr: "[LicensePlateNumber]", status:
 "BUILDING" }
    ],
    status: "ACTIVE"
}
```

テーブルメタデータフィールド

- `tableId`: テーブルの一意の ID。
- `name`: テーブル名。

- indexes: テーブルのインデックスのリスト。
  - indexId: インデックスの一意の ID。
  - expr: インデックス付きドキュメントパス。このフィールドは、[fieldName] 形式の文字列です。
  - status: インデックスの現在のステータス (BUILDING、FINALIZING、ONLINE、FAILED、または DELETING)。QLDB は、ステータスが ONLINE になるまで、クエリのインデックスを使用しません。
  - message: インデックスに FAILED ステータスがある理由を説明するエラーメッセージ。このフィールドは、失敗したインデックスにのみ含まれます。
- status: テーブルの現在のステータス (ACTIVE または INACTIVE)。テーブルは、DROP すると INACTIVE になります。

DROP TABLE および UNDROP TABLE ステートメントを使用してテーブルを管理する方法については、テーブルの管理 に進みます。

テーブルの管理

このセクションでは、Amazon QLDB の DROP TABLE および UNDROP TABLE ステートメントを使用してテーブルを管理する方法について説明します。また、テーブルの作成中にテーブルにタグを付ける方法についても説明します。作成できるアクティブなテーブル数と合計テーブル数のクォータは、Amazon QLDB でのクォータと制限 で定義します。

トピック

- 作成時のテーブルのタグ付け
- テーブルの削除
- 非アクティブなテーブルの履歴に対するクエリの実行
- テーブルを再アクティブ化

作成時のテーブルのタグ付け

> **Note**
>
> 作成時のテーブルのタグ付けは、現在 STANDARD 許可モードの台帳のみでサポートされています。

テーブルリソースにタグ付けすることができます。既存のテーブルのタグを管理するには、AWS Management Console または API オペレーション TagResource、UntagResource、および ListTagsForResource を使用します。詳細については、「Amazon QLDB リソースのタグ付け」を参照してください。

テーブルの作成中に、QLDB コンソールを使用するか、CREATE TABLE PartiQL ステートメントで指定して、テーブルタグを定義することもできます。次の例では、Vehicle という名前のテーブルをタグ environment=production を付けて作成します。

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

作成時にリソースにタグ付けすることで、リソース作成後にカスタムタグ付けスクリプトを実行する必要がなくなります。テーブルにタグを付けると、それらのタグに基づいてテーブルへのアクセスを制御できます。例えば、特定のタグを持つテーブルにのみフルアクセスを付与できます。JSON ポリシーの例については、「テーブルタグに基づくすべてのアクションへのフルアクセス」を参照してください。

テーブルの削除

テーブルを削除するには、基本 DROP TABLE ステートメントを使用します。QLDB でのテーブルの削除は、テーブルを無効にしているだけです。

例えば、次のステートメントは、VehicleRegistration テーブルを非アクティブにします。

```
DROP TABLE VehicleRegistration
```

DROP TABLE ステートメントは、テーブルのシステムで割り当てられた ID を返します。VehicleRegistration のステータスは、システムカタログテーブル information_schema.user_tables で INACTIVE になっています。

```
SELECT status FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

非アクティブなテーブルの履歴に対するクエリの実行

最初の入力引数として、テーブル名ではなくテーブル ID を渡しても、QLDB 履歴関数 のクエリを実行できます。非アクティブなテーブルの履歴をクエリするには、テーブル ID を使用する必要があり

ます。テーブルが非アクティブ化された後は、テーブル名ではその履歴に対してクエリを実行できなくなります。

まず、システムカタログテーブルをクエリして、テーブル ID を見つけます。たとえば、次のクエリは VehicleRegistration テーブルの tableId を返します。

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

次に、この ID を使用して、リビジョン履歴のクエリの実行 から同じ履歴クエリを実行できます。以下の例では、テーブル ID 5PLf9SXwndd63lPaSIa0O6、ドキュメント ID ADR2Ll1fGsU4Jr4EqTdnQF の履歴に対してクエリを実行しています。テーブル ID は文字列リテラルであり、単一引用符で囲む必要があります。

```
--replace both the table and document IDs with your values
SELECT * FROM history('5PLf9SXwndd63lPaSIa0O6', `2000T`, `2019-06-05T23:59:59Z`) AS h
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

テーブルを再アクティブ化

QLDB でテーブルを非アクティブ化したら、テーブルの削除の取り消し ステートメントを使用してテーブルを再度アクティブにすることができます。

まず、information_schema.user_tables からテーブル ID を見つけます。たとえば、次のクエリは VehicleRegistration テーブルの tableId を返します。ステータスは INACTIVE である必要があります。

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

次に、この ID を使用してテーブルを再度有効にします。以下に、テーブル ID 5PLf9SXwndd63lPaSIa0O6 の削除を取り消す例を示します。この場合、テーブル ID は、二重引用符で囲まれた一意の識別子です。

```
UNDROP TABLE "5PLf9SXwndd63lPaSIa0O6"
```

これで、VehicleRegistration のステータスが ACTIVE になります。

インデックスを作成、説明、および削除する方法については、インデックスの管理 に進んでください。

インデックスの管理

このセクションでは、Amazon QLDB でインデックスを作成、説明、削除する方法を説明します。作成できるテーブルあたりのインデックス数のクォータは、Amazon QLDB でのクォータと制限 で定義されます。

トピック

インデックスの作成

「テーブルとインデックスの作成」でも説明したように、CREATE INDEX ステートメントを使用して、次のように、指定したトップレベルフィールドのテーブル上にインデックスを作成することができます。テーブル名とインデックス付きフィールド名は、大文字と小文字を区別します。

```
CREATE INDEX ON VehicleRegistration (VIN)
```

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

テーブルで作成する各インデックスには、システムによって割り当てられた一意の ID があります。このインデックス ID を検索するには、次のセクション「インデックスの説明」を参照してください。

> **Important**
>
> ドキュメントを効率的に検索するには、インデックスが必要です。インデックスがないと、QLDB はドキュメントを読み取るときにテーブルスキャンを実行する必要があります。これにより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
>
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子 (= または IN) を使用する WHERE 述語句でステートメントを実行する必要があります。詳細については、「クエリパフォーマンスの最適化」を参照してください。

インデックスを作成する際には、以下の制約があることに注意してください。

- インデックスは 1 つのトップレベルフィールドでのみ作成できます。複合、ネスト、一意、および関数ベースのインデックスはサポートされていません。
- 任意の Ion データ型 (`list`、`struct` など) でインデックスを作成できます。ただし、Ion のデータ型に関わらず、Ion 値全体の等価によってインデックス付けされたルックアップのみを実行できます。例えば、`list` 型をインデックスとして使用すると、リスト内の 1 つの項目でインデックス付けされたルックアップは実行できません。
- クエリのパフォーマンスは、等価述語 (`WHERE indexedField = 123`、`WHERE indexedField IN (456, 789)` など) を使用する場合にのみ向上します。

  QLDB では、クエリの述語で不等式はサポートされていません。そのため、範囲でフィルタリングされるスキャンは実装されていません。
- インデックス付きフィールドの名前は大文字と小文字の区別があり 128 文字以下で指定します。
- QLDB でのインデックス作成は非同期です。空でないテーブルでのインデックスの作成を完了するのにかかる時間は、テーブルサイズによって異なります。詳細については、「インデックスの管理」を参照してください。

インデックスの説明

QLDB でのインデックス作成は非同期です。空でないテーブルでのインデックスの作成を完了するのにかかる時間は、テーブルサイズによって異なります。インデックス構築のステータスを確認するには、システムカタログテーブル information_schema.user_tables のクエリを実行します。

例えば、次のステートメントは、`VehicleRegistration` テーブルのすべてのインデックスに対してシステムカタログのクエリを実行します。

```
SELECT VALUE indexes
FROM information_schema.user_tables info, info.indexes indexes
WHERE info.name = 'VehicleRegistration'
```

```
{
    indexId: "Djg2nt0yIs2GY0T29Kud1z",
    expr: "[VIN]",
    status: "ONLINE"
},
{
    indexId: "4tPW3fUhaVhDinRgKRLhGU",
```

```
    expr: "[LicensePlateNumber]",
    status: "FAILED",
    message: "aws.ledger.errors.InvalidEntityError: Document contains multiple values
 for indexed field: LicensePlateNumber"
}
```

インデックスフィールド

- `indexId`: インデックスの一意の ID。
- `expr`: インデックス付きドキュメントパス。このフィールドは、`[fieldName]` 形式の文字列です。
- `status`: インデックスの現在のステータス。インデックスのステータスは以下のいずれかの値になります。
  - `BUILDING`: テーブルのインデックスをアクティブに構築しています。
  - `FINALIZING`: インデックスの構築が完了し、使用するためにアクティブ化を開始しています。
  - `ONLINE`: アクティブになっていて、クエリで使用できる状態です。QLDB は、ステータスがオンラインになるまで、クエリのインデックスを使用しません。
  - `FAILED`: 回復不能なエラーのため、インデックスを構築できません。この状態のインデックスは、テーブルあたりのインデックスのクォータにそのまま不利に作用します。詳細については、「一般的なエラー」を参照してください。
  - `DELETING`: ユーザーがインデックスを削除した後、インデックスをアクティブに削除します。
- `message`: インデックスに `FAILED` ステータスがある理由を説明するエラーメッセージ。このフィールドは、失敗したインデックスにのみ含まれます。

コンソールを使用する場合

AWS Management Console を使用してインデックスのステータスを確認することもできます。

インデックスのステータスを確認するには (コンソール)

1. AWS Management Console にサインインして、Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. [Ledgers] (台帳) リストで、インデックスを管理する台帳の名前を選択します。
4. 台帳の詳細ページの [Tables] (テーブル) タブで、インデックスを確認するテーブル名を選択します。

5. テーブルの詳細ページで、[Indexed fields] (インデックス付きフィールド) カードを見つけます。[Index status] (インデックス ステータス) 列に、テーブルの各インデックスの現在のステータスが表示されます。

インデックスの削除

DROP INDEX ステートメントを使用して、インデックスを削除します。インデックスを削除すると、テーブルから完全に削除されます。

まず、`information_schema.user_tables` からインデックス ID を見つけます。例えば、次のクエリは、`VehicleRegistration` テーブルでインデックス付き `LicensePlateNumber` フィールドの `indexId` を返します。

```
SELECT indexes.indexId
FROM information_schema.user_tables info, info.indexes indexes
WHERE info.name = 'VehicleRegistration' and indexes.expr = '[LicensePlateNumber]'
```

次に、この ID を使用してインデックスを削除します。以下は、インデックス ID `4tPW3fUhaVhDinRgKRLhGU` を削除する例です。インデックス ID は一意の識別子であり、二重引用符で囲まれています。

```
DROP INDEX "4tPW3fUhaVhDinRgKRLhGU" ON VehicleRegistration WITH (purge = true)
```

> **Note**
>
> `WITH (purge = true)` 句はすべての `DROP INDEX` ステートメントに必要です。現在 `true` のみが値としてサポートされています。
> キーワード `purge` は大文字と小文字が区別され、すべて小文字にする必要があります。

コンソールを使用する場合

AWS Management Console を使用してインデックスを削除することもできます。

インデックスを削除するには (コンソール)

1. AWS Management Console にサインインして、Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。

3. [Ledgers] (台帳) リストで、インデックスを管理する台帳の名前を選択します。
4. 台帳の詳細ページの [Tables] (テーブル) タブで、インデックスを削除するテーブル名を選択します。
5. テーブルの詳細ページで、[Indexed fields] (インデックス付きフィールド) カードを見つけます。削除するインデックスを選択し、[Drop index] (インデックスの削除) を選択します。

一般的なエラー

このセクションでは、インデックスの作成時に発生する可能性のある一般的なエラーについて説明し、考えられる解決策を提案します。

> **Note**
>
> ステータスが `FAILED` のインデックスはそのままテーブルあたりのインデックスのクォータに不利に作用します。また、失敗したインデックスによって、テーブルでのインデックス作成が失敗した原因となったドキュメントの変更と削除はできません。
> 明示的にインデックスを削除してクォータから削除する必要があります。

ドキュメントには、インデックス付きフィールド ***`fieldName`*** の複数の値が含まれています。

同じフィールドに対して複数の値があるドキュメントがテーブルに含まれている (つまり、フィールド名が重複する) ため、QLDB は指定したフィールド名にインデックスを構築することができません。

まず、失敗したインデックスを削除する必要があります。次に、インデックスの作成を再試行する前に、テーブル内のすべてのドキュメントが各フィールド名に 1 つの値しか持たないことを確認してください。重複のない別のフィールドのインデックスを作成することもできます。

QLDB は、テーブルで既にインデックス付けされているフィールドに対して複数の値を含むドキュメントを挿入しようとした場合も、このエラーを返します。

Exceeded indexes limit: Table ***`tableName`*** already has ***`n`*** indexes, and cannot create more. (インデックスの制限を超えました: 既にテーブル tableName には n インデックスがあり、これ以上作成することはできません。)

QLDB では、失敗したインデックスを含めて、テーブルあたりに 5 つのインデックスの制限が適用されます。新しいインデックスを作成する前に、既存のインデックスを削除する必要があります。

No defined index with identifier: ***indexId***. (識別子 IndexID を持つ定義済みのインデックスがありません。)

指定したテーブルとインデックス ID の組み合わせに存在しないインデックスを削除しようとしました。既存のインデックスを確認する方法については、「インデックスの説明」を参照してください。

Amazon QLDB で割り当てられる一意の ID

このセクションでは、Amazon QLDB のシステム割り当て一意 ID のプロパティと使用ガイドラインについて説明します。また、QLDB の一意な ID の例をいくつか示します。

トピック

- プロパティ
- 使用方法
- 例

プロパティ

QLDB では、システムによって割り当てられた ID はすべて、汎用一意 ID (UUID、Universally Unique Identifier) です。各 ID には次のプロパティがあります。

- 128 ビットの UUID 番号
- Base62 でエンコードされたテキストで表される
- 22 文字の固定長の英数字の文字列 (例: `3Qv67yjXEwB9SjmvkuG6Cp`)。

使用方法

アプリケーションで QLDB の一意の ID を使用する場合は、次のガイドラインに注意してください。

Do

- ID は文字列として扱います。

しません

- 文字列をデコードする。

- セマンティックな意味を文字列に帰する (時間コンポーネントの取得など)。
- 文字列をセマンティック順に並べ替える。

例

次の属性は、QLDB の一意の ID の例です。

- ドキュメント ID
- インデックス ID
- ストランド ID
- テーブル ID
- トランザクション ID

Amazon QLDB 同時実行モデル

Amazon QLDB は、高性能なオンライントランザクション処理 (OLTP) ワークロードのニーズに対応することを目的としています。QLDB は、SQL のようなクエリ機能をサポートし、完全な ACID トランザクションを提供します。加えて、QLDB のデータ項目は、柔軟なスキーマや直感的なデータモデリングを提供するドキュメントでもあります。ジャーナルを中核としているため、QLDB を使用して、データへのあらゆる変更の完全かつ検証可能な履歴にアクセスしたり、必要に応じて一貫したトランザクションを他のデータサービスにストリーミングしたりすることができます。

トピック

オプティミスティック同時実行制御

QLDB では、オプティミスティック同時実行制御 (OCC) を使用して同時実行制御が実行されます。OCC は、複数のトランザクションが干渉し合うことなく頻繁に完了できるという原則に基づいています。

OCC を使用すると、QLDB のトランザクションはデータベースリソースのロックを取得せず、完全な直列化可能分離で動作します。QLDB では、同時トランザクションが連続して実行されるため、トランザクションが連続して開始された場合と同じ効果を発揮します。

コミットする前に、各トランザクションは検証チェックを実行し、他のコミットされたトランザクションがアクセスしているデータを変更していないことを確認します。このチェックによって競合する変更、またはデータの変更の状態が明らかになった場合、トランザクションのコミットは拒否されます。ただし、トランザクションは再開できます。

トランザクションが QLDB にデータを書き込む場合、OCC モデルの検証チェックは QLDB 自身が実行します。OCC の検証フェーズで障害が発生したためにトランザクションをジャーナルに書き込めない場合、QLDB は、`OccConflictException` をアプリケーションレイヤーに返します。トラ

ンザクションの再責任はアプリケーションソフトウェアが負います。アプリケーションは、拒否されたトランザクションを中止し、最初からトランザクション全体を再試行する必要があります。

QLDB ドライバーが OCC の競合およびその他の一時的な例外を処理して再試行する方法については、「Amazon QLDB でドライバーが使用する再試行ポリシーを理解する」を参照してください。

インデックスを使用してテーブル全体のスキャンを回避する

QLDB では、すべての PartiQL ステートメント (すべての SELECT クエリを含む) はトランザクションで処理され、トランザクションタイムアウト制限の対象になります。

ベストプラクティスとして、インデックス付きフィールドまたはドキュメント ID でフィルタリングする WHERE 述語句を使用してステートメントを実行することをお勧めします。QLDB では、ドキュメントを効率的に検索するために、インデックス付きフィールドに等価演算子 (`WHERE indexedField = 123` や `WHERE indexedField IN (456, 789)` など) が必要です。

このインデックス付きルックアップがない場合、QLDB はドキュメントを読み取るときにフルテーブルスキャンを実行する必要があります。これにより、クエリのレイテンシーとトランザクションのタイムアウトが発生する可能性が生じ、競合するトランザクションとの OCC の競合の可能性も高くなります。

たとえば、VIN フィールドにのみインデックスがある `Vehicle` という名前のテーブルについて考えます。これには以下のデータが含まれています。

VIN	Make	[Model] (モデル)	Color
`"1N4AL11D75C109151"`	`"Audi"`	`"A5"`	`"Silver"`
`"KM8SRDHF6EU074761"`	`"Tesla"`	`"Model S"`	`"Blue"`
`"3HGGK5G53FM761765"`	`"Ducati"`	`"Monster 1200"`	`"Yellow"`
`"1HVBBAANXWH544237"`	`"Ford"`	`"F 150"`	`"Black"`

VIN	Make	[Model] (モデル)	Color
"1C4RJFAG 0FC625797"	"Mercedes"	"CLK 350"	"White"

2 人の同時ユーザー (Alice と Bob) が、台帳内の同じテーブルを使用しています。彼らは、次のように 2 つの異なるドキュメントを更新します。

Alice:

```
UPDATE Vehicle AS v
SET v.Color = 'Blue'
WHERE v.VIN = '1N4AL11D75C109151'
```

Bob:

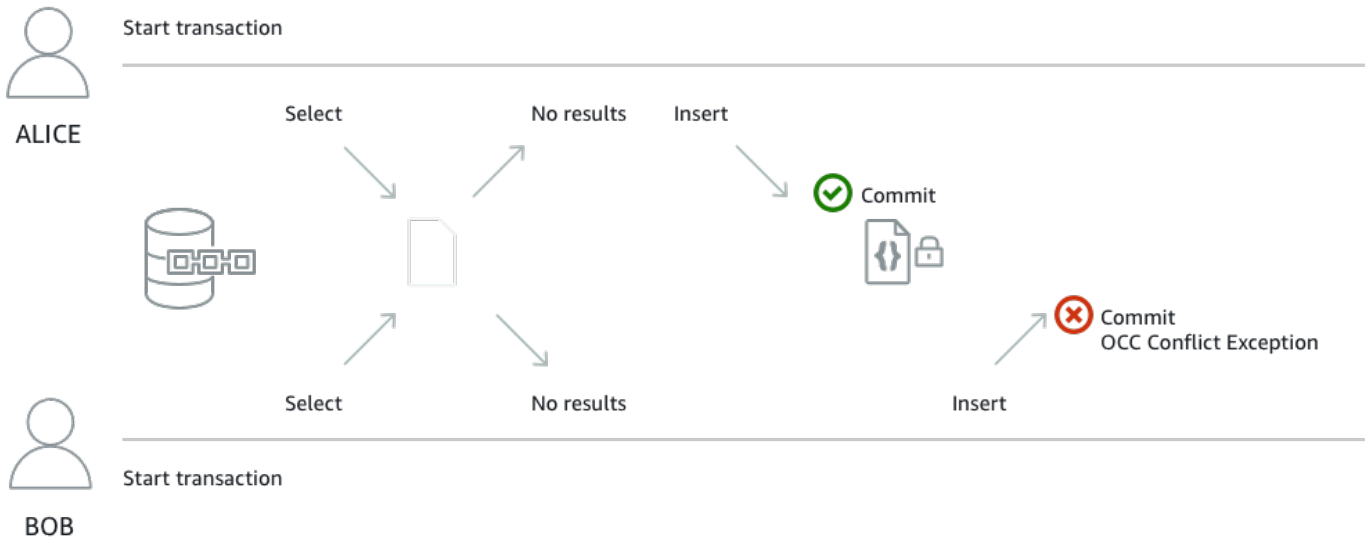
```
UPDATE Vehicle AS v
SET v.Color = 'Red'
WHERE v.Make = 'Tesla' AND v.Model = 'Model S'
```

Alice と Bob が同時にトランザクションを開始するとします。Alice の UPDATE ステートメントは VIN フィールドに対してインデックス付きのルックアップを行うため、該当する 1 つのドキュメントのみを読み取る必要があります。Alice は終了し、最初にトランザクションを正常にコミットします。

Bob のステートメントは、インデックス付けされていないフィールドをフィルタリングするため、テーブルスキャンを実行し、OccConflictException が発生します。これは、Alice のコミットされたトランザクションが Bob のステートメントがアクセスしているデータを変更したためです。これには、Bob が更新中のドキュメントだけでなく、テーブル内のすべてのドキュメントが含まれます。

挿入の OCC 競合

OCC の競合には、以前に存在していたドキュメントだけでなく、新しく挿入されたドキュメントも含まれる場合があります。次の図では、2 人の同時ユーザー (Alice と Bob) が台帳内の同じテーブルを操作しているとします。彼らは両方とも、述語値がまだ存在しないという条件の下でのみ新しいドキュメントを挿入したいと考えています。



この例では、Alice と Bob の両方が 1 つのトランザクション内で次の SELECT および INSERT ステートメントを実行します。アプリケーションは、INSERT ステートメントが結果を返さない場合にのみ、SELECT ステートメントを実行します。

```
SELECT * FROM Vehicle v WHERE v.VIN = 'ABCDE12345EXAMPLE'
```

```
INSERT INTO Vehicle VALUE
{
    'VIN' : 'ABCDE12345EXAMPLE',
    'Type' : 'Wagon',
    'Year' : 2019,
    'Make' : 'Subaru',
    'Model' : 'Outback',
    'Color' : 'Gray'
}
```

Alice と Bob が同時にトランザクションを開始するとします。どちらの SELECT クエリも、ABCDE12345EXAMPLE の　VIN が付いた既存のドキュメントを返しません 。したがって、彼らのアプリケーションは INSERT ステートメントを続行します。

Alice は終了し、最初にトランザクションを正常にコミットします。その後、Bob はトランザクションをコミットしようとしますが、それを QLDB が拒否し、OccConflictException がスローされます。これは、Alice のコミットされたトランザクションが Bob の SELECT クエリの結果セットを変更し、OCC が Bob のトランザクションをコミットする前にこの競合を検出するためです。

このトランザクション例をべき等にするには SELECT クエリが必要です。Bob は、最初からトランザクション全体を再試行することができます。しかし、彼の次の SELECT クエリは、Alice が挿入したドキュメントを返すので、Bob のアプリケーションは INSERT を実行しません。

トランザクションをべき等にする

前のセクションの insert トランザクションも、べき等トランザクションの例です。つまり、同じトランザクションを複数回実行すると、同じ結果が得られます。Bob が、特定の VIN が既に存在しているかどうかを最初に確認せずに INSERT を実行すると、テーブルには、重複した VIN 値を持つドキュメントが作成されることがあります。

OCC の競合に加えて、他の再試行シナリオを考えてみます。例えば、QLDB はサーバー側でトランザクションを正常にコミットできるが、レスポンスを待機中にクライアントはタイムアウトする可能性があります。ベストプラクティスとして、同時実行または再試行の場合に予期しない結果を避けるために、書き込みトランザクションをべき等にしてください。

秘匿化 OCC コンフリクト

QLDB は、同じジャーナルブロックのリビジョンの同時秘匿化を防ぎます。2 人の同時接続ユーザー (Alice と Bob) が、台帳の同じブロックにコミットされている 2 つの異なるドキュメントのリビジョンを秘匿化したいという例を考えてみましょう。まず、Alice は次のように REDACT_REVISION ストアドプロシージャを実行して、1 つのリビジョンの秘匿化をリクエストします。

```
EXEC REDACT_REVISION `{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`,
 '5PLf9SXwndd63lPaSIa0O6', 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

次に、Alice のリクエストがまだ処理中である間に、Bob は次のように別のリビジョンの秘匿化をリクエストします。

```
EXEC REDACT_REVISION `{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`,
 '8F0TPCmdNQ6JTRpiLj2TmW', '05K8zpGYWynDlEOK5afDRc'
```

QLDB は、2 つの異なるドキュメントのリビジョンを秘匿化しようとしているにもかかわらず、Bob のリクエストを OccConflictException で拒否します。これは、Alice が秘匿化中のリビジョンと Bob のリビジョンが同じブロックにあるからです。Alice のリクエストの処理が完了すると、Bob は秘匿化リクエストを再試行できます。

同様に、2 つの同時トランザクションが同じリビジョンを秘匿化しようとした場合、処理できるリクエストは 1 つだけです。もう 1 つのリクエストは、秘匿化が完了するまで OCC コンフリクト例外で

失敗します。その後、同じリビジョンを秘匿化しようとすると、そのリビジョンが既に秘匿化されていることを示すエラーが発生します。

同時セッションの管理

リレーショナルデータベース管理システム (RDBMS) を使用した経験のある方は同時接続の制限の知識をお持ちでしょう。QLDB の概念は、従来の RDBMS 接続とは異なります。HTTP のリクエストメッセージとレスポンスメッセージを使用してトランザクションを実行するからです。

QLDB では、同様の概念をアクティブセッションと呼びます。セッションは概念的にはユーザーログインに似ており、これによって、台帳に対するデータトランザクションリクエストの情報が管理されます。アクティブセッションとは、トランザクションがアクティブに実行されているセッションを意味します。トランザクションが終了したばかりで、サービスが別のトランザクションがすぐに開始されることを期待している状態もセッションと見なされることがあります。QLDB ではセッションごとに 1 つのトランザクションがアクティブに実行されます。

台帳ごとの同時アクティブセッションの制限数は、「Amazon QLDB でのクォータと制限」に定義されています。この制限に達すると、トランザクションの開始を試行するセッションはエラー (`LimitExceededException`) になります。

セッションのライフサイクルと、データトランザクションの実行時に QLDB ドライバーがセッションを処理する方法の詳細については、「ドライバーによるセッション管理」を参照してください。QLDB ドライバーを使用してアプリケーション内のセッションプールを設定するベストプラクティスについては、「Amazon QLDB の推奨事項」の「QldbDriver オブジェクトの設定」を参照してください。

Amazon QLDB でのデータ検証

Amazon QLDB を使用すると、アプリケーションデータの変更履歴が正確であると信頼できます。QLDB では、イミュータブルトランザクションログ (ジャーナル) をデータストレージに使用します。ジャーナルは、コミット済みデータへの変更をすべて追跡し、完全かつ検証可能な変更履歴を一定の期間にわたって維持します。

QLDB は、SHA-256 ハッシュ関数をマークルツリーベースのモデルで使用して、ジャーナルの暗号化表現 (ダイジェスト) を生成します。ダイジェストは、ある時点におけるデータの変更履歴全体の一意の署名として機能します。ダイジェストを使用して、その署名に関連するドキュメントのリビジョンの整合性を検証します。

トピック

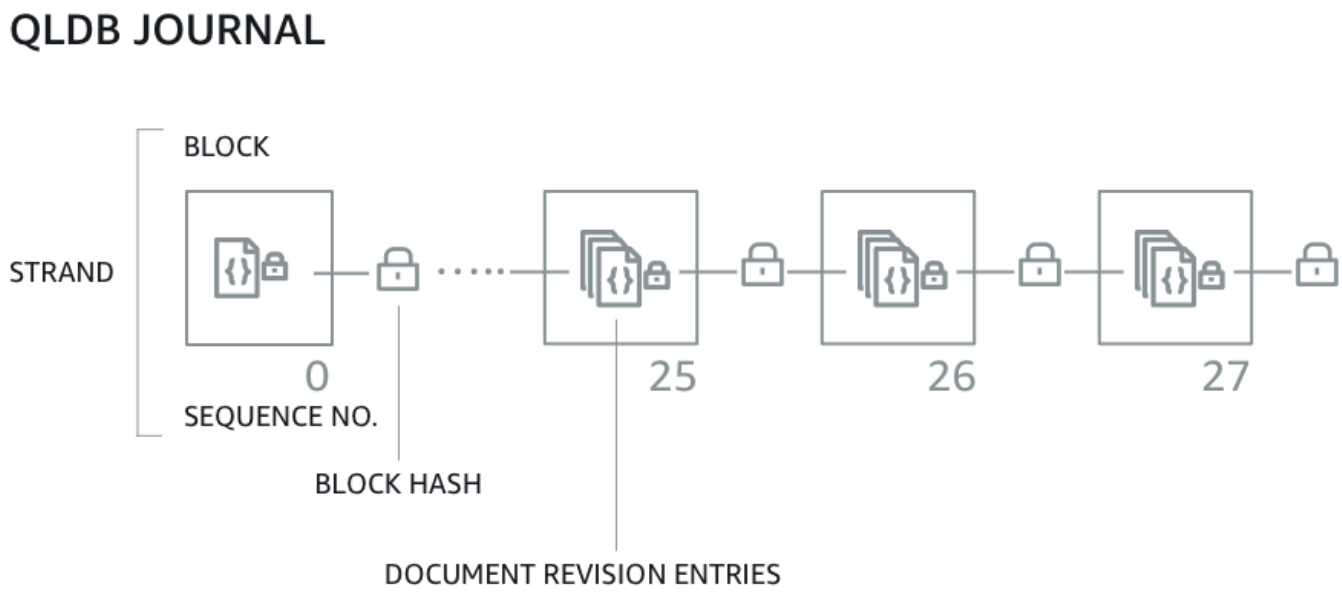
QLDB で検証できるデータの種類

QLDB では、台帳ごとにジャーナルが 1 つ作成されます。ジャーナルには、ジャーナルのパーティションとなる複数のストランドを設けることができます。

> **Note**
>
> QLDB は現在、単一ストランドのジャーナルのみをサポートしています。

ブロックは、トランザクション中にジャーナルストランドにコミットされるオブジェクトです。このブロックには、トランザクションに起因するドキュメントリビジョンを表すエントリオブジェクトが含まれます。QLDB では、個々のリビジョンまたはジャーナルブロック全体を検証できます。

以下の図は、このジャーナルの構造を示したものです。



この図は、トランザクションがドキュメントのリビジョンのエントリを含むブロックとしてジャーナルにコミットされていることを示しています。また、各ブロックが後続のブロックにハッシュチェーンされ、ストランド内のアドレスを指定するシーケンス番号があることも示しています。

ブロック内のデータコンテンツについては、「Amazon QLDB のジャーナルコンテンツ」を参照してください。

データの整合性とは

QLDB におけるデータ整合性とは、台帳のジャーナルが実際にイミュータブルであることを意味します。言い換えると、お客様のデータ (具体的には、各ドキュメントリビジョン) が、以下に該当する状態にあることをいいます。

1. ジャーナル内で最初に書き込まれた場所と同じ場所に存在する。
2. 書き込み以降にいかなる方法でも改変されていない。

検証の仕組み

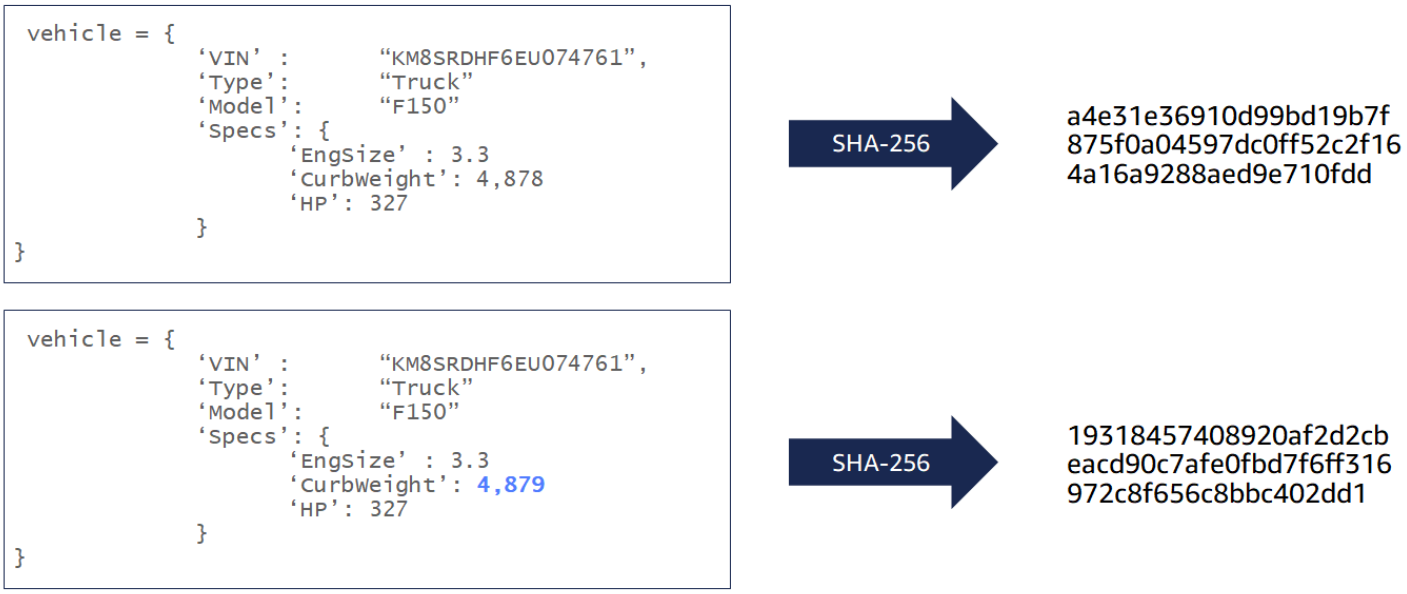
Amazon QLDB での検証の仕組みを理解するために、概念を 4 つの基本コンポーネントに分けることができます。

- ハッシュ
- ダイジェスト
- マークルツリー
- 証明

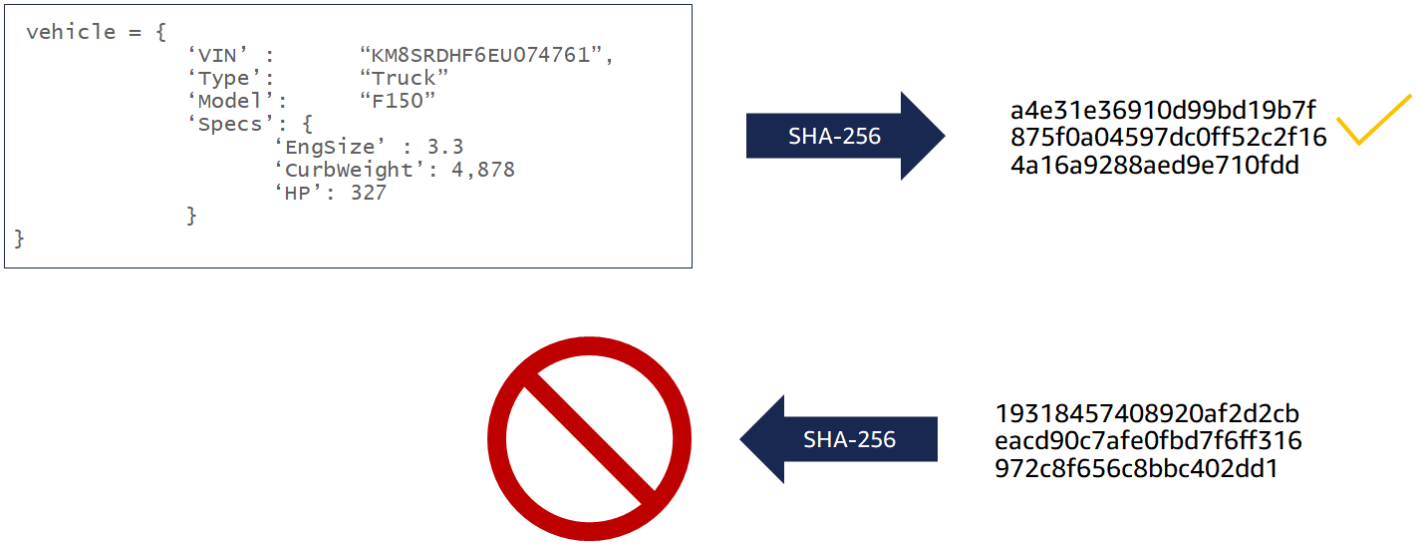
ハッシュ

QLDB は、SHA-256 暗号化ハッシュ関数を使用して 256 ビットのハッシュ値を作成します。ハッシュは、任意の量の入力データの一意の固定長の署名として機能します。入力の任意の部分 (1 文字またはビットでも) を変更すると、出力ハッシュは完全に変更されます。

以下の図は、SHA-256 ハッシュ関数が、1 桁だけ異なる 2 つの QLDB ドキュメントに対して完全に一意のハッシュ値を作成することを示しています。



SHA-256 ハッシュ関数は一方向です。つまり、出力が与えられたときに入力を計算することは数学的に実行可能ではありません。以下の図は、出力ハッシュ値が与えられたときに入力 QLDB ドキュメントを計算することが不可能であることを示しています。



検証の目的で、次のデータ入力が QLDB でハッシュされます。

- ドキュメントの改訂
- PartiQL ステートメント
- リビジョンのエントリ
- ジャーナルブロック

ダイジェスト

ダイジェストは、ある時点での台帳のジャーナル全体の暗号表現です。ジャーナルは追加専用であり、ジャーナルブロックはブロックチェーンと同様に配列され、ハッシュチェーンされます。

台帳のダイジェストはいつでもリクエストできます。QLDB はダイジェストを生成してセキュアな出力ファイルとして返します。その後、そのダイジェストを使用して、以前の時点でコミットされたドキュメントのリビジョンの整合性を検証できます。リビジョンで始まり、ダイジェストで終わることによってハッシュを再計算すると、データがその間に変更されていないことが証明されます。

マークルツリー

台帳のサイズが大きくなるにつれて、検証のためにジャーナルの完全なハッシュチェーンを再計算することはますます非効率的になります。QLDB はマークルツリーモデルを使用して、この非効率性を解決します。

マークルツリーは、構造内の各リーフノードがデータブロックのハッシュを表すツリーデータ構造です。リーフ以外の各ノードは、その子ノードのハッシュです。ブロックチェーンで一般的に使用され

ているマークルツリーを使うと、監査プルーフの仕組みを使って大規模なデータセットを効率よく検証できます。マークルツリーの詳細については、Wikipedia のマークルツリーに関するページを参照してください。マークルツリーにおける監査プルーフとそのユースケース例については、Certificate Transparency サイトの「How Log Proofs Work」を参照してください。

マークルツリーの QLDB 実装は、ジャーナルの完全なハッシュチェーンから構築されます。このモデルでは、リーフノードはすべての個々のドキュメントリビジョンハッシュのセットです。ルートノードは、ある時点でのジャーナル全体のダイジェストを表します。

マークル監査プルーフを使用すると、台帳の改訂履歴の小さなサブセットのみをチェックすることで、リビジョンを検証できます。これを行うには、特定のリーフノード (リビジョン) からルート (ダイジェスト) までツリーをトラバースします。このトラバーサルパスに沿って、ダイジェストで終わるまで、親ハッシュを計算するために、ノードの兄弟ペアを再帰的にハッシュします。このトラバーサルには、ツリーの `log(n)` ノードの時間計算量があります。

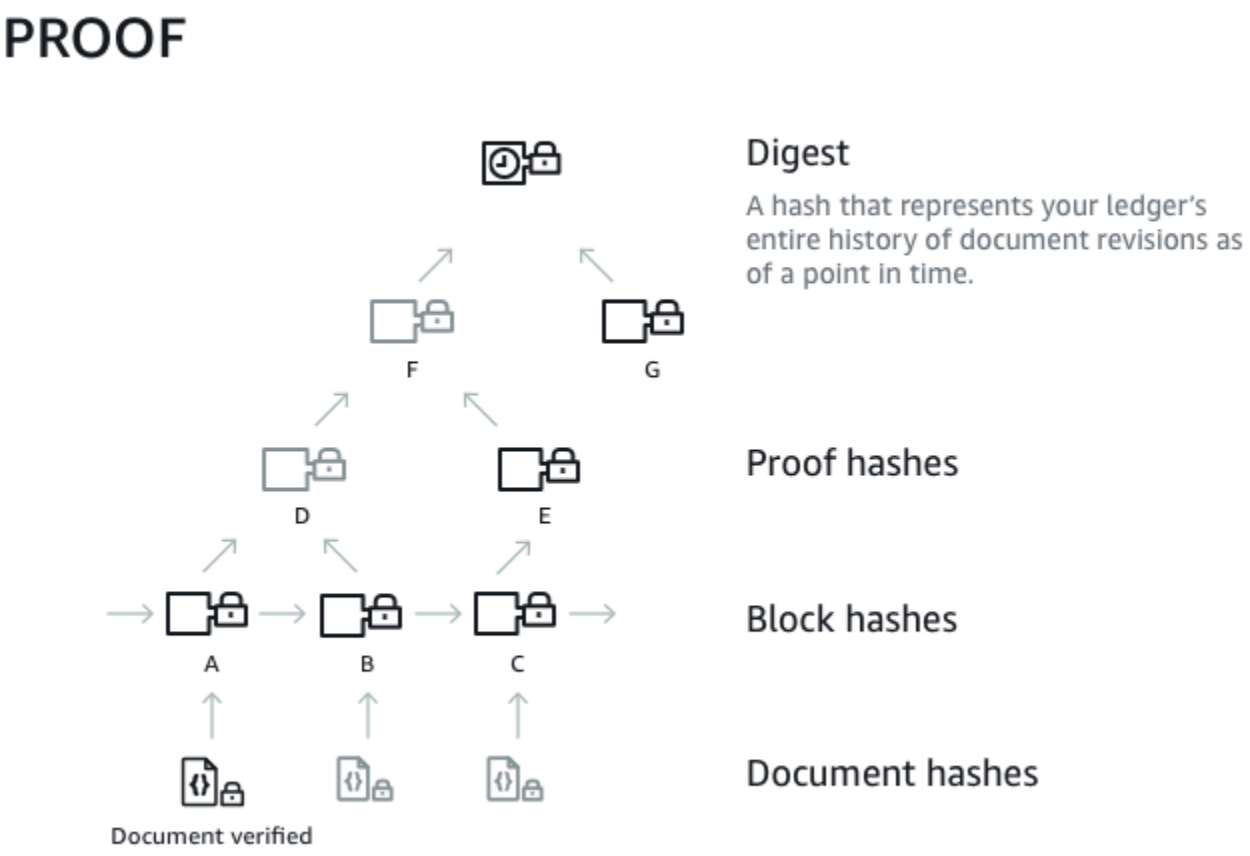
証明

証明は、指定されたダイジェストとドキュメントリビジョンに対して QLDB が返すノードハッシュの順序付きリストです。これは、マークルツリーモデルによって指定されたリーフノードハッシュ (リビジョン) をルートハッシュ (ダイジェスト) に連鎖させるために必要なハッシュで構成されています。

リビジョンとダイジェストの間でコミットされたデータを変更すると、ジャーナルのハッシュチェーンが壊れ、証明を生成できなくなります。

検証の例

以下の図は、Amazon QLDB のハッシュツリーモデルを示したものです。この図は、ジャーナルストランドのダイジェストを表す、トップルートノードまでロールアップする一連のブロックハッシュを表しています。単一ストランドのジャーナルを持つ台帳では、このルートノードが台帳全体のダイジェストも兼ねます。



ノード [A] が、そのハッシュを検証するドキュメントのリビジョンを含むブロックであると仮定します。ノード B、E、G は、QLDB が証明で示すハッシュの順序付きリストを表します。これらのハッシュではハッシュ A からダイジェストを再計算する必要があります。

ダイジェストを再計算するには、以下の手順を実行します。

1. まず、ハッシュ A をハッシュ B と連結し、その結果をハッシュして D を計算します。
2. D と E を使用して F を計算します。
3. F と G を使用してダイジェストを計算します。

再計算したダイジェストが想定値と一致したら検証は成功です。リビジョンハッシュとダイジェストが与えられた場合、証明のハッシュをリバースエンジニアリングすることは不可能です。したがって、この演習では、リビジョンが実際にダイジェストに対してこのジャーナルの位置に書かれていることを証明します。

データの秘匿化は検証にどのような影響を与えますか?

Amazon QLDB では、DELETE ステートメントは、削除済みとしてマークする新しいリビジョンを作成することによってのみドキュメントを論理的に削除します。QLDB テーブルの履歴にある使用頻度の低いドキュメントリビジョンを完全に削除できるデータの秘匿化オペレーションもサポートしています。

秘匿化オペレーションでは、指定されたリビジョンのユーザーデータのみが削除され、ジャーナルシーケンスとドキュメントのメタデータは変更されません。リビジョンが秘匿化されると、リビジョン内の (data 構造によって表される) ユーザーデータが新しい dataHash フィールドに置き換えられます。このフィールドの値は、削除された data 構造の Amazon Ion ハッシュです。詳細と秘匿化オペレーションの例については、「ドキュメントのリビジョンを秘匿化する」を参照してください。

その結果、台帳は全体的なデータ整合性を維持し、既存の検証 API オペレーションを通じて暗号的に検証できる状態を維持します。これらの API オペレーションを期待どおりに使用して、ダイジェストをリクエストし (GetDigest)、証明をリクエストし (GetBlock または GetRevision)、返されたオブジェクトを使用して検証アルゴリズムを実行できます。

リビジョンハッシュの再計算

ハッシュを再計算することによって、個々のドキュメントのリビジョンを検証する場合は、そのリビジョンが秘匿化されたかどうかを条件付きで確認する必要があります。リビジョンが秘匿化された場合は、dataHash フィールドに提供されているハッシュ値を使用できます。秘匿化されていない場合は、data フィールドを使用してハッシュを再計算できます。

この条件付きチェックを行うことで、秘匿化されたリビジョンを特定し、適切な処置を取ることができます。例えば、モニタリングのためにデータ操作イベントをログに記録できます。

検証の使用開始

データを検証するには、台帳にダイジェストをリクエストし、後の照合に使用できるよう保存しておく必要があります。ダイジェストにより力バーされる最新のブロックより前にコミットされたすべてのドキュメントリビジョンが、ダイジェストに照らし合わせて検証できる対象となります。

その後、検証対象となる使用可能なリビジョンに関する証明を Amazon QLDB でリクエストします。このプルーフを使用して、ダイジェストを再計算するためのクライアント側 API を呼び出し、リビジョンのハッシュから始めます。過去に保存されたダイジェストがわかっており、QLDB の外部で信頼されている限り、ドキュメントの整合性は、再計算したダイジェストハッシュが保存されているダイジェストハッシュと一致した場合に証明されます。

Important

- この検証で具体的に証明できることは、このダイジェストを保存した時点から検証を実施する時点までの間に、ドキュメントリビジョンが改変されていないということです。後で検証するリビジョンがジャーナルにコミットされ次第、ダイジェストをリクエストして保存できます。
- ベストプラクティスとして、定期的にダイジェストをリクエストし、台帳から離れた場所に保管することをお勧めします。台帳でリビジョンをコミットする頻度に基づいて、ダイジェストをリクエストする頻度を決定します。

  実際のユースケースにおける暗号化検証の価値について説明する詳細な AWS ブログ記事については、「Amazon QLDB による実際の暗号化検証」を参照してください。

台帳からダイジェストをリクエストし、データを検証する step-by-step 方法については、以下を参照してください。

- ステップ 1: QLDB でのダイジェストのリクエスト
- ステップ 2: QLDB でのデータの検証

ステップ 1: QLDB でのダイジェストのリクエスト

Amazon QLDB には、台帳のジャーナルの現在のティップを含むダイジェストをリクエストする API が用意されています。ジャーナルのティップとは、QLDB がリクエストを受けた時点でコミット済みの最新のブロックのことです。 AWS Management Console、 AWS SDK、または AWS Command Line Interface （ AWS CLI) を使用してダイジェストを取得できます。

トピック

- AWS Management Console
- QLDB API

AWS Management Console

QLDB コンソールを使用してダイジェストをリクエストするには、以下のステップに従います。

ダイジェストをリクエストするには (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/qldb で Amazon QLDB コンソールを開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. 台帳のリストで、ダイジェストをリクエストする台帳名を選択します。
4. [Get digest (ダイジェストの取得)] を選択します。[Get digest (ダイジェストの取得)] ダイアログボックスに、以下のようなダイジェスト詳細が表示されます。

   - [Digest] (ダイジェスト) - リクエストしたダイジェストの SHA-256 ハッシュ値。
   - [Digest tip address] (ダイジェストティップアドレス) - リクエストしたダイジェストの対象となっているジャーナル内の最新のブロック位置。アドレスには以下の 2 つのフィールドがあります。
     - `strandId` - ブロックを含むジャーナルストランドの一意の ID。
     - `sequenceNo` - ストランド内のブロックの位置を指定するインデックス番号。
   - [Ledger] (台帳) - ダイジェストをリクエストした台長名。
   - [Date] (日付) - ダイジェストをリクエストしたときのタイムスタンプ。
5. ダイジェスト情報を確認します。次に、[Save] (保存) を選択します。デフォルトのファイル名はそのままにしておくことも、新しい名前を入力することもできます。

   > **Note**
   >
   > 台帳内のデータに修正を加えていない場合でも、ダイジェストハッシュやティップアドレス値に変更が認められる場合があります。これは、PartiQL エディタでクエリを実行するたびに、コンソールによって台帳のシステムカタログが取得されるためです。これは読み取りトランザクションで、ジャーナルにコミットされ、最新ブロックアドレスの変更を引き起こします。

   このステップでは、Amazon Ion 形式のコンテンツを含むプレーンテキストファイルが保存されます。このファイルには、`.ion.txt` のファイル名拡張子が付いており、前述のダイアログボックスに列挙されたすべてのダイジェスト情報が含まれています。以下は、ダイジェストファイルのコンテンツ例です。フィールドの順序はブラウザにより異なる場合があります。

```
{
  "digest": "42zaJOfV8iGutVGNaIuzQWhD5Xb/5B9lScHnvxPXm9E=",
```

```
  "digestTipAddress": "{strandId:\"BlFTjlSXze9BIh1KOszcE3\",sequenceNo:73}",
  "ledger": "my-ledger",
  "date": "2019-04-17T16:57:26.749Z"
}
```

6. 後でアクセスできるよう、このファイルを保存しておきます。このファイルは、後でドキュメントリビジョンを検証する際に使用します。

> **Important**
>
> 後に検証するドキュメントリビジョンは、保存したダイジェストにより カバーされているものでなければなりません。つまり、ドキュメントのアドレスのシーケンス番号は、[Digest tip address (ダイジェストティップアドレス)] のシーケンス番号以下であることが必要です。

QLDB API

AWS SDK または AWS CLIで Amazon QLDB API を使用して、台帳のダイジェストをリクエストすることもできます。QLDB API は、アプリケーションプログラムで使用する、以下のオペレーションを提供します。

- GetDigest – ジャーナル内の最後にコミットされたブロックにある台帳のダイジェストを返します。応答には、256 ビットのハッシュ値とブロックアドレスが含まれます。

を使用してダイジェストをリクエストする方法については AWS CLI、 AWS CLI コマンドリファレンス の get-digest コマンドを参照してください。

サンプルアプリケーション

Java コードの例については、 GitHub リポジトリ aws-samples/amazon-qldb-dmv-sample-java を参照してください。このサンプルアプリケーションをダウンロードしてインストールする方法については、「Amazon QLDB Java サンプルアプリケーションのインストール」を参照してください。ダイジェストをリクエストする前に、「Java チュートリアル」のステップ 1 ~ 3 を実行し、サンプル台帳を作成して、サンプルデータを使用してロードしてください。

クラス のチュートリアルコードGetDigestは、`vehicle-registration`サンプル台帳からダイジェストをリクエストする例を示しています。

保存したダイジェストを使用してドキュメントリビジョンを検証するには、「ステップ 2: QLDB でのデータの検証」に進みます。

ステップ 2: QLDB でのデータの検証

Amazon QLDB は、指定されたドキュメント ID とそれに関連付けられたブロックの証明をリクエストするための API を備えています。「ステップ 1: QLDB でのダイジェストのリクエスト」で説明されているように、過去に保存したダイジェストのティップアドレスも指定する必要があります。AWS Management Console、 AWS SDK、または を使用して証明 AWS CLI を取得できます。

続いて、QLDB から返された証明を使用し、クライアント側 API により、保存したダイジェストと照合してドキュメントリビジョンを検証できます。これにより、お客様は、データ検証に使用するアルゴリズムを制御できます。

トピック

- AWS Management Console
- QLDB API

AWS Management Console

このセクションでは、Amazon QLDB コンソールを使用し、過去に保存したダイジェストと照合してドキュメントリビジョンを検証するための手順について説明します。

始める前には必ず「ステップ 1: QLDB でのダイジェストのリクエスト」のステップに従ってください。検証には、検証するドキュメントリビジョンを対象としている、過去に保存したダイジェストが必要です。

ドキュメントリビジョンを検証するには (コンソール)

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. 最初に、検証するリビジョンの `id` と `blockAddress` に関するクエリを台帳に対して実行します。これらのフィールドはドキュメントのメタデータに含まれており、コミット済みビューでクエリを実行できます。

   ドキュメント `id` は、システムによって割り当てられた一意の ID 文字列です。`blockAddress` は、リビジョンがコミットされたブロックの位置を指定する Ion 構造です。

   ナビゲーションペイン内で [PartiQL エディタ] を選択します。

3. リビジョンを検証する台帳名を選択します。
4. クエリエディタウィンドウ内で SELECT 文を下記の構文に入力し、[Run (実行)] を選択します。

```
SELECT metadata.id, blockAddress FROM _ql_committed_table_name
WHERE criteria
```

例えば、以下のクエリは、「Amazon QLDB コンソールの使用開始方法」で作成されたサンプル台帳の VehicleRegistration テーブルのドキュメントを返します。

```
SELECT r.metadata.id, r.blockAddress FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN = 'KM8SRDHF6EU074761'
```

5. クエリによって返された id と blockAddress の値をコピーし、保存します。id フィールドの二重引用符は必ず省略してください。Amazon Ion では、文字列データ型は二重引用符で区切られます。たとえば、以下のスニペットでは、英数文字のテキストのみをコピーする必要があります。

   "*LtMNJYNjSwzBLgf7sLifrG*"

6. これでドキュメントリビジョンが選択されたため、検証プロセスを開始できます。

   ナビゲーションペイン内で [Verification (検証)] を選択します。

7. [Verify document (ドキュメントの検証)] フォームの [Specify the document that you want to verify (検証するドキュメントの指定)] に、次の入力パラメータを入力します。

   - [Ledger] (台帳) - リビジョンを検証する台帳。
   - [ブロックアドレス] - ステップ 4 のクエリで返された blockAddress 値。
   - [ドキュメント ID] - ステップ 4 のクエリで返された id 値。

8. [Specify the digest to use for verification (検証に使用するダイジェストの指定)] で、[Choose digest (ダイジェストの選択)] を選択することで、以前に保存されたダイジェストを選択します。ファイルが有効であれば、コンソールにあるすべてのダイジェストフィールドにデータが自動入力されます。また、以下の値をダイジェストファイルから手動で直接コピーして貼り付けることもできます。

   - [Digest] (ダイジェスト) - ダイジェストファイルからの digest 値。
   - [Digest tip address] (ダイジェストティップアドレス) - ダイジェストファイルからの digestTipAddress 値。

9. ドキュメントおよびダイジェストの入力パラメータを再確認し、[Verify (検証)] を選択します。

コンソールが次の 2 つのステップを自動的に実行します。

a. 指定されたドキュメントに関するプルーフを QLDB にリクエストします。
b. QLDB によって返された証拠を使用してクライアント側 API を呼び出すことで、提供されたダイジェストと照合してドキュメントのリビジョンを検証します。この検証のアルゴリズムを調べるには、以下のセクション「QLDB API」を参照し、サンプルコードをダウンロードしてください。

コンソールの [Verification results (検証結果)] カードに、リクエストの結果が表示されます。詳細については、「検証結果」を参照してください。

QLDB API

Amazon QLDB API を AWS SDK または AWS CLIで使用して、ドキュメントリビジョンを検証することもできます。QLDB API は、アプリケーションプログラムで使用する、以下のオペレーションを提供します。

- `GetDigest` - ジャーナル内の最後にコミットされたブロックにある台帳のダイジェストを返します。応答には、256 ビットのハッシュ値とブロックアドレスが含まれます。
- `GetBlock` - ジャーナルの指定されたアドレスにあるブロックオブジェクトを返します。また、`DigestTipAddress` が指定されている場合は、指定されたブロックの証明を検証のために返します。
- `GetRevision` - 指定されたドキュメント ID とブロックアドレスのリビジョンデータオブジェクトを返します。また、`DigestTipAddress` が指定されている場合は、指定されたリビジョンの証明を検証のために返します。

これらの API オペレーションの詳細については、Amazon QLDB API リファレンス を参照してください。

を使用してデータを検証する方法については AWS CLI、 コマンドAWS CLI リファレンス を参照してください。

サンプルアプリケーション

Java コードの例については、 GitHub リポジトリ aws-samples/amazon-qldb-dmv-sample-java を参照してください。このサンプルアプリケーションをダウンロードしてインストールする方法について

は、「Amazon QLDB Java サンプルアプリケーションのインストール」を参照してください。検証を実行する前に、「Java チュートリアル」のステップ 1～3 を実行し、サンプル台帳を作成して、サンプルデータを使用してロードしてください。

クラス のチュートリアルコードGetRevisionは、ドキュメントリビジョンの証明をリクエストし、そのリビジョンを検証する例を示しています。このクラスは以下のステップを実行します。

1. サンプル台帳 `vehicle-registration` から新しいダイジェストをリクエストします。
2. `vehicle-registration` 台帳の `VehicleRegistration` テーブルに、サンプルのドキュメントリビジョンのプルーフをリクエストします。
3. 返されたダイジェストとプルーフを使用してサンプルリビジョンを検証します。

検証結果

このセクションでは、 AWS Management Consoleにおいて Amazon QLDB データ検証リクエストから返された結果について説明します。検証リクエストを送信する方法の詳しいステップについては、「ステップ 2: QLDB でのデータの検証」を参照してください。

QLDB コンソールの [Verification] (検証) ページの [Verification results] (検証結果) カードにリクエストの結果が表示されます。[Proof] (証明) タブには、ユーザーが指定したドキュメントリビジョンとダイジェストに関して QLDB が返した証明の内容が表示されます。これには、以下の詳細が含まれています。

- [リビジョンハッシュ] - 検証するドキュメントリビジョンを一意に表す SHA-256 値。
- [Proof hashes] (証明ハッシュ) - QLDB が返す、指定されたダイジェストの再計算に使用するハッシュの順序付きリスト。コンソールは、[Revision hash (リビジョンハッシュ)] から開始し、その結果と各プルーフのハッシュを順次組み合わせ、再計算されたダイジェストに達するまで実行します。

  デフォルトではリストは折りたたまれていますが、これを展開してハッシュ値を表示することもできます。必要であれば、「証明を使用したダイジェストの再計算」に掲載のステップに従いハッシュをご自身で計算することもできます。
- [Digest calculated] (算出されたダイジェスト) - [Revision hash] (リビジョンハッシュ) で実行された一連のハッシュ計算結果のハッシュ。この値が、過去に保存した [Digest (ダイジェスト)] と一致すれば、検証は成功です。

[ブロック] タブには、検証しているリビジョンを含むブロックの内容が表示されます。これには、以下の詳細が含まれています。

- [Transaction ID] (トランザクション ID) - このブロックをコミットしたトランザクションの一意の ID。
- [Transaction time] (トランザクション時間) - このブロックがストランドにコミットされた時刻のタイムスタンプ。
- [Block hash] (ブロックハッシュ) - このブロックとそのすべてのコンテンツを一意に表す SHA-256 値。
- [Block address] (ブロックアドレス) - このブロックがコミットされた台帳のジャーナル内の場所。アドレスには以下の 2 つのフィールドがあります。
  - [Strand ID] (ストランド ID) - このブロックを含むジャーナルストランドの一意の ID。
  - [Sequence number] (シーケンス番号) - ストランド内でのこのブロックの位置を指定するインデックス番号。
- [Statements] (ステートメント) - このブロックでエントリをコミットするために実行された PartiQL ステートメント。

> **Note**
>
> パラメータ化されたステートメントをプログラムで実行すると、リテラルデータではなくバインドパラメータを使用してジャーナルブロックに記録されます。たとえば、ジャーナルブロックに次のステートメントがあるとします。ここで、疑問符 (?) はドキュメントコンテンツの変数プレースホルダーです。
>
> ```
> INSERT INTO Vehicle ?
> ```

- [Document entries] (ドキュメントエントリ) - このブロックにコミットされたドキュメントリビジョン。

ドキュメントリビジョンの検証リクエストに失敗した場合には、「検証の一般的なエラー」を参照し、考えられる原因に関する情報を参照してください。

証明を使用したダイジェストの再計算

QLDB からドキュメント検証リスエストに対応した証明が返されたら、自分でハッシュ計算を試すこともできます。このセクションでは、返された証明を使用してダイジェストを再計算する手順の概要について説明します。

初めに、[Revision hash (リビジョンハッシュ] を [Proof hashes (プルーフハッシュ] リストにある最初のハッシュとペアにします。ペアにしたら以下のステップに従います。

1. 2 つのハッシュをソートします。ハッシュを符号付きバイト値でリトルエンディアン順に比較します。
2. ソートした順に 2 つのハッシュを連結します。
3. 連結後のペアを SHA-256 ハッシュジェネレータでハッシュします。
4. 新しいハッシュと証明内の次のハッシュを連結し、ステップ 1～3 を繰り返します。最後のプルーフハッシュの処理後に新たに生成されたハッシュが、再計算したハッシュとなります。

再計算したダイジェストが過去に保存したダイジェストと一致すれば、ドキュメントの検証は成功です。

これらの検証手順を示すコード例を含む step-by-step チュートリアルについては、「」を参照してくださいチュートリアル: AWS SDK を使用したデータ検証。

チュートリアル: AWS SDK を使用したデータ検証

このチュートリアルでは、 AWS SDK を介して QLDB API を使用して、Amazon QLDB 台帳のドキュメントリビジョンハッシュとジャーナルブロックハッシュを検証します。QLDB ドライバーを使用して、ドキュメントリビジョンのクエリも実行します。

車両識別番号 (VIN、Vehicle Identification Number) が `KM8SRDHF6EU074761` である車両のデータを含むドキュメントリビジョンがあるとします。ドキュメントリビジョンはテーブル `VehicleRegistration` にあり、このテーブルは台帳 `vehicle-registration` にあります。この車両のドキュメントリビジョンとこのリビジョンを含むジャーナルブロックの両方の整合性を検証するとします。

> **Note**
>
> 現実的なユースケースにおける暗号化検証の価値について説明する詳細な AWS ブログ記事については、「Amazon QLDB による実際の暗号化検証」を参照してください。

トピック

前提条件

作業を始める前に、次の操作を実行してください。

1. 「Amazon QLDB ドライバーの開始方法」で述べた各前提条件を満たすことにより、目的の言語の QLDB ドライバーを設定します。これには、 へのサインアップ AWS、開発用のプログラムによるアクセスの付与、開発環境の設定が含まれます。
2. 「Amazon QLDB コンソールの使用開始方法」のステップ 1～2 を実行して、台帳 `vehicle-registration` を作成し、事前定義済みのサンプルデータを使用してロードします。

この後、以下のステップを確認して検証の仕組みを学習し、完全なコード例を最初から最後まで実行します。

ステップ 1: ダイジェストをリクエストする

データを検証するには、まず台帳 `vehicle-registration` に後で使用するダイジェストをリクエストする必要があります。

Java

```
// Get a digest
GetDigestRequest digestRequest = new GetDigestRequest().withName(ledgerName);
GetDigestResult digestResult = client.getDigest(digestRequest);

java.nio.ByteBuffer digest = digestResult.getDigest();

// expectedDigest is the buffer we will use later to compare against our calculated
 digest
byte[] expectedDigest = new byte[digest.remaining()];
digest.get(expectedDigest);
```

.NET

```
// Get a digest
GetDigestRequest getDigestRequest = new GetDigestRequest
{
    Name = ledgerName
};
GetDigestResponse getDigestResponse =
 client.GetDigestAsync(getDigestRequest).Result;

// expectedDigest is the buffer we will use later to compare against our calculated
 digest
MemoryStream digest = getDigestResponse.Digest;
byte[] expectedDigest = digest.ToArray();
```

Go

```
// Get a digest
currentLedgerName := ledgerName
input := qldb.GetDigestInput{Name: &currentLedgerName}
digestOutput, err := client.GetDigest(&input)
if err != nil {
    panic(err)
}

// expectedDigest is the buffer we will later use to compare against our calculated
 digest
expectedDigest := digestOutput.Digest
```

Node.js

```
// Get a digest
const getDigestRequest: GetDigestRequest = {
    Name: ledgerName
};
const getDigestResponse: GetDigestResponse = await
 qldbClient.getDigest(getDigestRequest).promise();

// expectedDigest is the buffer we will later use to compare against our calculated
 digest
const expectedDigest: Uint8Array = <Uint8Array>getDigestResponse.Digest;
```

Python

```
# Get a digest
get_digest_response = qldb_client.get_digest(Name=ledger_name)

# expected_digest is the buffer we will later use to compare against our calculated
 digest
expected_digest = get_digest_response.get('Digest')
digest_tip_address = get_digest_response.get('DigestTipAddress')
```

ステップ 2: ドキュメントリビジョンをクエリする

QLDB ドライバーを使用して、VIN KM8SRDHF6EU074761 に関連付けられているブロックアドレス、ハッシュ、ドキュメント ID をクエリします。

Java

```
// Retrieve info for the given vin's document revisions
Result result = driver.execute(txn -> {
    final String query = String.format("SELECT blockAddress, hash, metadata.id FROM
 _ql_committed_%s WHERE data.VIN = '%s'", tableName, vin);
    return txn.execute(query);
});
```

.NET

```
// Retrieve info for the given vin's document revisions
```

```
var result = driver.Execute(txn => {
    string query = $"SELECT blockAddress, hash, metadata.id FROM
 _ql_committed_{tableName} WHERE data.VIN = '{vin}'";
    return txn.Execute(query);
});
```

Go

```
// Retrieve info for the given vin's document revisions
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    statement := fmt.Sprintf(
            "SELECT blockAddress, hash, metadata.id FROM _ql_committed_%s WHERE
 data.VIN = '%s'",
            tableName,
            vin)
    result, err := txn.Execute(statement)
    if err != nil {
        return nil, err
    }

    results := make([]map[string]interface{}, 0)

    // Convert the result set into a map
    for result.Next(txn) {
        var doc map[string]interface{}
        err := ion.Unmarshal(result.GetCurrentData(), &doc)
        if err != nil {
            return nil, err
        }
        results = append(results, doc)
    }
    return results, nil
})
if err != nil {
    panic(err)
}
resultSlice := result.([]map[string]interface{})
```

Node.js

```
const result: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor): Promise<dom.Value[]> => {
```

```
    const query: string = `SELECT blockAddress, hash, metadata.id FROM
 _ql_committed_${tableName} WHERE data.VIN = '${vin}'`;
    const queryResult: Result = await txn.execute(query);
    return queryResult.getResultList();
});
```

Python

```
def query_doc_revision(txn):
    query = "SELECT blockAddress, hash, metadata.id FROM _ql_committed_{} WHERE
 data.VIN = '{}'".format(table_name, vin)
    return txn.execute_statement(query)

# Retrieve info for the given vin's document revisions
result = qldb_driver.execute_lambda(query_doc_revision)
```

ステップ 3: リビジョンの証明をリクエストする

クエリ結果を反復処理し、各ブロックアドレス、ドキュメント ID および台帳名を使用して、GetRevision リクエストを実行します。リビジョンの証明を取得するには、以前に保存したダイジェストのティップアドレスも指定する必要があります。以下の API オペレーションは、ドキュメントリビジョンとリビジョンの証明を含むオブジェクトを返します。

リビジョンの構造と内容の詳細については、「ドキュメントのメタデータのクエリの実行」を参照してください。

Java

```
for (IonValue ionValue : result) {
    IonStruct ionStruct = (IonStruct)ionValue;

    // Get the requested fields
    IonValue blockAddress = ionStruct.get("blockAddress");
    IonBlob hash = (IonBlob)ionStruct.get("hash");
    String metadataId = ((IonString)ionStruct.get("id")).stringValue();

    System.out.printf("Verifying document revision for id '%s'%n", metadataId);

    String blockAddressText = blockAddress.toString();

    // Submit a request for the revision
```

```
    GetRevisionRequest revisionRequest = new GetRevisionRequest()
            .withName(ledgerName)
            .withBlockAddress(new ValueHolder().withIonText(blockAddressText))
            .withDocumentId(metadataId)
            .withDigestTipAddress(digestResult.getDigestTipAddress());

    // Get a result back
    GetRevisionResult revisionResult = client.getRevision(revisionRequest);

    ...
}
```

.NET

```
foreach (IIonValue ionValue in result)
{
    IIonStruct ionStruct = ionValue;

    // Get the requested fields
    IIonValue blockAddress = ionStruct.GetField("blockAddress");
    IIonBlob hash = ionStruct.GetField("hash");
    String metadataId = ionStruct.GetField("id").StringValue;

    Console.WriteLine($"Verifying document revision for id '{metadataId}'");

    // Use an Ion Reader to convert block address to text
    IIonReader reader = IonReaderBuilder.Build(blockAddress);
    StringWriter sw = new StringWriter();
    IIonWriter textWriter = IonTextWriterBuilder.Build(sw);
    textWriter.WriteValues(reader);
    string blockAddressText = sw.ToString();

    // Submit a request for the revision
    GetRevisionRequest revisionRequest = new GetRevisionRequest
    {
        Name = ledgerName,
        BlockAddress = new ValueHolder
        {
            IonText = blockAddressText
        },
        DocumentId = metadataId,
        DigestTipAddress = getDigestResponse.DigestTipAddress
    };
```

```
    // Get a response back
    GetRevisionResponse revisionResponse =
 client.GetRevisionAsync(revisionRequest).Result;

    ...
}
```

Go

```
for _, value := range resultSlice {
    // Get the requested fields
    ionBlockAddress, err := ion.MarshalText(value["blockAddress"])
    if err != nil {
        panic(err)
    }
    blockAddress := string(ionBlockAddress)
    metadataId := value["id"].(string)
    documentHash := value["hash"].([]byte)

    fmt.Printf("Verifying document revision for id '%s'\n", metadataId)

    // Submit a request for the revision
    revisionInput := qldb.GetRevisionInput{
        BlockAddress:     &qldb.ValueHolder{IonText: &blockAddress},
        DigestTipAddress: digestOutput.DigestTipAddress,
        DocumentId:       &metadataId,
        Name:             &currentLedgerName,
    }

    // Get a result back
    revisionOutput, err := client.GetRevision(&revisionInput)
    if err != nil {
        panic(err)
    }

    ...
}
```

Node.js

```
for (let value of result) {
    // Get the requested fields
```

```
    const blockAddress: dom.Value = value.get("blockAddress");
    const hash: dom.Value = value.get("hash");
    const metadataId: string = value.get("id").stringValue();

    console.log(`Verifying document revision for id '${metadataId}'`);

    // Submit a request for the revision
    const revisionRequest: GetRevisionRequest = {
        Name: ledgerName,
        BlockAddress: {
            IonText: dumpText(blockAddress)
        },
        DocumentId: metadataId,
        DigestTipAddress: getDigestResponse.DigestTipAddress
    };

    // Get a response back
    const revisionResponse: GetRevisionResponse = await
 qldbClient.getRevision(revisionRequest).promise();

    ...
}
```

Python

```
for value in result:
    # Get the requested fields
    block_address = value['blockAddress']
    document_hash = value['hash']
    metadata_id = value['id']

    print("Verifying document revision for id '{}'".format(metadata_id))

    # Submit a request for the revision and get a result back
    proof_response = qldb_client.get_revision(Name=ledger_name,

 BlockAddress=block_address_to_dictionary(block_address),
                                             DocumentId=metadata_id,
                                             DigestTipAddress=digest_tip_address)
```

次に、リクエストしたリビジョンの証明を取得します。

QLDB API は、ノードハッシュの順序付きリストの文字列表現として証明を返します。この文字列をノードハッシュのバイナリ表現のリストに変換するには、Amazon Ion ライブラリの Ion リーダーを使用します。Ion ライブラリの使用方法については、「Amazon Ion Cookbook」(Amazon Ion クックブック) を参照してください。

Java

次の例では、IonReader を使用してバイナリ変換を実行します。

```
String proofText = revisionResult.getProof().getIonText();

// Take the proof and convert it to a list of byte arrays
List<byte[]> internalHashes = new ArrayList<>();
IonReader reader = SYSTEM.newReader(proofText);
reader.next();
reader.stepIn();
while (reader.next() != null) {
    internalHashes.add(reader.newBytes());
}
```

.NET

次の例では、IonLoader を使用して証明を Ion データグラムにロードします。

```
string proofText = revisionResponse.Proof.IonText;
IIonDatagram proofValue = IonLoader.Default.Load(proofText);
```

Go

次の例では、Ion リーダーを使用して証明をバイナリに変換し、証明のノードハッシュのリストを反復処理します。

```
proofText := revisionOutput.Proof.IonText

// Use ion.Reader to iterate over the proof's node hashes
reader := ion.NewReaderString(*proofText)
// Enter the struct containing node hashes
reader.Next()
if err := reader.StepIn(); err != nil {
    panic(err)
}
```

Node.js

次の例では、load 関数を使用してバイナリ変換を実行します。

```
let proofValue: dom.Value = load(revisionResponse.Proof.IonText);
```

Python

次の例では、loads 関数を使用してバイナリ変換を実行します。

```
proof_text = proof_response.get('Proof').get('IonText')
proof_hashes = loads(proof_text)
```

ステップ 4: リビジョンのダイジェストを再計算する

証明のハッシュリストを使用して、ダイジェストを再計算します。これは、リビジョンハッシュから始めます。過去に保存されたダイジェストがわかっており、QLDB の外部で信頼されている限り、ドキュメントリビジョンの整合性は、再計算したダイジェストハッシュが保存されているダイジェストハッシュと一致した場合に証明されます。

Java

```
// Calculate digest
byte[] calculatedDigest = internalHashes.stream().reduce(hash.getBytes(),
 BlockHashVerification::dot);

boolean verified = Arrays.equals(expectedDigest, calculatedDigest);

if (verified) {
    System.out.printf("Successfully verified document revision for id '%s'!%n",
 metadataId);
} else {
    System.out.printf("Document revision for id '%s' verification failed!%n",
 metadataId);
    return;
}
```

.NET

```
byte[] documentHash = hash.Bytes().ToArray();
```

```
foreach (IIonValue proofHash in proofValue.GetElementAt(0))
{
    // Calculate the digest
    documentHash = Dot(documentHash, proofHash.Bytes().ToArray());
}

bool verified = expectedDigest.SequenceEqual(documentHash);

if (verified)
{
    Console.WriteLine($"Successfully verified document revision for id
 '{metadataId}'!");
}
else
{
    Console.WriteLine($"Document revision for id '{metadataId}' verification
 failed!");
    return;
}
```

Go

```
// Going through nodes and calculate digest
for reader.Next() {
    val, _ := reader.ByteValue()
    documentHash, err = dot(documentHash, val)
}

// Compare documentHash with the expected digest
verified := reflect.DeepEqual(documentHash, expectedDigest)

if verified {
    fmt.Printf("Successfully verified document revision for id '%s'!\n", metadataId)
} else {
    fmt.Printf("Document revision for id '%s' verification failed!\n", metadataId)
    return
}
```

Node.js

```
let documentHash: Uint8Array = hash.uInt8ArrayValue();
proofValue.elements().forEach((proofHash: dom.Value) => {
    // Calculate the digest
```

```
    documentHash = dot(documentHash, proofHash.uInt8ArrayValue());
});

let verified: boolean = isEqual(expectedDigest, documentHash);

if (verified) {
    console.log(`Successfully verified document revision for id '${metadataId}'!`);
} else {
    console.log(`Document revision for id '${metadataId}' verification failed!`);
    return;
}
```

Python

```
# Calculate digest
calculated_digest = reduce(dot, proof_hashes, document_hash)

verified = calculated_digest == expected_digest
if verified:
    print("Successfully verified document revision for id
 '{}'!".format(metadata_id))
else:
    print("Document revision for id '{}' verification failed!".format(metadata_id))
```

ステップ 5: ジャーナルブロックの証明をリクエストする

次に、ドキュメントリビジョンを含むジャーナルブロックを検証します。

ステップ 1 で保存したダイジェストのブロックアドレスとティップアドレスを使用して、GetBlock リクエストを実行します。ステップ 2 の GetRevision リクエストと同様に、保存されたダイジェストのティップアドレスを再度指定してブロックの証明を取得する必要があります。以下の API オペレーションは、ブロックとブロックの証明を含むオブジェクトを返します。

ジャーナルブロックの構造と内容の詳細については、「Amazon QLDB のジャーナルコンテンツ」を参照してください。

Java

```
// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest()
        .withName(ledgerName)
```

```
        .withBlockAddress(new ValueHolder().withIonText(blockAddressText))
        .withDigestTipAddress(digestResult.getDigestTipAddress());

// Get a result back
GetBlockResult getBlockResult = client.getBlock(getBlockRequest);
```

.NET

```
// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest
{
    Name = ledgerName,
    BlockAddress = new ValueHolder
    {
        IonText = blockAddressText
    },
    DigestTipAddress = getDigestResponse.DigestTipAddress
};

// Get a response back
GetBlockResponse getBlockResponse = client.GetBlockAsync(getBlockRequest).Result;
```

Go

```
// Submit a request for the block
blockInput := qldb.GetBlockInput{
    Name:             &currentLedgerName,
    BlockAddress:     &qldb.ValueHolder{IonText: &blockAddress},
    DigestTipAddress: digestOutput.DigestTipAddress,
}

// Get a result back
blockOutput, err := client.GetBlock(&blockInput)
if err != nil {
    panic(err)
}
```

Node.js

```
// Submit a request for the block
const getBlockRequest: GetBlockRequest = {
    Name: ledgerName,
    BlockAddress: {
```

```
        IonText: dumpText(blockAddress)
    },
    DigestTipAddress: getDigestResponse.DigestTipAddress
};

// Get a response back
const getBlockResponse: GetBlockResponse = await
 qldbClient.getBlock(getBlockRequest).promise();
```

Python

```
def block_address_to_dictionary(ion_dict):
"""
Convert a block address from IonPyDict into a dictionary.
Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">, sequenceNo:
 <sequenceNo>}"}

:type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
:param ion_dict: The block address value to convert.

:rtype: dict
:return: The converted dict.
"""
block_address = {'IonText': {}}
if not isinstance(ion_dict, str):
    py_dict = '{{strandId: "{}", sequenceNo:{}}}'.format(ion_dict['strandId'],
 ion_dict['sequenceNo'])
    ion_dict = py_dict
block_address['IonText'] = ion_dict
return block_address

# Submit a request for the block and get a result back
block_response = qldb_client.get_block(Name=ledger_name,
 BlockAddress=block_address_to_dictionary(block_address),
                                        DigestTipAddress=digest_tip_address)
```

次に、結果からブロックハッシュと証明を取得します。

Java

次の例では、IonLoader を使用してブロックオブジェクトを IonDatagram コンテナにロードします。

```
String blockText = getBlockResult.getBlock().getIonText();

IonDatagram datagram = SYSTEM.getLoader().load(blockText);
ionStruct = (IonStruct)datagram.get(0);

final byte[] blockHash = ((IonBlob)ionStruct.get("blockHash")).getBytes();
```

また、IonLoader を使用して証明を IonDatagram にロードします。

```
proofText = getBlockResult.getProof().getIonText();

// Take the proof and create a list of hash binary data
datagram = SYSTEM.getLoader().load(proofText);
ListIterator<IonValue> listIter =
 ((IonList)datagram.iterator().next()).listIterator();

internalHashes.clear();
while (listIter.hasNext()) {
    internalHashes.add(((IonBlob)listIter.next()).getBytes());
}
```

.NET

次の例では、IonLoader を使用してブロックと証明をそれぞれ Ion データグラムにロードします。

```
string blockText = getBlockResponse.Block.IonText;
IIonDatagram blockValue = IonLoader.Default.Load(blockText);

// blockValue is a IonDatagram, and the first value is an IonStruct containing the
 blockHash
byte[] blockHash =
 blockValue.GetElementAt(0).GetField("blockHash").Bytes().ToArray();

proofText = getBlockResponse.Proof.IonText;
proofValue = IonLoader.Default.Load(proofText);
```

Go

次の例では、Ion リーダーを使用して証明をバイナリに変換し、証明のノードハッシュのリストを反復処理します。

```
proofText = blockOutput.Proof.IonText

block := new(map[string]interface{})
err = ion.UnmarshalString(*blockOutput.Block.IonText, block)
if err != nil {
    panic(err)
}

blockHash := (*block)["blockHash"].([]byte)

// Use ion.Reader to iterate over the proof's node hashes
reader = ion.NewReaderString(*proofText)
// Enter the struct containing node hashes
reader.Next()
if err := reader.StepIn(); err != nil {
    panic(err)
}
```

Node.js

次の例では、load 関数を使用してブロックと証明をバイナリに変換します。

```
const blockValue: dom.Value = load(getBlockResponse.Block.IonText)
let blockHash: Uint8Array = blockValue.get("blockHash").uInt8ArrayValue();

proofValue = load(getBlockResponse.Proof.IonText);
```

Python

次の例では、loads 関数を使用してブロックと証明をバイナリに変換します。

```
block_text = block_response.get('Block').get('IonText')
block = loads(block_text)

block_hash = block.get('blockHash')

proof_text = block_response.get('Proof').get('IonText')
proof_hashes = loads(proof_text)
```

ステップ 6: ブロックのダイジェストを再計算する

証明のハッシュリストを使用して、ダイジェストを再計算します。これは、ブロックハッシュから始めます。過去に保存されたダイジェストがわかっており、QLDB の外部で信頼されている限り、ブロックの整合性は、再計算したダイジェストハッシュが保存されているダイジェストハッシュと一致した場合に証明されます。

Java

```
// Calculate digest
calculatedDigest = internalHashes.stream().reduce(blockHash,
 BlockHashVerification::dot);

verified = Arrays.equals(expectedDigest, calculatedDigest);

if (verified) {
    System.out.printf("Block address '%s' successfully verified!%n",
 blockAddressText);
} else {
    System.out.printf("Block address '%s' verification failed!%n",
 blockAddressText);
}
```

.NET

```
foreach (IIonValue proofHash in proofValue.GetElementAt(0))
{
    // Calculate the digest
    blockHash = Dot(blockHash, proofHash.Bytes().ToArray());
}

verified = expectedDigest.SequenceEqual(blockHash);

if (verified)
{
    Console.WriteLine($"Block address '{blockAddressText}' successfully verified!");
}
else
{
    Console.WriteLine($"Block address '{blockAddressText}' verification failed!");
}
```

Go

```
// Going through nodes and calculate digest
for reader.Next() {
    val, err := reader.ByteValue()
    if err != nil {
        panic(err)
    }
    blockHash, err = dot(blockHash, val)
}

// Compare blockHash with the expected digest
verified = reflect.DeepEqual(blockHash, expectedDigest)

if verified {
    fmt.Printf("Block address '%s' successfully verified!\n", blockAddress)
} else {
    fmt.Printf("Block address '%s' verification failed!\n", blockAddress)
    return
}
```

Node.js

```
proofValue.elements().forEach((proofHash: dom.Value) => {
    // Calculate the digest
    blockHash = dot(blockHash, proofHash.uInt8ArrayValue());
});

verified = isEqual(expectedDigest, blockHash);

if (verified) {
    console.log(`Block address '${dumpText(blockAddress)}' successfully verified!`);
} else {
    console.log(`Block address '${dumpText(blockAddress)}' verification failed!`);
}
```

Python

```
# Calculate digest
calculated_digest = reduce(dot, proof_hashes, block_hash)

verified = calculated_digest == expected_digest
if verified:
```

```
    print("Block address '{}' successfully verified!".format(dumps(block_address,
                                                                   binary=False,

 omit_version_marker=True)))
else:
    print("Block address '{}' verification failed!".format(block_address))
```

上のコード例では、ダイジェストの再計算の際に以下の dot 関数を使用しています。この関数に 2 つのハッシュを入力すると、それらを並べ替え、連結して、連結された配列のハッシュを返します。

Java

```
/**
 * Takes two hashes, sorts them, concatenates them, and then returns the
 * hash of the concatenated array.
 *
 * @param h1
 *              Byte array containing one of the hashes to compare.
 * @param h2
 *              Byte array containing one of the hashes to compare.
 * @return the concatenated array of hashes.
 */
public static byte[] dot(final byte[] h1, final byte[] h2) {
    if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
        throw new IllegalArgumentException("Invalid hash.");
    }

    int byteEqual = 0;
    for (int i = h1.length - 1; i >= 0; i--) {
        byteEqual = Byte.compare(h1[i], h2[i]);
        if (byteEqual != 0) {
            break;
        }
    }

    byte[] concatenated = new byte[h1.length + h2.length];
    if (byteEqual < 0) {
        System.arraycopy(h1, 0, concatenated, 0, h1.length);
        System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
    } else {
        System.arraycopy(h2, 0, concatenated, 0, h2.length);
        System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
```

```
    }

    MessageDigest messageDigest;
    try {
        messageDigest = MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        throw new IllegalStateException("SHA-256 message digest is unavailable", e);
    }

    messageDigest.update(concatenated);
    return messageDigest.digest();
}
```

.NET

```
/// <summary>
/// Takes two hashes, sorts them, concatenates them, and then returns the
/// hash of the concatenated array.
/// </summary>
/// <param name="h1">Byte array containing one of the hashes to compare.</param>
/// <param name="h2">Byte array containing one of the hashes to compare.</param>
/// <returns>The concatenated array of hashes.</returns>
private static byte[] Dot(byte[] h1, byte[] h2)
{
    if (h1.Length == 0)
    {
        return h2;
    }

    if (h2.Length == 0)
    {
        return h1;
    }

    HashAlgorithm hashAlgorithm = HashAlgorithm.Create("SHA256");
    HashComparer comparer = new HashComparer();
    if (comparer.Compare(h1, h2) < 0)
    {
        return hashAlgorithm.ComputeHash(h1.Concat(h2).ToArray());
    }
    else
    {
        return hashAlgorithm.ComputeHash(h2.Concat(h1).ToArray());
```

```
    }
}

private class HashComparer : IComparer<byte[]>
{
    private static readonly int HASH_LENGTH = 32;

    public int Compare(byte[] h1, byte[] h2)
    {
        if (h1.Length != HASH_LENGTH || h2.Length != HASH_LENGTH)
        {
            throw new ArgumentException("Invalid hash");
        }

        for (var i = h1.Length - 1; i >= 0; i--)
        {
            var byteEqual = (sbyte)h1[i] - (sbyte)h2[i];
            if (byteEqual != 0)
            {
                return byteEqual;
            }
        }

        return 0;
    }
}
```

Go

```
// Takes two hashes, sorts them, concatenates them, and then returns the hash of the
 concatenated array.
func dot(h1, h2 []byte) ([]byte, error) {
    compare, err := hashComparator(h1, h2)
    if err != nil {
        return nil, err
    }

    var concatenated []byte
    if compare < 0 {
        concatenated = append(h1, h2...)
    } else {
        concatenated = append(h2, h1...)
    }
```

```
    newHash := sha256.Sum256(concatenated)
    return newHash[:], nil
}

func hashComparator(h1 []byte, h2 []byte) (int16, error) {
    if len(h1) != hashLength || len(h2) != hashLength {
        return 0, errors.New("invalid hash")
    }
    for i := range h1 {
        // Reverse index for little endianness
        index := hashLength - 1 - i

        // Handle byte being unsigned and overflow
        h1Int := int16(h1[index])
        h2Int := int16(h2[index])
        if h1Int > 127 {
            h1Int = 0 - (256 - h1Int)
        }
        if h2Int > 127 {
            h2Int = 0 - (256 - h2Int)
        }

        difference := h1Int - h2Int
        if difference != 0 {
            return difference, nil
        }
    }
    return 0, nil
}
```

Node.js

```
/**
 * Takes two hashes, sorts them, concatenates them, and calculates a digest based on
 the concatenated hash.
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The digest calculated from the concatenated hash values.
 */
function dot(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
```

```
    }
    if (h2.length === 0) {
        return h1;
    }

    const newHashLib = createHash("sha256");

    let concatenated: Uint8Array;
    if (hashComparator(h1, h2) < 0) {
        concatenated = concatenate(h1, h2);
    } else {
        concatenated = concatenate(h2, h1);
    }
    newHashLib.update(concatenated);
    return newHashLib.digest();
}

/**
 * Compares two hashes by their **signed** byte values in little-endian order.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 the first pair of non-matching
 *          bytes.
 * @throws RangeError When the hash is not the correct hash size.
 */
function hashComparator(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_SIZE || hash2.length !== HASH_SIZE) {
        throw new RangeError("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of arrays to concatenate, in the order provided.
 * @returns The concatenated array.
 */
```

```
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}

/**
 * Helper method that checks for equality between two Uint8Array.
 * @param expected Byte array containing one of the hashes to compare.
 * @param actual Byte array containing one of the hashes to compare.
 * @returns Boolean indicating equality between the two Uint8Array.
 */
function isEqual(expected: Uint8Array, actual: Uint8Array): boolean {
    if (expected === actual) return true;
    if (expected == null || actual == null) return false;
    if (expected.length !== actual.length) return false;

    for (let i = 0; i < expected.length; i++) {
        if (expected[i] !== actual[i]) {
            return false;
        }
    }
    return true;
}
```

Python

```
def dot(hash1, hash2):
    """
    Takes two hashes, sorts them, concatenates them, and then returns the
    hash of the concatenated array.

    :type hash1: bytes
    :param hash1: The hash value to compare.
```

```
    :type hash2: bytes
    :param hash2: The hash value to compare.

    :rtype: bytes
    :return: The new hash value generated from concatenated hash values.
    """
    if len(hash1) != hash_length or len(hash2) != hash_length:
        raise ValueError('Illegal hash.')

    hash_array1 = array('b', hash1)
    hash_array2 = array('b', hash2)

    difference = 0
    for i in range(len(hash_array1) - 1, -1, -1):
        difference = hash_array1[i] - hash_array2[i]
        if difference != 0:
            break

    if difference < 0:
        concatenated = hash1 + hash2
    else:
        concatenated = hash2 + hash1

    new_hash_lib = sha256()
    new_hash_lib.update(concatenated)
    new_digest = new_hash_lib.digest()
    return new_digest
```

完全なコード例を実行する

以下の完全なコード例を実行すると、前述のステップをすべて最初から最後まで実行します。

Java

```
import com.amazon.ion.IonBlob;
import com.amazon.ion.IonDatagram;
import com.amazon.ion.IonList;
import com.amazon.ion.IonReader;
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
```

```
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.GetBlockRequest;
import com.amazonaws.services.qldb.model.GetBlockResult;
import com.amazonaws.services.qldb.model.GetDigestRequest;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.ListIterator;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.awssdk.services.qldbsession.QldbSessionClientBuilder;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;

public class BlockHashVerification {
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    private static final QldbDriver driver = createQldbDriver();
    private static final AmazonQLDB client =
 AmazonQLDBClientBuilder.standard().build();
    private static final String region = "us-east-1";
    private static final String ledgerName = "vehicle-registration";
    private static final String tableName = "VehicleRegistration";
    private static final String vin = "KM8SRDHF6EU074761";
    private static final int HASH_LENGTH = 32;

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver() {
        QldbSessionClientBuilder sessionClientBuilder = QldbSessionClient.builder();
        sessionClientBuilder.region(Region.of(region));

        return QldbDriver.builder()
```

```
                .ledger(ledgerName)
                .sessionClientBuilder(sessionClientBuilder)
                .build();
    }

    /**
     * Takes two hashes, sorts them, concatenates them, and then returns the
     * hash of the concatenated array.
     *
     * @param h1
     *              Byte array containing one of the hashes to compare.
     * @param h2
     *              Byte array containing one of the hashes to compare.
     * @return the concatenated array of hashes.
     */
    public static byte[] dot(final byte[] h1, final byte[] h2) {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }

        int byteEqual = 0;
        for (int i = h1.length - 1; i >= 0; i--) {
            byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                break;
            }
        }

        byte[] concatenated = new byte[h1.length + h2.length];
        if (byteEqual < 0) {
            System.arraycopy(h1, 0, concatenated, 0, h1.length);
            System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
        } else {
            System.arraycopy(h2, 0, concatenated, 0, h2.length);
            System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
        }

        MessageDigest messageDigest;
        try {
            messageDigest = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException("SHA-256 message digest is unavailable",
e);
        }
```

```
        messageDigest.update(concatenated);
        return messageDigest.digest();
    }

    public static void main(String[] args) {
        // Get a digest
        GetDigestRequest digestRequest = new
GetDigestRequest().withName(ledgerName);
        GetDigestResult digestResult = client.getDigest(digestRequest);

        java.nio.ByteBuffer digest = digestResult.getDigest();

        // expectedDigest is the buffer we will use later to compare against our
calculated digest
        byte[] expectedDigest = new byte[digest.remaining()];
        digest.get(expectedDigest);

        // Retrieve info for the given vin's document revisions
        Result result = driver.execute(txn -> {
            final String query = String.format("SELECT blockAddress, hash,
metadata.id FROM _ql_committed_%s WHERE data.VIN = '%s'", tableName, vin);
            return txn.execute(query);
        });

        System.out.printf("Verifying document revisions for vin '%s' in table '%s'
in ledger '%s'%n", vin, tableName, ledgerName);

        for (IonValue ionValue : result) {
            IonStruct ionStruct = (IonStruct)ionValue;

            // Get the requested fields
            IonValue blockAddress = ionStruct.get("blockAddress");
            IonBlob hash = (IonBlob)ionStruct.get("hash");
            String metadataId = ((IonString)ionStruct.get("id")).stringValue();

            System.out.printf("Verifying document revision for id '%s'%n",
metadataId);

            String blockAddressText = blockAddress.toString();

            // Submit a request for the revision
            GetRevisionRequest revisionRequest = new GetRevisionRequest()
                    .withName(ledgerName)
```

```
                    .withBlockAddress(new
ValueHolder().withIonText(blockAddressText))
                    .withDocumentId(metadataId)
                    .withDigestTipAddress(digestResult.getDigestTipAddress());

            // Get a result back
            GetRevisionResult revisionResult = client.getRevision(revisionRequest);

            String proofText = revisionResult.getProof().getIonText();

            // Take the proof and convert it to a list of byte arrays
            List<byte[]> internalHashes = new ArrayList<>();
            IonReader reader = SYSTEM.newReader(proofText);
            reader.next();
            reader.stepIn();
            while (reader.next() != null) {
                internalHashes.add(reader.newBytes());
            }

            // Calculate digest
            byte[] calculatedDigest =
internalHashes.stream().reduce(hash.getBytes(), BlockHashVerification::dot);

            boolean verified = Arrays.equals(expectedDigest, calculatedDigest);

            if (verified) {
                System.out.printf("Successfully verified document revision for id
'%s'!%n", metadataId);
            } else {
                System.out.printf("Document revision for id '%s' verification
failed!%n", metadataId);
                return;
            }

            // Submit a request for the block
            GetBlockRequest getBlockRequest = new GetBlockRequest()
                    .withName(ledgerName)
                    .withBlockAddress(new
ValueHolder().withIonText(blockAddressText))
                    .withDigestTipAddress(digestResult.getDigestTipAddress());

            // Get a result back
            GetBlockResult getBlockResult = client.getBlock(getBlockRequest);
```

```
            String blockText = getBlockResult.getBlock().getIonText();

            IonDatagram datagram = SYSTEM.getLoader().load(blockText);
            ionStruct = (IonStruct)datagram.get(0);

            final byte[] blockHash =
 ((IonBlob)ionStruct.get("blockHash")).getBytes();

            proofText = getBlockResult.getProof().getIonText();

            // Take the proof and create a list of hash binary data
            datagram = SYSTEM.getLoader().load(proofText);
            ListIterator<IonValue> listIter =
 ((IonList)datagram.iterator().next()).listIterator();

            internalHashes.clear();
            while (listIter.hasNext()) {
                internalHashes.add(((IonBlob)listIter.next()).getBytes());
            }

            // Calculate digest
            calculatedDigest = internalHashes.stream().reduce(blockHash,
 BlockHashVerification::dot);

            verified = Arrays.equals(expectedDigest, calculatedDigest);

            if (verified) {
                System.out.printf("Block address '%s' successfully verified!%n",
 blockAddressText);
            } else {
                System.out.printf("Block address '%s' verification failed!%n",
 blockAddressText);
            }
        }
    }
}
```

.NET

```
using Amazon.IonDotnet;
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB;
```

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;

namespace BlockHashVerification
{
    class BlockHashVerification
    {
        private static readonly string ledgerName = "vehicle-registration";
        private static readonly string tableName = "VehicleRegistration";
        private static readonly string vin = "KM8SRDHF6EU074761";
        private static readonly IQldbDriver driver =
 QldbDriver.Builder().WithLedger(ledgerName).Build();
        private static readonly IAmazonQLDB client = new AmazonQLDBClient();

        /// <summary>
        /// Takes two hashes, sorts them, concatenates them, and then returns the
        /// hash of the concatenated array.
        /// </summary>
        /// <param name="h1">Byte array containing one of the hashes to compare.</
param>
        /// <param name="h2">Byte array containing one of the hashes to compare.</
param>
        /// <returns>The concatenated array of hashes.</returns>
        private static byte[] Dot(byte[] h1, byte[] h2)
        {
            if (h1.Length == 0)
            {
                return h2;
            }

            if (h2.Length == 0)
            {
                return h1;
            }

            HashAlgorithm hashAlgorithm = HashAlgorithm.Create("SHA256");
            HashComparer comparer = new HashComparer();
            if (comparer.Compare(h1, h2) < 0)
            {
```

```
                return hashAlgorithm.ComputeHash(h1.Concat(h2).ToArray());
            }
            else
            {
                return hashAlgorithm.ComputeHash(h2.Concat(h1).ToArray());
            }
        }

        private class HashComparer : IComparer<byte[]>
        {
            private static readonly int HASH_LENGTH = 32;

            public int Compare(byte[] h1, byte[] h2)
            {
                if (h1.Length != HASH_LENGTH || h2.Length != HASH_LENGTH)
                {
                    throw new ArgumentException("Invalid hash");
                }

                for (var i = h1.Length - 1; i >= 0; i--)
                {
                    var byteEqual = (sbyte)h1[i] - (sbyte)h2[i];
                    if (byteEqual != 0)
                    {
                        return byteEqual;
                    }
                }

                return 0;
            }
        }

        static void Main()
        {
            // Get a digest
            GetDigestRequest getDigestRequest = new GetDigestRequest
            {
                Name = ledgerName
            };
            GetDigestResponse getDigestResponse =
    client.GetDigestAsync(getDigestRequest).Result;

            // expectedDigest is the buffer we will use later to compare against our
    calculated digest
```

```
            MemoryStream digest = getDigestResponse.Digest;
            byte[] expectedDigest = digest.ToArray();

            // Retrieve info for the given vin's document revisions
            var result = driver.Execute(txn => {
                string query = $"SELECT blockAddress, hash, metadata.id FROM
_ql_committed_{tableName} WHERE data.VIN = '{vin}'";
                return txn.Execute(query);
            });

            Console.WriteLine($"Verifying document revisions for vin '{vin}' in
table '{tableName}' in ledger '{ledgerName}'");

            foreach (IIonValue ionValue in result)
            {
                IIonStruct ionStruct = ionValue;

                // Get the requested fields
                IIonValue blockAddress = ionStruct.GetField("blockAddress");
                IIonBlob hash = ionStruct.GetField("hash");
                String metadataId = ionStruct.GetField("id").StringValue;

                Console.WriteLine($"Verifying document revision for id
'{metadataId}'");

                // Use an Ion Reader to convert block address to text
                IIonReader reader = IonReaderBuilder.Build(blockAddress);
                StringWriter sw = new StringWriter();
                IIonWriter textWriter = IonTextWriterBuilder.Build(sw);
                textWriter.WriteValues(reader);
                string blockAddressText = sw.ToString();

                // Submit a request for the revision
                GetRevisionRequest revisionRequest = new GetRevisionRequest
                {
                    Name = ledgerName,
                    BlockAddress = new ValueHolder
                    {
                        IonText = blockAddressText
                    },
                    DocumentId = metadataId,
                    DigestTipAddress = getDigestResponse.DigestTipAddress
                };
```

```
                // Get a response back
                GetRevisionResponse revisionResponse =
client.GetRevisionAsync(revisionRequest).Result;

                string proofText = revisionResponse.Proof.IonText;
                IIonDatagram proofValue = IonLoader.Default.Load(proofText);

                byte[] documentHash = hash.Bytes().ToArray();
                foreach (IIonValue proofHash in proofValue.GetElementAt(0))
                {
                    // Calculate the digest
                    documentHash = Dot(documentHash, proofHash.Bytes().ToArray());
                }

                bool verified = expectedDigest.SequenceEqual(documentHash);

                if (verified)
                {
                    Console.WriteLine($"Successfully verified document revision for
id '{metadataId}'!");
                }
                else
                {
                    Console.WriteLine($"Document revision for id '{metadataId}'
verification failed!");
                    return;
                }

                // Submit a request for the block
                GetBlockRequest getBlockRequest = new GetBlockRequest
                {
                    Name = ledgerName,
                    BlockAddress = new ValueHolder
                    {
                        IonText = blockAddressText
                    },
                    DigestTipAddress = getDigestResponse.DigestTipAddress
                };

                // Get a response back
                GetBlockResponse getBlockResponse =
client.GetBlockAsync(getBlockRequest).Result;

                string blockText = getBlockResponse.Block.IonText;
```

```
                IIonDatagram blockValue = IonLoader.Default.Load(blockText);

                // blockValue is a IonDatagram, and the first value is an IonStruct
 containing the blockHash
                byte[] blockHash =
 blockValue.GetElementAt(0).GetField("blockHash").Bytes().ToArray();

                proofText = getBlockResponse.Proof.IonText;
                proofValue = IonLoader.Default.Load(proofText);

                foreach (IIonValue proofHash in proofValue.GetElementAt(0))
                {
                    // Calculate the digest
                    blockHash = Dot(blockHash, proofHash.Bytes().ToArray());
                }

                verified = expectedDigest.SequenceEqual(blockHash);

                if (verified)
                {
                    Console.WriteLine($"Block address '{blockAddressText}'
 successfully verified!");
                }
                else
                {
                    Console.WriteLine($"Block address '{blockAddressText}'
 verification failed!");
                }
            }
        }
    }
}
```

Go

```
package main

import (
    "context"
    "crypto/sha256"
    "errors"
    "fmt"
    "reflect"
```

```
    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    AWSSession "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldb"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qldb-driver-go/qldbdriver"
)

const (
    hashLength = 32
    ledgerName = "vehicle-registration"
    tableName  = "VehicleRegistration"
    vin        = "KM8SRDHF6EU074761"
)

// Takes two hashes, sorts them, concatenates them, and then returns the hash of the
 concatenated array.
func dot(h1, h2 []byte) ([]byte, error) {
    compare, err := hashComparator(h1, h2)
    if err != nil {
        return nil, err
    }

    var concatenated []byte
    if compare < 0 {
        concatenated = append(h1, h2...)
    } else {
        concatenated = append(h2, h1...)
    }

    newHash := sha256.Sum256(concatenated)
    return newHash[:], nil
}

func hashComparator(h1 []byte, h2 []byte) (int16, error) {
    if len(h1) != hashLength || len(h2) != hashLength {
        return 0, errors.New("invalid hash")
    }
    for i := range h1 {
        // Reverse index for little endianness
        index := hashLength - 1 - i

        // Handle byte being unsigned and overflow
```

```
        h1Int := int16(h1[index])
        h2Int := int16(h2[index])
        if h1Int > 127 {
            h1Int = 0 - (256 - h1Int)
        }
        if h2Int > 127 {
            h2Int = 0 - (256 - h2Int)
        }

        difference := h1Int - h2Int
        if difference != 0 {
            return difference, nil
        }
    }
    return 0, nil
}

func main() {
    driverSession := AWSSession.Must(AWSSession.NewSession(aws.NewConfig()))
    qldbSession := qldbsession.New(driverSession)
    driver, err := qldbdriver.New(ledgerName, qldbSession, func(options
 *qldbdriver.DriverOptions) {})
    if err != nil {
        panic(err)
    }
    client := qldb.New(driverSession)

    // Get a digest
    currentLedgerName := ledgerName
    input := qldb.GetDigestInput{Name: &currentLedgerName}
    digestOutput, err := client.GetDigest(&input)
    if err != nil {
        panic(err)
    }

    // expectedDigest is the buffer we will later use to compare against our
 calculated digest
    expectedDigest := digestOutput.Digest

    // Retrieve info for the given vin's document revisions
    result, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
        statement := fmt.Sprintf(
```

```
                "SELECT blockAddress, hash, metadata.id FROM _ql_committed_%s WHERE
data.VIN = '%s'",
                tableName,
                vin)
        result, err := txn.Execute(statement)
        if err != nil {
            return nil, err
        }

        results := make([]map[string]interface{}, 0)

        // Convert the result set into a map
        for result.Next(txn) {
            var doc map[string]interface{}
            err := ion.Unmarshal(result.GetCurrentData(), &doc)
            if err != nil {
                return nil, err
            }
            results = append(results, doc)
        }
        return results, nil
    })
    if err != nil {
        panic(err)
    }
    resultSlice := result.([]map[string]interface{})

    fmt.Printf("Verifying document revisions for vin '%s' in table '%s' in ledger
'%s'\n", vin, tableName, ledgerName)

    for _, value := range resultSlice {
        // Get the requested fields
        ionBlockAddress, err := ion.MarshalText(value["blockAddress"])
        if err != nil {
            panic(err)
        }
        blockAddress := string(ionBlockAddress)
        metadataId := value["id"].(string)
        documentHash := value["hash"].([]byte)

        fmt.Printf("Verifying document revision for id '%s'\n", metadataId)

        // Submit a request for the revision
        revisionInput := qldb.GetRevisionInput{
```

```
            BlockAddress:     &qldb.ValueHolder{IonText: &blockAddress},
            DigestTipAddress: digestOutput.DigestTipAddress,
            DocumentId:       &metadataId,
            Name:             &currentLedgerName,
        }

        // Get a result back
        revisionOutput, err := client.GetRevision(&revisionInput)
        if err != nil {
            panic(err)
        }

        proofText := revisionOutput.Proof.IonText

        // Use ion.Reader to iterate over the proof's node hashes
        reader := ion.NewReaderString(*proofText)
        // Enter the struct containing node hashes
        reader.Next()
        if err := reader.StepIn(); err != nil {
            panic(err)
        }

        // Going through nodes and calculate digest
        for reader.Next() {
            val, _ := reader.ByteValue()
            documentHash, err = dot(documentHash, val)
        }

        // Compare documentHash with the expected digest
        verified := reflect.DeepEqual(documentHash, expectedDigest)

        if verified {
            fmt.Printf("Successfully verified document revision for id '%s'!\n",
metadataId)
        } else {
            fmt.Printf("Document revision for id '%s' verification failed!\n",
metadataId)
            return
        }

        // Submit a request for the block
        blockInput := qldb.GetBlockInput{
            Name:             &currentLedgerName,
            BlockAddress:     &qldb.ValueHolder{IonText: &blockAddress},
```

```
            DigestTipAddress: digestOutput.DigestTipAddress,
        }

        // Get a result back
        blockOutput, err := client.GetBlock(&blockInput)
        if err != nil {
            panic(err)
        }

        proofText = blockOutput.Proof.IonText

        block := new(map[string]interface{})
        err = ion.UnmarshalString(*blockOutput.Block.IonText, block)
        if err != nil {
            panic(err)
        }

        blockHash := (*block)["blockHash"].([]byte)

        // Use ion.Reader to iterate over the proof's node hashes
        reader = ion.NewReaderString(*proofText)
        // Enter the struct containing node hashes
        reader.Next()
        if err := reader.StepIn(); err != nil {
            panic(err)
        }

        // Going through nodes and calculate digest
        for reader.Next() {
            val, err := reader.ByteValue()
            if err != nil {
                panic(err)
            }
            blockHash, err = dot(blockHash, val)
        }

        // Compare blockHash with the expected digest
        verified = reflect.DeepEqual(blockHash, expectedDigest)

        if verified {
            fmt.Printf("Block address '%s' successfully verified!\n", blockAddress)
        } else {
            fmt.Printf("Block address '%s' verification failed!\n", blockAddress)
            return
```

```
        }
    }
}
```

Node.js

```
import { QldbDriver, Result, TransactionExecutor} from "amazon-qldb-driver-nodejs";
import { QLDB } from "aws-sdk"
import { GetBlockRequest, GetBlockResponse, GetDigestRequest, GetDigestResponse,
 GetRevisionRequest, GetRevisionResponse } from "aws-sdk/clients/qldb";
import { createHash } from "crypto";
import { dom, dumpText, load } from "ion-js"

const ledgerName: string = "vehicle-registration";
const tableName: string = "VehicleRegistration";
const vin: string = "KM8SRDHF6EU074761";
const driver: QldbDriver = new QldbDriver(ledgerName);
const qldbClient: QLDB = new QLDB();
const HASH_SIZE = 32;

/**
 * Takes two hashes, sorts them, concatenates them, and calculates a digest based on
 the concatenated hash.
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The digest calculated from the concatenated hash values.
 */
function dot(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }

    const newHashLib = createHash("sha256");

    let concatenated: Uint8Array;
    if (hashComparator(h1, h2) < 0) {
        concatenated = concatenate(h1, h2);
    } else {
        concatenated = concatenate(h2, h1);
    }
```

```
    newHashLib.update(concatenated);
    return newHashLib.digest();
}

/**
 * Compares two hashes by their **signed** byte values in little-endian order.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 the first pair of non-matching
 *          bytes.
 * @throws RangeError When the hash is not the correct hash size.
 */
function hashComparator(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_SIZE || hash2.length !== HASH_SIZE) {
        throw new RangeError("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of arrays to concatenate, in the order provided.
 * @returns The concatenated array.
 */
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}
```

```
/**
 * Helper method that checks for equality between two Uint8Array.
 * @param expected Byte array containing one of the hashes to compare.
 * @param actual Byte array containing one of the hashes to compare.
 * @returns Boolean indicating equality between the two Uint8Array.
 */
function isEqual(expected: Uint8Array, actual: Uint8Array): boolean {
    if (expected === actual) return true;
    if (expected == null || actual == null) return false;
    if (expected.length !== actual.length) return false;

    for (let i = 0; i < expected.length; i++) {
        if (expected[i] !== actual[i]) {
            return false;
        }
    }
    return true;
}

const main = async function (): Promise<void> {
    // Get a digest
    const getDigestRequest: GetDigestRequest = {
        Name: ledgerName
    };
    const getDigestResponse: GetDigestResponse = await
 qldbClient.getDigest(getDigestRequest).promise();

    // expectedDigest is the buffer we will later use to compare against our
 calculated digest
    const expectedDigest: Uint8Array = <Uint8Array>getDigestResponse.Digest;

    const result: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor): Promise<dom.Value[]> => {
        const query: string = `SELECT blockAddress, hash, metadata.id FROM
 _ql_committed_${tableName} WHERE data.VIN = '${vin}'`;
        const queryResult: Result = await txn.execute(query);
        return queryResult.getResultList();
    });

    console.log(`Verifying document revisions for vin '${vin}' in table
 '${tableName}' in ledger '${ledgerName}'`);

    for (let value of result) {
```

```
        // Get the requested fields
        const blockAddress: dom.Value = value.get("blockAddress");
        const hash: dom.Value = value.get("hash");
        const metadataId: string = value.get("id").stringValue();

        console.log(`Verifying document revision for id '${metadataId}'`);

        // Submit a request for the revision
        const revisionRequest: GetRevisionRequest = {
            Name: ledgerName,
            BlockAddress: {
                IonText: dumpText(blockAddress)
            },
            DocumentId: metadataId,
            DigestTipAddress: getDigestResponse.DigestTipAddress
        };

        // Get a response back
        const revisionResponse: GetRevisionResponse = await
qldbClient.getRevision(revisionRequest).promise();

        let proofValue: dom.Value = load(revisionResponse.Proof.IonText);

        let documentHash: Uint8Array = hash.uInt8ArrayValue();
        proofValue.elements().forEach((proofHash: dom.Value) => {
            // Calculate the digest
            documentHash = dot(documentHash, proofHash.uInt8ArrayValue());
        });

        let verified: boolean = isEqual(expectedDigest, documentHash);

        if (verified) {
            console.log(`Successfully verified document revision for id
'${metadataId}'!`);
        } else {
            console.log(`Document revision for id '${metadataId}' verification
failed!`);
            return;
        }

        // Submit a request for the block
        const getBlockRequest: GetBlockRequest = {
            Name: ledgerName,
            BlockAddress: {
```

```
                IonText: dumpText(blockAddress)
            },
            DigestTipAddress: getDigestResponse.DigestTipAddress
        };

        // Get a response back
        const getBlockResponse: GetBlockResponse = await
 qldbClient.getBlock(getBlockRequest).promise();

        const blockValue: dom.Value = load(getBlockResponse.Block.IonText)
        let blockHash: Uint8Array = blockValue.get("blockHash").uInt8ArrayValue();

        proofValue = load(getBlockResponse.Proof.IonText);

        proofValue.elements().forEach((proofHash: dom.Value) => {
            // Calculate the digest
            blockHash = dot(blockHash, proofHash.uInt8ArrayValue());
        });

        verified = isEqual(expectedDigest, blockHash);

        if (verified) {
            console.log(`Block address '${dumpText(blockAddress)}' successfully
 verified!`);
        } else {
            console.log(`Block address '${dumpText(blockAddress)}' verification
 failed!`);
        }
    }
};

if (require.main === module) {
    main();
}
```

Python

```
from amazon.ion.simpleion import dumps, loads
from array import array
from boto3 import client
from functools import reduce
from hashlib import sha256
from pyqldb.driver.qldb_driver import QldbDriver
```

```
ledger_name = 'vehicle-registration'
table_name = 'VehicleRegistration'
vin = 'KM8SRDHF6EU074761'
qldb_client = client('qldb')
hash_length = 32


def query_doc_revision(txn):
    query = "SELECT blockAddress, hash, metadata.id FROM _ql_committed_{} WHERE
 data.VIN = '{}'".format(table_name, vin)
    return txn.execute_statement(query)


def block_address_to_dictionary(ion_dict):
    """
    Convert a block address from IonPyDict into a dictionary.
    Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">,
 sequenceNo: <sequenceNo>}"}

    :type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
    :param ion_dict: The block address value to convert.

    :rtype: dict
    :return: The converted dict.
    """
    block_address = {'IonText': {}}
    if not isinstance(ion_dict, str):
        py_dict = '{{strandId: "{}", sequenceNo:{}}}'.format(ion_dict['strandId'],
 ion_dict['sequenceNo'])
        ion_dict = py_dict
    block_address['IonText'] = ion_dict
    return block_address


def dot(hash1, hash2):
    """
    Takes two hashes, sorts them, concatenates them, and then returns the
    hash of the concatenated array.

    :type hash1: bytes
    :param hash1: The hash value to compare.

    :type hash2: bytes
```

```
    :param hash2: The hash value to compare.

    :rtype: bytes
    :return: The new hash value generated from concatenated hash values.
    """
    if len(hash1) != hash_length or len(hash2) != hash_length:
        raise ValueError('Illegal hash.')

    hash_array1 = array('b', hash1)
    hash_array2 = array('b', hash2)

    difference = 0
    for i in range(len(hash_array1) - 1, -1, -1):
        difference = hash_array1[i] - hash_array2[i]
        if difference != 0:
            break

    if difference < 0:
        concatenated = hash1 + hash2
    else:
        concatenated = hash2 + hash1

    new_hash_lib = sha256()
    new_hash_lib.update(concatenated)
    new_digest = new_hash_lib.digest()
    return new_digest


# Get a digest
get_digest_response = qldb_client.get_digest(Name=ledger_name)

# expected_digest is the buffer we will later use to compare against our calculated
 digest
expected_digest = get_digest_response.get('Digest')
digest_tip_address = get_digest_response.get('DigestTipAddress')

qldb_driver = QldbDriver(ledger_name=ledger_name)

# Retrieve info for the given vin's document revisions
result = qldb_driver.execute_lambda(query_doc_revision)

print("Verifying document revisions for vin '{}' in table '{}' in ledger
 '{}'".format(vin, table_name, ledger_name))
```

```
for value in result:
    # Get the requested fields
    block_address = value['blockAddress']
    document_hash = value['hash']
    metadata_id = value['id']

    print("Verifying document revision for id '{}'".format(metadata_id))

    # Submit a request for the revision and get a result back
    proof_response = qldb_client.get_revision(Name=ledger_name,

 BlockAddress=block_address_to_dictionary(block_address),
                                              DocumentId=metadata_id,
                                              DigestTipAddress=digest_tip_address)

    proof_text = proof_response.get('Proof').get('IonText')
    proof_hashes = loads(proof_text)

    # Calculate digest
    calculated_digest = reduce(dot, proof_hashes, document_hash)

    verified = calculated_digest == expected_digest
    if verified:
        print("Successfully verified document revision for id
 '{}'!".format(metadata_id))
    else:
        print("Document revision for id '{}' verification
 failed!".format(metadata_id))

    # Submit a request for the block and get a result back
    block_response = qldb_client.get_block(Name=ledger_name,
 BlockAddress=block_address_to_dictionary(block_address),
                                           DigestTipAddress=digest_tip_address)

    block_text = block_response.get('Block').get('IonText')
    block = loads(block_text)

    block_hash = block.get('blockHash')

    proof_text = block_response.get('Proof').get('IonText')
    proof_hashes = loads(proof_text)

    # Calculate digest
    calculated_digest = reduce(dot, proof_hashes, block_hash)
```

```
    verified = calculated_digest == expected_digest
    if verified:
        print("Block address '{}' successfully
verified!".format(dumps(block_address,
                                                                binary=False,

omit_version_marker=True)))
    else:
        print("Block address '{}' verification failed!".format(block_address))
```

検証の一般的なエラー

このセクションでは、検証リクエストに関して Amazon QLDB がスローするランタイムエラーについて説明します。

以下は、サービスによって返される一般的な例外のリストです。各例外には、固有のエラーメッセージ、エラーをスローする可能性のある API オペレーション、簡単な説明、および考えられる解決策の案が含まれています。

IllegalArgumentException

メッセージ: 提供された Ion 値は無効なため解析できません。

API オペレーション: `GetDigest`, `GetBlock`, `GetRevision`

リクエストを再試行する前に、有効な Amazon Ion 値が指定されているか確認してください。

IllegalArgumentException

メッセージ: 指定されたブロックアドレスは無効です。

API オペレーション: `GetDigest`, `GetBlock`, `GetRevision`

リクエストを再試行する前に、有効なブロックアドレスが指定されていることを確認してください。ブロックアドレスは、`strandId` と `sequenceNo` という 2 つのフィールドを含む Amazon Ion 構造です。

例: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`

IllegalArgumentException

メッセージ: The sequence number of the provided digest tip address is beyond the strand's latest committed record. (指定されたダイジェストティップアドレスのシーケンス番号が、ストランドの最新のコミット済みレコードを超えています。)

API オペレーション: `GetDigest, GetBlock, GetRevision`

指定するダイジェストティップアドレスのシーケンス番号は、ジャーナルストランドの最新のコミット済みレコードのシーケンス番号以下でなければなりません。リクエストを再試行する前に、ダイジェストティップアドレスに有効なシーケンス番号を指定していることを確認してください。

IllegalArgumentException

メッセージ: The Strand ID of the provided block address is not valid. (指定されたブロックアドレスのストランド ID が無効です。)

API オペレーション: `GetDigest, GetBlock, GetRevision`

指定するブロックアドレスのストランド ID は、ジャーナルのストランド ID に一致するものでなければなりません。リクエストを再試行する前に、ブロックアドレスに有効なストランド ID を指定していることを確認してください。

IllegalArgumentException

メッセージ: The sequence number of the provided block address is beyond the strand's latest committed record. (指定されたブロックアドレスのシーケンス番号が、ストランドの最新のコミット済みレコードを超えています。)

API オペレーション: `GetBlock, GetRevision`

指定するブロックアドレスのシーケンス番号は、ストランドの最新のコミット済みレコードのシーケンス番号以下でなければなりません。リクエストを再試行する前に、有効なシーケンス番号を持つブロックアドレスが指定されているか確認してください。

IllegalArgumentException

メッセージ: 指定されたブロックアドレスのストランド ID は、指定されたダイジェストティップアドレスのストランド ID と一致するものでなければなりません。

API オペレーション: `GetBlock, GetRevision`

ドキュメントリビジョンまたはブロックは、指定するダイジェストと同じジャーナルストランドに存在する場合に限り検証できます。

IllegalArgumentException

メッセージ: 指定されたブロックアドレスのシーケンス番号は、指定されたダイジェストティップアドレスのシーケンス番号以下でなければなりません。

API オペレーション: `GetBlock, GetRevision`

ドキュメントリビジョンまたはブロックは、指定されたダイジェストでカバーされている場合に限り検証できます。言い換えると、ダイジェストティップアドレスより前のジャーナルにコミットされている場合に限り検証できます。

IllegalArgumentException

メッセージ: 指定されたドキュメント ID が、指定されたブロックアドレスのブロックに存在しません。

API オペレーション: `GetRevision`

指定するドキュメント ID は、指定するブロック内に存在するものでなければなりません。リクエストを再試行する前に、これら 2 つのパラメータが一貫しているか確認してください。

Amazon QLDB からのジャーナルデータのエクスポート

Amazon QLDB では、イミュータブルトランザクションログ (ジャーナル) をデータストレージに使用します。ジャーナルは、コミット済みデータへの変更をすべて追跡し、完全かつ検証可能な変更履歴を一定の期間にわたって維持します。

台帳のジャーナルのコンテンツには、分析、監査、データ保持、検証、他のシステムへのエクスポートなど、さまざまな目的でアクセスできます。以下のトピックでは、ジャーナルブロックを、台帳から AWS アカウントの Amazon Simple Storage Service (Amazon S3) バケットにエクスポートする方法について説明します。ジャーナルエクスポートジョブは、Amazon Ion 形式のテキストまたはバイナリ表現、または JSON Lines テキスト形式のオブジェクトとして Amazon S3 にデータを書き込みます。

JSON Lines 形式では、エクスポートされたデータオブジェクトの各ブロックは、改行で区切られた有効な JSON オブジェクトです。この形式を使用して、JSON エクスポートを Amazon Athena やなどの分析ツールと直接統合できます。 AWS Glue これらのサービスは改行で区切られた JSON を自動的に解析できるためです。形式の詳細については、JSON Lines を参照してください。

Amazon S3 に関する詳細は、「Amazon Simple Storage Service ユーザーガイド」を参照してください。

> **Note**
>
> エクスポートジョブの出力形式として JSON を指定すると、QLDB はエクスポートされたデータオブジェクトの Ion ジャーナルデータを JSON にダウンコンバートします。詳細については、「JSON へのダウンコンバート」を参照してください。

トピック

- QLDB でのジャーナルエクスポートのリクエスト
- QLDB のジャーナルエクスポート出力
- QLDB のジャーナルエクスポート権限
- ジャーナルエクスポートの一般的なエラー

QLDB でのジャーナルエクスポートのリクエスト

Amazon QLDB は、指定した日時範囲で指定した Amazon S3 バケット送信先へのジャーナルブロックエクスポートをリクエストするための API を提供します。ジャーナルエクスポートジョブは、Amazon Ion 形式のテキストまたはバイナリ表現、または JSON Lines テキスト形式でデータオブジェクトを書き込むことができます。 AWS Management Console、 AWS SDK、または AWS Command Line Interface （ AWS CLI) を使用してエクスポートジョブを作成できます。

トピック

- AWS Management Console
- QLDB API
- エクスポートジョブの有効期限

AWS Management Console

QLDB コンソールを使用して QLDB でジャーナルエクスポートリクエストを送信するには、以下の手順に従います。

エクスポートをリクエストするには (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/qldb で Amazon QLDB コンソールを開きます。
2. ナビゲーションペインの [Export (エクスポート)] を選択します。
3. [Create export job (エクスポートジョブの作成)] を選択します。
4. [Create export job (エクスポートジョブの作成)] ページで、次のエクスポート設定を入力します。

   - [Ledger] (台帳) – エクスポートするジャーナルブロックを含む台帳。
   - [Start date and time] (開始日時) – エクスポートするジャーナルブロックの範囲の開始タイムスタンプ (UTC、協定世界時)。この日時は範囲内に含まれます。このタイムスタンプは、[End date and time (終了日時)] よりも前の日時にしてください。タイムスタンプの開始日時を台帳の `CreationDateTime` 以前に指定した場合、QLDB によって開始日時が台帳の `CreationDateTime` にデフォルト設定されます。
   - [End date and time] (終了日時) – エクスポートするジャーナルブロックの範囲の終了タイムスタンプ (UTC、協定世界時)。この日時は範囲内に含まれません。現在の日時よりも先の日時を指定することはできません。

- [Destination for journal blocks] (ジャーナルブロックの送信先) – エクスポートジョブでデータオブジェクトに書き込む Simple Storage Service (Amazon S3) バケット名とプレフィックス名。次の Simple Storage Service (Amazon S3) URI 形式を使用します。

  ```
  s3://DOC-EXAMPLE-BUCKET/prefix/
  ```

  出力するオブジェクトの S3 バケット名とオプションのプレフィックス名を指定してください。次に例を示します。

  ```
  s3://DOC-EXAMPLE-BUCKET/journalExport/
  ```

  バケット名とプレフィックスは Simple Storage Service (Amazon S3) の命名規則と表記規則に準拠する必要があります。バケット命名規則の詳細については、「Amazon S3 デベロッパーガイド」の「バケットの制約と制限」を参照してください。キー名のプレフィックスの詳細については、「オブジェクトキーとメタデータ」を参照してください。

  > **Note**
  >
  > クロスリージョンエクスポートはサポートされていません。指定された Amazon S3 バケットは台帳 AWS リージョン と同じ にある必要があります。

- [S3 Encryption Configuration] (S3 暗号化設定) – Simple Storage Service (Amazon S3) バケットにデータを書き込むエクスポートジョブで使用される暗号化設定。Simple Storage Service (Amazon S3) のサーバー側の暗号化オプションの詳細については、「Amazon S3 デベロッパーガイド」の「サーバー側の暗号化を使用したデータの保護」を参照してください。
  - [Bucket default encryption] (バケットのデフォルト暗号化) – 指定した Simple Storage Service (Amazon S3) バケットのデフォルトの暗号化設定を使用します。
  - [AES-256] - Simple Storage Service (Amazon S3) マネージドキーによるサーバー側の暗号化 (SSE-S3) を使用します。
  - AWS-KMS – AWS KMS マネージドキーによるサーバー側の暗号化 (SSE-KMS) を使用します。

    このタイプとともに、[異なる AWS KMS keyキーを選択] オプションを選択した場合は、次の Amazon リソースネーム (ARN) 形式で対称暗号化 KMS キーも指定する必要があります。

```
arn:aws:kms:aws-region:account-id:key/key-id
```

- [Service access] (サービスへのアクセス) — Simple Storage Service (Amazon S3) バケットへの書き込みアクセス許可を QLDB に付与する IAM ロール。該当する場合、この IAM ロールによって、KMS キーを使用するためのアクセス許可も QLDB に付与する必要があります。

  ジャーナルエクスポートをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで *iam:PassRole* アクションを実行するためのアクセス許可が必要です。

  - [Create and use a new service role] (新しいサービスロールを作成して使用する) - 指定された Simple Storage Service (Amazon S3) バケットに必要なアクセス許可を持つ新しいロールをコンソールで作成できます。
  - [Use an existing service role] (既存のサービスロールを使用する) - IAM でこのロールを手動で作成する方法については、「エクスポートアクセス許可」を参照してください。
- 出力形式 – エクスポートされたジャーナルデータの出力形式
  - Ion テキスト – (デフォルト) Amazon Ion のテキスト表現
  - Ion バイナリ – Amazon Ion のバイナリ表現
  - JSON – 改行で区切られた JSON テキスト形式

    JSON を選択した場合、QLDB はエクスポートされたデータオブジェクトの Ion ジャーナルデータを JSON にダウンコンバートします。詳細については、「JSON へのダウンコンバート」を参照してください。

5. すべての設定が正しいことを確認したら、[Create export job (エクスポートジョブの作成)] を選択します。

   エクスポートジョブにかかる時間はデータの大きさにより異なります。リクエストの送信が正常に完了すると、コンソールがメインの [Export (エクスポート)] ページに戻り、エクスポートジョブが最新のステータスで一覧表示されます。

6. エクスポートオブジェクトは Simple Storage Service (Amazon S3) コンソールで確認できます。

   https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。

   これらの出力オブジェクトの形式の詳細については、「QLDB のジャーナルエクスポート出力」を参照してください。

> **Note**
>
> エクスポートジョブは、完了後 7 日で期限切れになります。詳細については、「エクスポートジョブの有効期限」を参照してください。

QLDB API

AWS SDK または で Amazon QLDB API を使用してジャーナルエクスポートをリクエストすることもできます AWS CLI。QLDB API は、アプリケーションプログラムで使用する、以下のオペレーションを提供します。

- `ExportJournalToS3` – 日時範囲内のジャーナル内容を、特定の台帳から指定先の Simple Storage Service (Amazon S3) バケットにエクスポートします。エクスポートジョブは、Amazon Ion 形式のテキストまたはバイナリ表現、または JSON Lines テキスト形式のいずれかのオブジェクトとしてデータを書き込むことができます。
- `DescribeJournalS3Export` – ジャーナルエクスポートジョブに関する詳細情報を返します。出力には、現在のステータス、作成時刻、および元のエクスポートリクエストのパラメータが含まれます。
- `ListJournalS3Exports` – 現在の AWS アカウント およびリージョンに関連付けられているすべての台帳について、ジャーナルエクスポートジョブの説明のリストを返します。各エクスポートジョブの説明の出力は、`DescribeJournalS3Export` から返される詳細と同じ内容です。
- `ListJournalS3ExportsForLedger` – 特定の台帳について、ジャーナルエクスポートジョブの説明のリストを返します。各エクスポートジョブの説明の出力は、`DescribeJournalS3Export` から返される詳細と同じ内容です。

これらの API オペレーションの詳細については、Amazon QLDB API リファレンス を参照してください。

を使用したジャーナルデータのエクスポートについては AWS CLI、 コマンドAWS CLI リファレンス を参照してください。

サンプルアプリケーション (Java)

基本的なエクスポートオペレーションの Java コード例については、 GitHub リポジトリ aws-samples/-java amazon-qldb-dmv-sampleを参照してください。このサンプルアプリケーションをダウンロードしてインストールする方法については、「Amazon QLDB Java サンプルアプリケーショ

ンのインストール」を参照してください。エクスポートをリクエストする前に、「Java チュートリアル」のステップ 1～3 を実行し、サンプル台帳を作成して、サンプルデータを使用してロードしてください。

以下のクラスのチュートリアルコードは、エクスポートの作成、エクスポートのステータスの確認、およびエクスポートの出力の処理の例を示しています。

Class	説明
ExportJournal	現時刻から 10 分前までのタイムスタンプの vehicle-registration サンプル台帳からジャーナルブロックをエクスポートします。指定した S3 バケットに出力オブジェクトに書き込むか、バケットがない場合は一意のバケットを作成します。
DescribeJournalExport	vehicle-registration サンプル台帳の指定された exportId のジャーナルエクスポートジョブについて説明します。
ListJournalExports	vehicle-registration サンプル台帳のジャーナルエクスポートジョブの記述のリストを返します。
ValidateQldbHashChain	指定された exportId を使用して vehicle-registration サンプル台帳のハッシュチェーンを検証します。提供されていない場合は、ハッシュチェーンの検証に使用する新しいエクスポートをリクエストします。

エクスポートジョブの有効期限

完了したジャーナルエクスポートジョブには、7 日間の保持期間が適用されます。この制限の有効期限が切れると、自動的にハード削除されます。この有効期限はハード制限であり、変更できません。

完了したエクスポートジョブが削除されると、QLDB コンソールまたは次の API オペレーションを使用してジョブに関するメタデータを取得できなくなります。

- `DescribeJournalS3Export`
- `ListJournalS3Exports`
- `ListJournalS3ExportsForLedger`

ただし、この有効期限は、エクスポートされたデータ自体には影響しません。すべてのメタデータは、エクスポートによって書き込まれたマニフェストファイルに保持されています。この有効期限の目的は、ジャーナルエクスポートジョブを一覧表示する API オペレーションの円滑なエクスペリエンスを提供することです。QLDB では、複数ページのジョブを解析しなくても済むように、古いエクスポートジョブが削除され最近のエクスポートのみが表示されます。

QLDB のジャーナルエクスポート出力

Amazon QLDB ジャーナルエクスポートジョブでは、ジャーナルブロックを含むデータオブジェクトに加え、2 つのマニフェストファイルが書き出されます。これらはすべて、エクスポートリクエストで指定した Amazon S3 バケットに保存されます。以下のセクションでは、各出力オブジェクトの形式と内容について説明します。

> **Note**
>
> エクスポートジョブの出力形式として JSON を指定すると、QLDB はエクスポートされたデータオブジェクトの Amazon Ion ジャーナルデータを JSON にダウンコンバートします。詳細については、「JSON へのダウンコンバート」に進んでください。

トピック

- マニフェストファイル
- データオブジェクト
- JSON へのダウンコンバート
- エクスポートプロセッサライブラリ (Java)

マニフェストファイル

Amazon QLDB では、エクスポートリクエストごとに、指定した S3 バケットに 2 つのマニフェストファイルが作成されます。エクスポートリクエストを送信するとすぐに、最初のマニフェストファイ

ルが作成されます。エクスポートが完了したら、最終的なマニフェストファイルが書き出されます。これらのファイルを使用して、Simple Storage Service (Amazon S3) のエクスポートジョブのステータスを確認できます。

マニフェストファイルの内容の形式は、エクスポートで要求された出力形式に対応しています。

最初のマニフェスト

最初のマニフェストでは、エクスポートジョブが開始したことを示します。このマニフェストには、リクエストに渡した入力パラメータが含まれます。エクスポートの Simple Storage Service (Amazon S3) 送信先、エクスポート用の開始時間パラメータや終了時間パラメータに加え、このファイルには `exportId` も含まれます。`exportId` は、QLDB が各エクスポートジョブに割り当てる、一意の ID です。

このファイルの命名規則は次のとおりです。

```
s3://DOC-EXAMPLE-BUCKET/prefix/exportId.started.manifest
```

以下は、Ion テキスト形式の最初のマニフェストファイルとその内容の例です。

```
s3://DOC-EXAMPLE-BUCKET/journalExport/8UyXulxccYLAsbN1aon7e4.started.manifest
```

```
{
  ledgerName:"my-example-ledger",
  exportId:"8UyXulxccYLAsbN1aon7e4",
  inclusiveStartTime:2019-04-15T00:00:00.000Z,
  exclusiveEndTime:2019-04-15T22:00:00.000Z,
  bucket:"DOC-EXAMPLE-BUCKET",
  prefix:"journalExport",
  objectEncryptionType:"NO_ENCRYPTION",
  outputFormat:"ION_TEXT"
}
```

初期のマニフェストには、エクスポートリクエストで指定された場合のみ、`outputFormat` が含まれます。出力形式を指定しない場合、エクスポートされたデータはデフォルトの `ION_TEXT` 形式 になります。

DescribeJournalS3Export API オペレーションとエクスポートされた Amazon S3 オブジェクトのコンテンツタイプも出力形式を示します。

最終的なマニフェスト

最終的なマニフェストは、特定のジャーナルストランドのエクスポートジョブが完了したことを示します。エクスポートジョブでは、ストランドごとに個別に最終的なマニフェストファイルが書き出されます。

> **Note**
>
> Amazon QLDB で、ストランドは台帳のジャーナルの仕切りです。QLDB は現在、単一ストランドのジャーナルのみをサポートしています。

最終的なマニフェストには、エクスポート中に書き出されたデータオブジェクトキーの順序付きリストが含まれます。このファイルの命名規則は次のとおりです。

```
s3://DOC-EXAMPLE-BUCKET/prefix/exportId.strandId.completed.manifest
```

`strandId` は、QLDB がストランドに割り当てる一意の ID です。以下は、Ion テキスト形式の最終的なマニフェストファイルとその内容の例です。

```
s3://DOC-EXAMPLE-BUCKET/
journalExport/8UyXulxccYLAsbN1aon7e4.JdxjkR9bSYB5jMHWcI464T.completed.manifest
```

```
{
  keys:[
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.1-4.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.5-10.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.11-12.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.13-20.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.21-21.ion"
  ]
}
```

データオブジェクト

Amazon QLDB は、指定された Amazon S3 バケットに Amazon Ion 形式のバイナリ表現または JSON Lines テキスト形式で、ジャーナルデータオブジェクトを書き込みます。

JSON Lines 形式では、エクスポートされたデータオブジェクトの各ブロックは、改行で区切られた有効な JSON オブジェクトです。この形式を使用して、JSON エクスポートを Amazon Athena や

などの分析ツールと直接統合できます。 AWS Glue これらのサービスは改行で区切られた JSON を自動的に解析できるためです。形式の詳細については、JSON Lines を参照してください。

データオブジェクト名

ジャーナルエクスポートジョブでは、以下の命名規則に従ってこれらのデータオブジェクトが書き込まれます。

```
s3://DOC-EXAMPLE-BUCKET/prefix/yyyy/mm/dd/hh/strandId.startSn-endSn.ion|.json
```

- 各エクスポートジョブの出力データはチャンクに分割されます。
- yyyy/mm/dd/hh – エクスポートリクエストを送信したときの日時。同じ時間内にエクスポートされたオブジェクトは、同じ Simple Storage Service (Amazon S3) プレフィックスにグループ分けされます。
- strandId – エクスポートされるジャーナルブロックを含む特定のストランドの一意の ID。
- startSn-endSn – オブジェクトに含まれるシーケンス番号の範囲。シーケンス番号は、ストランド内のブロックの場所を指定します。

たとえば、次のパスを指定するとします。

```
s3://DOC-EXAMPLE-BUCKET/journalExport/
```

エクスポートジョブで、次のような Simple Storage Service (Amazon S3) データオブジェクトが作成されます。この例では、オブジェクト名を Ion 形式で示しています。

```
s3://DOC-EXAMPLE-BUCKET/journalExport/2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.1-5.ion
```

データオブジェクトのコンテンツ

各データオブジェクトには、以下の形式のジャーナルブロックオブジェクトが含まれます。

```
{
  blockAddress: {
    strandId: String,
    sequenceNo: Int
  },
  transactionId: String,
  blockTimestamp: Datetime,
  blockHash: SHA256,
```

```
  entriesHash: SHA256,
  previousBlockHash: SHA256,
  entriesHashList: [ SHA256 ],
  transactionInfo: {
    statements: [
      {
        //PartiQL statement object
      }
    ],
    documents: {
      //document-table-statement mapping object
    }
  },
  revisions: [
    {
      //document revision object
    }
  ]
}
```

ブロックは、トランザクション中にジャーナルにコミットされるオブジェクトです。ブロックには、トランザクションでコミットされたドキュメントリビジョンを表すエントリと、それらを実行した PartiQL ステートメントと共に、トランザクションのメタデータが含まれます。

以下に、Ion テキスト形式のサンプルデータを含むブロックの例を示します。ブロックオブジェクトのフィールドの詳細については、「Amazon QLDB のジャーナルコンテンツ」を参照してください。

> **Note**
>
> このブロック例は、情報提供のみを目的として記載されています。示されているハッシュは、実際に計算されたハッシュ値ではありません。

```
{
  blockAddress:{
    strandId:"JdxjkR9bSYB5jMHWcI464T",
    sequenceNo:1234
  },
  transactionId:"D35qctdJRU1L1N2VhxbwSn",
  blockTimestamp:2019-10-25T17:20:21.009Z,
  blockHash:{{WYLOfZClk0lYWT3lUsSr0ONXh+Pw8MxxB+9zvTgSvlQ=}},
  entriesHash:{{xN9X96atkMvhvF3nEy6jMSVQzKjHJfz1H3bsNeg8GMA=}},
```

```
  previousBlockHash:{{IAfZ0h22ZjvcuHPSBCDy/6XNQTsqEmeY3GW0gBae8mg=}},
  entriesHashList:[
      {{F7rQIKCNn0vXVWPexilGfJn5+MCrtsSQqqVdlQxXpS4=}},
      {{C+L8gRhkzVcxt3qRJpw8w6hVEqA5A6ImGne+E7iHizo=}}
  ],
  transactionInfo:{
    statements:[
      {
        statement:"CREATE TABLE VehicleRegistration",
        startTime:2019-10-25T17:20:20.496Z,
        statementDigest:{{3jeSdejOgp6spJ8huZxDRUtp2fRXRqpOMtG43V0nXg8=}}
      },
      {
        statement:"CREATE INDEX ON VehicleRegistration (VIN)",
        startTime:2019-10-25T17:20:20.549Z,
        statementDigest:{{099D+5ZWDgA7r+aWeNUrWhc8ebBTXjgscq+mZ2dVibI=}}
      },
      {
        statement:"CREATE INDEX ON VehicleRegistration (LicensePlateNumber)",
        startTime:2019-10-25T17:20:20.560Z,
        statementDigest:{{B73tVJzVyVXicnH4n96NzU2L2JFY8e9Tjg895suWMew=}}
      },
      {
        statement:"INSERT INTO VehicleRegistration ?",
        startTime:2019-10-25T17:20:20.595Z,
        statementDigest:{{ggpon5qCXLo95K578YVhAD8ix0A0M5CcBx/W40Ey/Tk=}}
      }
    ],
    documents:{
      '8F0TPCmdNQ6JTRpiLj2TmW':{
        tableName:"VehicleRegistration",
        tableId:"BPxNiDQXCIB5l5F68KZoOz",
        statements:[3]
      }
    }
  },
  revisions:[
    {
      hash:{{FR1IWcWew0yw1TnRklo2YMF/qtwb7ohsu5FD8A4DSVg=}}
    },
    {
      blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:1234
```

```
      },
      hash:{{t8Hj6/VC4SBitxnvBqJbOmrGytF2XAA/1c0AoSq2NQY=}},
      data:{
        VIN:"1N4AL11D75C109151",
        LicensePlateNumber:"LEWISR261LL",
        State:"WA",
        City:"Seattle",
        PendingPenaltyTicketAmount:90.25,
        ValidFromDate:2017-08-21,
        ValidToDate:2020-05-11,
        Owners:{
          PrimaryOwner:{
            PersonId:"GddsXfIYfDlKCEprOLOwYt"
          },
          SecondaryOwners:[]
        }
      },
      metadata:{
        id:"8F0TPCmdNQ6JTRpiLj2TmW",
        version:0,
        txTime:2019-10-25T17:20:20.618Z,
        txId:"D35qctdJRU1L1N2VhxbwSn"
      }
    }
  ]
}
```

revisions フィールドには、hash 値のみを含み、その他の属性を含まないリビジョンオブジェクトもあります。これらは、ユーザーデータを含まない内部のみのシステムリビジョンです。これらのリビジョンのハッシュはジャーナルのハッシュチェーン全体の一部であるため、エクスポートジョブでは、これらのリビジョンが対応するブロックに含まれます。暗号検証には、完全なハッシュチェーンが必要です。

JSON へのダウンコンバート

エクスポートジョブの出力形式として JSON を指定すると、QLDB はエクスポートされたデータオブジェクトの Amazon Ion ジャーナルデータを JSON にダウンコンバートします。ただし、JSON にないリッチ Ion 型をデータで使用している特定のケースでは、Ion を JSON に変換するとデータが劣化する可能性があります。

Ion から JSON への変換ルールの詳細については、「Amazon Ion Cookbook」(Amazon Ion クックブック) の「Down-converting to JSON」(JSON へのダウンコンバート) を参照してください。

エクスポートプロセッサライブラリ (Java)

QLDB は、Amazon S3 でのエクスポート処理を効率化する Java 用の拡張可能なフレームワークを提供します。このフレームワークライブラリは、エクスポートの出力を読み取り、エクスポートされたブロックを順番に反復処理します。このエクスポートプロセッサを使用するには、 GitHub リポジトリ awslabs/amazon-qldb-export-processor-java を参照してください。

QLDB のジャーナルエクスポート権限

Amazon QLDB でジャーナルエクスポートリクエストを送信する前に、指定した Amazon S3 バケットへの書き込みアクセス許可を QLDB に付与する必要があります。Amazon S3 バケットのカスタマーマネージド AWS KMS key をエクスポートジョブのオブジェクト暗号化タイプに選択する場合は、指定した対称暗号化キーを使用するためのアクセス許可を QLDB に付与する必要もあります。Simple Storage Service (Amazon S3) は非対称 KMS キーをサポートしていません。

エクスポートジョブに必要なアクセス許可を提供するために、QLDB が適切なアクセス許可ポリシーを持つ IAM サービスロールを引き受けるようにすることができます。サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに権限を委任するロールの作成」を参照してください。

> **Note**
>
> ジャーナルエクスポートをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで `iam:PassRole` アクションを実行するためのアクセス許可が必要です。これは、QLDB 台帳リソースへの `qldb:ExportJournalToS3` アクセス許可に追加されるものです。
>
> IAM を使用して QLDB へのアクセスを制御する方法については、「Amazon QLDB で IAM が機能する仕組み」を参照してください。QLDB ポリシーの例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

この例では、お客様に代わって Simple Storage Service (Amazon S3) バケットにオブジェクトを書き込むことを QLDB に許可するロールを作成します。詳細については、「IAM ユーザーガイド」の「AWS のサービスに権限を委任するロールの作成」を参照してください。

で初めて QLDB ジャーナルをエクスポートする場合は AWS アカウント 、まず以下の操作を行って、適切なポリシーで IAM ロールを作成する必要があります。または、QLDB コンソールを使用して、ロールを自動的に作成できます。それ以外の場合は、前に作成したロールを選択できます。

トピック

- 許可ポリシーを作成する
- IAM ロールを作成する

許可ポリシーを作成する

以下のステップを完了して、QLDB ジャーナルエクスポートジョブに対するアクセス許可ポリシーを作成します。この例で示しているのは、指定したバケットにオブジェクトを書き込むためのアクセス許可を QLDB に付与する Simple Storage Service (Amazon S3) バケットポリシーです。該当する場合、QLDB に対称暗号化 KMS キーの使用を許可するキーポリシーも示しています。

Simple Storage Service (Amazon S3) バケットポリシーの詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットポリシーとユーザーポリシーの使用」を参照してください。 AWS KMS キーポリシーの詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMSでキーポリシーを使用する」を参照してください。

> **Note**
>
> Amazon S3 バケットと KMS キーはどちらも QLDB 台帳 AWS リージョン と同じ にある必要があります。

JSON ポリシーエディタでポリシーを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーション列で、[ポリシー] を選択します。

   [Policies] (ポリシー) を初めて選択する場合は、[Welcome to Managed Policies] (マネージドポリシーにようこそ) ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [JSON] タブを選択します。
5. JSON ポリシードキュメントを入力します。

- Simple Storage Service (Amazon S3) オブジェクトの暗号化にカスタマーマネージド KMS キーを使用する場合は、以下のポリシードキュメントの例を使用します。このポリシーを使用するには、この例の *DOC-EXAMPLE-BUCKET*、*us-east-1*、*123456789012*、*1234abcd-12ab-34cd-56ef-1234567890ab* を自分の情報と置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportS3Permission",
            "Action": [
                "s3:PutObjectAcl",
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        },
        {
            "Sid": "QLDBJournalExportKMSPermission",
            "Action": [ "kms:GenerateDataKey" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

- 他の暗号化タイプの場合は、以下のポリシードキュメントの例を使用します。このポリシーを使用するには、以下の例の *DOC-EXAMPLE-BUCKET* を自分の Simple Storage Service (Amazon S3) バケット名と置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportS3Permission",
            "Action": [
                "s3:PutObjectAcl",
                "s3:PutObject"
            ],
            "Effect": "Allow",
```

```
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        }
    ]
}
```

6. [ポリシーの確認] を選択します。

> **Note**
>
> いつでも [Visual editor] (ビジュアルエディタ) タブと [JSON] タブを切り替えることができます。ただし、[Visual editor] (ビジュアルエディタ) タブで [Review policy] (ポリシーの確認) を変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイドの「ポリシーの再構成」を参照してください。

7. [ポリシーの確認] ページに作成するポリシーの [名前] と [説明] を入力します。ポリシーの [Summary] (概要) を参照して、ポリシーによって付与された許可を確認します。次に、[Create policy] (ポリシーの作成) を選択して作業を保存します。

IAM ロールを作成する

QLDB ジャーナルエクスポートジョブのアクセス許可ポリシーを作成したら、次に IAM ロールを作成し、それにポリシーをアタッチできます。

QLDB のサービスロールを作成するには (IAM コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
3. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
4. [サービス] または [ユースケース] で [QLDB] を選択し、次に [QLDB] ユースケースを選択します。
5. [次へ] をクリックします。
6. 前のステップで作成したポリシーの横にあるボックスをオンにします。
7. (オプション) アクセス許可の境界を設定します。このアドバンスド機能は、サービスロールで使用できますが、サービスにリンクされたロールではありません。

a. [アクセス許可の境界の設定] セクションを開き、[アクセス許可の境界を使用してロールのアクセス許可の上限を設定する] を選択します。

   IAM には、アカウント内の AWS 管理ポリシーとカスタマー管理ポリシーのリストが含まれます。

b. アクセス許可の境界として使用するポリシーを選択します。

8. [次へ] をクリックします。
9. このロールの目的を識別しやすいロール名またはロール名サフィックスを入力します。

> **Important**
>
> ロールに名前を付けるときは、次のことに注意してください。
>
> - ロール名は 内で一意である必要があり AWS アカウント、大文字と小文字を区別することはできません。
>
>   例えば、**PRODROLE** と **prodrole** の両方の名前でロールを作成することはできません。ロール名がポリシーまたは ARN の一部として使用される場合、ロール名は大文字と小文字が区別されます。ただし、サインインプロセスなど、コンソールにロール名がユーザーに表示される場合、ロール名は大文字と小文字が区別されません。
> - 他のエンティティがロールを参照する可能性があるため、ロールを作成した後にロール名を編集することはできません。

10. (オプション) [説明] にロールの説明を入力します。
11. (オプション) ロールのユースケースとアクセス許可を編集するには、[ステップ 1: 信頼されたエンティティを選択] または [ステップ 2: アクセス権限を追加] のセクションで [編集] を選択します。
12. (オプション) ロールの識別、整理、検索を簡単にするには、キーと値のペアとしてタグを追加します。IAM でのタグの使用に関する詳細については、『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。
13. ロールを確認したら、[Create role] (ロールを作成) を選択します。

以下の JSON ドキュメントは、特定の許可がアタッチされた IAM ロールを引き受けることを QLDB に許可する信頼ポリシーの例です。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

> **Note**
>
> この信頼ポリシーの例では、aws:SourceArn および aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題を回避する方法を示します。この信頼ポリシーを使用すると、QLDB は、アカウント 123456789012 のみの QLDB リソースのロールを引き受けることができます。
> 詳細については、「クロスサービスの混乱した副防止」を参照してください。

IAM ロールを作成した後、QLDB コンソールに戻り、[Create export job] (エクスポートジョブの作成) ページを更新すると、新しいロールが見つかります。

ジャーナルエクスポートの一般的なエラー

このセクションでは、ジャーナルエクスポートのリクエストに対して Amazon QLDB がスローするランタイムエラーについて説明します。

以下は、サービスによって返される一般的な例外のリストです。それぞれの例外には、特定のエラーメッセージに加え、簡単な説明と考えられる解決方法に関する推奨事項が記載されています。

AccessDeniedException

メッセージ: ユーザー: *userARN* は、iam:PassRole on resource: *roleARN* を実行する権限がありません

IAM ロールを QLDB サービスに渡す許可がありません。QLDB には、すべてのジャーナルエクスポートリクエストに対応するロールが必要であり、このロールを QLDB に渡す許可が必要です。このロールにより、指定した Amazon S3 バケットへの書き込みアクセス許可が QLDB に付与されます。

指定した IAM ロールリソースに対して `PassRole` API オペレーションを実行する許可を付与する IAM ポリシーを QLDB サービス (`qldb.amazonaws.com`) に定義していることを確認します。ポリシーの例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

IllegalArgumentException

メッセージ: QLDB encountered an error validating S3 configuration (QLDB で S3 設定の検証中にエラーが発生しました): *errorCode errorMessage*

このエラーの原因として考えられるのは、指定したバケットが Simple Storage Service (Amazon S3) に存在しないことです。または、指定した Simple Storage Service (Amazon S3) バケットにオブジェクトを書き込むための権限が QLDB にないと考えられます。

エクスポートジョブリクエストで指定した S3 バケット名が正しいか確認します。バケットの命名規則の詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの制約と制限」を参照してください。

また、指定したバケットに対する `PutObject` および `PutObjectAcl` アクセス許可を QLDB サービス (`qldb.amazonaws.com`) に付与するポリシーを定義していることを確認します。詳細については、「エクスポートアクセス許可」を参照してください。

IllegalArgumentException

メッセージ: Unexpected response from Amazon S3 while validating the S3 configuration. (S3 設定の検証中に、Amazon S3 から不測のレスポンスがありました。) S3 からのレスポンス: *errorCode errorMessage*

指定した S3 バケットにジャーナルエクスポートデータを書き込もうとしましたが、Simple Storage Service (Amazon S3) エラーレスポンスがあり失敗しました。考えられる原因の詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 のトラブルシューティング」を参照してください。

IllegalArgumentException

メッセージ: Amazon S3 bucket prefix must not exceed 128 characters (Amazon S3 バケットプレフィックスは 128 文字以下でなければなりません)

ジャーナルエクスポートのリクエストで指定したプレフィックスが 128 文字を超えています。

IllegalArgumentException

メッセージ: Start date must not be greater than end date (開始日は終了日よりも前の日付でなければなりません)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、ISO 8601 の日時形式、協定世界時 (UTC) にしてください。

IllegalArgumentException

メッセージ: End date cannot be in future (終了日を現在よりも後の日付にすることはできません)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、`ISO 8601` の日時形式、UTC にしてください。

IllegalArgumentException

メッセージ: 指定されたオブジェクト暗号化設定 (S3EncryptionConfiguration) は AWS Key Management Service （ AWS KMS) キーと互換性がありません

`ObjectEncryptionType` が `NO_ENCRYPTION` または `SSE_S3` の `KMSKeyArn` が指定されています。`SSE_KMS` のオブジェクトの暗号化タイプに対してのみ、カスタマーマネージド AWS KMS key を指定できます。Simple Storage Service (Amazon S3) のサーバー側の暗号化オプションの詳細については、「Amazon S3 デベロッパーガイド」の「サーバー側の暗号化を使用したデータの保護」を参照してください。

LimitExceededException

メッセージ: Exceeded the limit of 2 concurrently running Journal export jobs (ジャーナルエクスポートジョブの同時実行の上限数 2 を超えています)

QLDB では、デフォルトで、ジャーナルエクスポートジョブの同時実行は最大 2 件までという制限が適用されます。

Amazon QLDB からのジャーナルデータのストリーミング

Amazon QLDB では、イミュータブルトランザクションログ (ジャーナル) をデータストレージに使用します。ジャーナルは、コミット済みデータへの変更をすべて追跡し、完全かつ検証可能な変更履歴を一定の期間にわたって維持します。

ジャーナルへコミットされたドキュメントリビジョンをすべてキャプチャするストリーミングを QLDB で作成し、このデータを Amazon Kinesis Data Streams にほぼリアルタイムで配信できます。QLDB ストリーミングは、台帳のジャーナルから Kinesis データストリームリソースへの連続的なデータのフローです。

その後、Kinesis ストリーミングプラットフォームまたは Kinesis Client Library を使用して、ストリーミングの消費、データレコードの処理、およびデータコンテンツの分析を行います。QLDB ストリーミングは、コントロール、ブロックサマリー、およびリビジョンの詳細という 3 種類のレコードで Kinesis Data Streams にデータを書き込みます。詳細については、「Kinesis での QLDB ストリーミングレコード」を参照してください。

トピック

一般的なユースケース

ストリーミングを使用すると、ジャーナルデータを他のサービスと統合しながら、検証可能な信頼できる唯一の情報源として QLDB を使用できます。QLDB ジャーナルストリーミングでサポートされている一般的ユースケースの一部を以下に示します。

- イベント駆動型アーキテクチャ - デカップリングされたコンポーネントを使用して、イベント駆動型アーキテクチャスタイルでアプリケーションを構築します。例えば、銀行は AWS Lambda 関数を使用して、アカウント残高がしきい値を下回ったときに顧客に警告する通知システムを実装できます。このようなシステムでは、口座残高は QLDB 台帳に維持され、残高の変更はジャーナルに記録されます。 AWS Lambda 関数は、ジャーナルにコミットされ、Kinesis データストリームに送信されるバランス更新イベントを消費すると、通知ロジックをトリガーできます。
- リアルタイム分析 - イベントデータに対してリアルタイム分析を実行する Kinesis コンシューマーアプリケーションを構築します。この機能により、ほぼリアルタイムでインサイトを取得し、変化するビジネス環境に迅速に対応できます。たとえば、e コマースウェブサイトでは、商品の販売データを分析し、販売が制限に達するとすぐに割引商品の広告を停止できます。
- 履歴分析 - 履歴イベントデータを再生することで、Amazon QLDB のジャーナル指向アーキテクチャを活用します。QLDB ストリーミングを過去の任意の時点から開始するように選択し、その時点以降のすべてのリビジョンが Kinesis Data Streams に配信されるようにできます。この機能を使用すると、履歴データの分析ジョブを実行する Kinesis コンシューマーアプリケーションを構築できます。例えば、e コマースウェブサイトでは、必要に応じてアドホック分析を実行し、これまで取得されていなかった過去の販売メトリクスを生成できます。
- 目的別データベースへのレプリケーション - QLDB ジャーナルストリーミングを使用して、QLDB 台帳を他の目的別データストアに接続します。例えば、Kinesis ストリーミングデータプラットフォームを使用して Amazon OpenSearch Service と統合し、QLDB ドキュメントのフルテキスト検索機能を提供できます。また、カスタム Kinesis コンシューマーアプリケーションを構築して、異なるマテリアライズドビューを提供する他の目的別データベースにジャーナルデータをレプリケートすることもできます。たとえば、リレーショナルデータの場合は Amazon Aurora、グラフベースのデータの場合は Amazon Neptune にレプリケートします。

ストリームの消費

データレコードの大量のストリーミングの連続的な消費、処理、分析には、Kinesis Data Streams を使用します。Kinesis Data Streams に加えて、Kinesis ストリーミングデータプラットフォームには Amazon Data Firehose と Amazon Managed Service for Apache Flink が含まれています。このプラットフォームを使用して、Amazon OpenSearch Service、Amazon Redshift、Amazon S3、Splunk などの サービスにデータレコードを直接送信できます。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Amazon Kinesis Data Streams コンシューマー」を参照してください。

Kinesis Client Library (KCL) を使用して、カスタムな方法でデータレコードを処理するストリーミングコンシューマーアプリケーションを構築することもできます。KCL は、低レベルの Kinesis Data

Streams API の上で役に立つ抽象化を提供することによりコーディングを簡素化します。KCL の詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Using the Kinesis Client Library」を参照してください。

配信の保証

QLDB ストリームはat-least-once配信保証を提供します。QLDB ストリーミングで生成された各データレコードは、Kinesis Data Streams に少なくとも 1 回配信されます。同じレコードが Kinesis データストリームに複数回表示されることがあります。したがって、ユースケースで必要な場合は、コンシューマーアプリケーションレイヤーに重複除外ロジックが必要です。

また、注文の保証はありません。状況によっては、QLDB ブロックとリビジョンが Kinesis データストリーム内で誤った順序で生成されることがあります。詳細については、「重複する レコードと out-of-order レコードの処理」を参照してください。

配信のレイテンシーに関する注意事項

QLDB ストリームは通常、ほぼリアルタイムで Kinesis データデータストリーム に更新を配信します。ただし、次のシナリオでは、新しくコミットされた QLDB データが Kinesis データストリームに出力されるまでに、さらにレイテンシーが発生する可能性があります。

- Kinesis は、Kinesis データストリームのプロビジョニングに応じて、QLDB からストリーミングされるデータをスロットルすることができます。例えば、単一の Kinesis データストリームに書き込む QLDB ストリームが複数あり、QLDB のリクエストレートが Kinesis ストリームリソースの容量を超えた場合にこれが発生する可能性があります。Kinesis のスロットリングは、オンデマンドプロビジョニングを使用している場合、スループットが 15 分未満で前回のピークの 2 倍以上に増加した場合にも発生する可能性があります。

  この超過スループットは、Kinesis メトリクス `WriteProvisionedThroughputExceeded` を監視することで測定できます。詳細と可能な解決策については、「Kinesis データストリームのスロットリングエラーのトラブルシューティング方法」を参照してください。
- QLDB ストリームでは、開始日時が過去のもので、終了日時がない無期限のストリームを作成できます。設計上、QLDB は、指定された開始日時に先行データがすべて正常に配信された後にのみ、新しくコミットされたデータを Kinesis データストリームに送信し始めます。このシナリオでさらにレイテンシーが発生する場合は、先行データが配信されるのを待つか、ストリームをより遅い開始日時から開始する必要があるかもしれません。

ストリームの使用開始

Kinesis Data Streams へのジャーナルデータのストリーミングを開始するために必要な手順の概要を次に示します。

1. Kinesis Data Streams リソースを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「データストリームの作成および更新」を参照してください。
2. QLDB が Kinesis データストリームの書き込み許可を引き受けることができる IAM ロールを作成します。手順については、「QLDB のストリーミング許可」を参照してください。
3. QLDB ジャーナルストリーミングを作成します。手順については、「QLDB でのストリーミングの作成と管理」を参照してください。
4. 前のセクション「ストリームの消費」で説明したように、Kinesis データストリームを消費します。Kinesis Client Library または の使用方法を示すコード例については AWS Lambda、「」を参照してくださいQLDB でのストリーミングを使用した開発。

QLDB でのストリーミングの作成と管理

Amazon QLDB では、台帳から Amazon Kinesis Data Streams へのジャーナルデータのストリーミングを作成し管理するための API オペレーションを提供します。QLDB ストリーミングは、ジャーナルにコミットされたドキュメントリビジョンをすべてキャプチャし、Kinesis データストリームに送信します。

AWS Management Console、 AWS SDK、または AWS Command Line Interface（ AWS CLI) を使用してジャーナルストリームを作成できます。また、AWS CloudFormation テンプレートを使用してストリーミングを作成することもできます。詳細については、「 ユーザーガイド」の「 AWS::QLDB::StreamリソースAWS CloudFormation 」を参照してください。

トピック

- ストリームパラメータ
- ストリーム ARN
- AWS Management Console
- ストリームの状態
- 障害のあるストリームの処理

ストリームパラメータ

QLDB ジャーナルストリーミングを作成するには、以下の設定パラメータを指定する必要があります。

台帳名

Kinesis Data Streams へストリーミングするジャーナルデータがある QLDB 台帳。

ストリーム名

QLDB ジャーナルストリームに割り当てる名前。ユーザー定義の名前は、ストリームの目的を識別して示すのに役立ちます。

ストリーム名は、特定の台帳の他のアクティブなストリーム間で一意である必要があります。ストリーム名には、「Amazon QLDB でのクォータと制限」で定義されている台帳名と同じ命名制約があります。

QLDB では、ストリーミング名に加えて、作成する各 QLDB ストリーミングにストリーミング ID を送信します。ストリーム ID は、そのステータスに関係なく、特定の台帳のすべてのストリーム間で一意です。

開始日時

ジャーナルデータのストリーミングを開始する日時。この値には、過去の任意の日付と時刻を指定できますが、将来の日付を指定することはできません。

終了日時

(任意) ストリームの終了を指定する日時。

終了時刻のない無期限のストリームを作成する場合、ストリームを終了するには手動でキャンセルする必要があります。また、指定された終了日時に達していないアクティブな有限ストリームをキャンセルすることもできます。

ストリーミング先の Kinesis データストリーム

ストリーミングがデータレコードを書き込む Kinesis Data Streams ターゲットリソース。Kinesis データストリームの作成方法については、「Amazon Kinesis Data Streams デベロッパーガイド」の「データストリームの作成および更新」を参照してください。

> **Important**
>
> - クロスリージョンおよびクロスアカウントストリームはサポートされていません。指定する Kinesis データストリームは、台帳と同じ AWS リージョン およびアカウントに存在する必要があります。
> - Kinesis Data Streams のレコード集約は、デフォルトで有効になっています。QLDB が 1 つの Kinesis Data Streams レコードで複数のデータレコードを発行できるようにします。これにより、API コールごとに送信されるレコードの数が増加します。
>
> レコード集約は、レコードの処理に重要な意味を持ち、ストリーミングコンシューマーで集約を解除する必要があります。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「KPL の主要な概念」および「コンシューマーの集約解除」を参照してください。

IAM ロール

QLDB が Kinesis データストリームへの書き込み許可を引き受けられるようにする IAM ロール。QLDB コンソールを使用してこのロールを自動的に作成するか、IAM で手動で作成できます。手動で作成する方法については、「ストリームアクセス許可」を参照してください。

ジャーナルストリーミングをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで `iam:PassRole` アクションを実行するための許可が必要です。

ストリーム ARN

すべての QLDB ジャーナルストリーミングは台帳のサブリソースであり、Amazon リソースネーム (ARN) によって一意に識別されます。以下は、`exampleLedger` という名前の台帳の、ストリーミング ID が `IiPT4brpZCqCq3f4MTHbYy` である QLDB ストリーミングの ARN の例です。

```
arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/IiPT4brpZCqCq3f4MTHbYy
```

次のセクションでは、 AWS Management Consoleを使用して QLDB ストリーミングを作成およびキャンセルする方法について説明します。

AWS Management Console

QLDB コンソールを使用して QLDB ストリーミングを作成またはキャンセルするには、次の手順に従います。

ストリームを作成するには (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/qldb で Amazon QLDB コンソールを開きます。
2. ナビゲーションペインで、[ストリーム] を選択します。
3. [Create QLDB stream] (QLDB ストリームを作成) を選択します。
4. [Create QLDB stream] (QLDB ストリームを作成) ページで、以下の設定を入力します。

   - [Stream name] (ストリーム名) - QLDB ストリーミングに割り当てる名前。
   - [Ledger] (台帳) - ストリーミングするジャーナルデータがある台帳。
   - [Start date and time] (開始日時) - ジャーナルデータのストリーミングを開始する、協定世界時 (UTC、Coordinated Universal Time) でのタイムスタンプ (ストリーミング期間に含まれます)。このタイムスタンプのデフォルトは、現在の日時です。これを将来の日時にすることはできません。また、[終了日時] より前の日付にする必要があります。
   - [End date and time] (終了日時) - (オプション) ストリーミングの終了時期を指定する UTC 形式のタイムスタンプ (ストリーミング期間に含まれません)。このパラメータを空白のままにすると、ストリームはキャンセルするまで無期限に実行されます。
   - [Destination stream] (送信先ストリーム) - ストリーミングがデータレコードを書き込む Kinesis Data Streams ターゲットリソース。次の ARN 形式を使用します。

     ```
     arn:aws:kinesis:aws-region:account-id:stream/kinesis-stream-name
     ```

     次に例を示します。

     ```
     arn:aws:kinesis:us-east-1:123456789012:stream/stream-for-qldb
     ```

     クロスリージョンおよびクロスアカウントストリームはサポートされていません。指定された Kinesis データストリームは、台帳と同じ AWS リージョン とアカウントに存在する必要があります。
   - [Enable record aggregation in Kinesis Data Streams] (Kinesis Data Streams でレコード集約を有効にする) — (デフォルトで有効) QLDB が 1 つの Kinesis Data Streams レコードで複数の

データレコードを発行できるようにします。これにより、API コールごとに送信されるレコードの数が増加します。

- [Service access] (サービスアクセス) - Kinesis データストリームへの書き込み許可を QLDB に付与する IAM ロール。

  ジャーナルストリーミングをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで *iam:PassRole* アクションを実行するための許可が必要です。

  - [Create and use a new service role] (新しいサービスロールを作成して使用する) - 指定された Kinesis データストリームに必要な許可を持つ新しいロールをコンソールで作成できます。
  - [Use an existing service role] (既存のサービスロールを使用する) - IAM でこのロールを手動で作成する方法については、「ストリームアクセス許可」を参照してください。

- [Tags] (タグ) - (オプション) タグをキーと値の組み合わせとしてアタッチすることで、メタデータをストリーミングに追加します。ストリームにタグを追加すると、台帳の整理と識別がしやすくなります。詳細については、「Amazon QLDB リソースのタグ付け」を参照してください。

  [Add tag (タグの追加)] を選択し、必要に応じて、任意のキーと値の組み合わせを入力します。

5. 設定が正しいことを確認したら、[Create QLDB stream] (QLDB ストリームを作成) を選択します。

   リクエストの送信が正常に完了すると、コンソールがメインの [Streams] (ストリーム) ページに戻り、QLDB ストリーミングが最新のステータスで一覧表示されます。

6. ストリーミングがアクティブになったら、Kinesis を使用して、コンシューマーアプリケーションでストリーミングデータを処理します。

   Kinesis Data Streams コンソール (https://console.aws.amazon.com/kinesis/) を開きます。

ストリームデータレコード形式の詳細については、「Kinesis での QLDB ストリーミングレコード」を参照してください。

エラーの原因となるストリームの処理方法については、「障害のあるストリームの処理」を参照してください。

ストリームをキャンセルするには (コンソール)

キャンセルすると、QLDB ストリーミングは再開できません。Kinesis Data Streams へのデータの配信を再開するには、新しい QLDB ストリーミングを作成します。

1. Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペインで、[ストリーム] を選択します。
3. QLDB ストリーミングのリストで、キャンセルするアクティブなストリーミングを選択します。
4. [Cancel stream (ストリームをキャンセル)] を選択します。表示されたボックスに「**`cancel stream`**」と入力して、削除を確定します。

AWS SDK または で QLDB API を使用してジャーナルストリームを作成および管理 AWS CLI する方法の詳細については、「」を参照してくださいQLDB でのストリーミングを使用した開発。

ストリームの状態

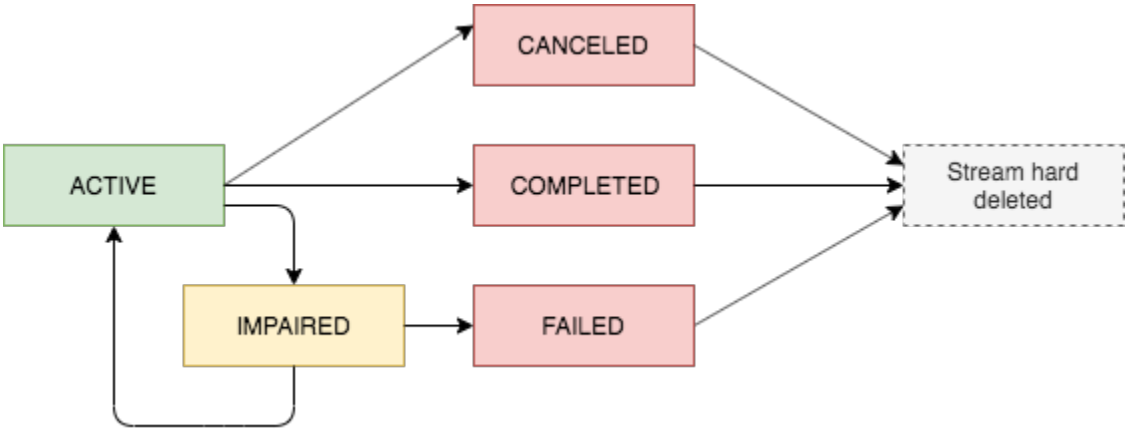
QLDB ストリーミングのステータスは、以下のいずれかになります。

- `ACTIVE` - 現在、データのストリーミング中またはストリーミングを待機 (終了時刻が指定されていない無期限ストリーミングの場合) しています。
- `COMPLETED` - 指定された時間範囲内のすべてのジャーナルブロックのストリーミングが正常に完了しました。これは終了状態です。

- `CANCELED` - 指定された終了時刻より前にユーザーのリクエストによって終了し、データのストリーミングはアクティブではありません。これは終了状態です。
- `IMPAIRED` - アクションが必要なエラーが発生したため、レコードを Kinesis に書き込めません。これは、回復可能な非終端状態です。

  1 時間以内にエラーを解決すると、ストリームは自動的に `ACTIVE` 状態に移行します。1 時間経過してもエラーが未解決のままである場合、ストリームは自動的に `FAILED` 状態に移行します。
- `FAILED` - エラーのため、レコードを Kinesis に書き込めず、回復不能な終了状態です。

次の図は、QLDB ストリーミングリソースの状態がどのように遷移するかを示しています。



端末ストリームの有効期限

端末状態 (CANCELED、COMPLETED、および FAILED) のストリームリソースには、7 日間の保持期間が適用されます。この制限の有効期限が切れると、自動的にハード削除されます。

終了状態のストリーミングが削除されると、QLDB コンソールまたは QLDB API を使用してストリーミングリソースの詳細取得または一覧表示ができなくなります。

障害のあるストリームの処理

ストリーミングでエラーが発生した場合は、IMPAIRED 状態にまず移行します。QLDB は引き続き IMPAIRED ストリーミングを最大 1 時間再試行します。

1 時間以内にエラーを解決すると、ストリームは自動的に ACTIVE 状態に移行します。1 時間経過してもエラーが未解決のままである場合、ストリームは自動的に FAILED 状態に移行します。

障害のあるストリームまたは失敗したストリームには、次のいずれかのエラー原因があります。

- KINESIS_STREAM_NOT_FOUND - ストリーミング先の Kinesis Data Streams リソースは存在しません。QLDB ストリーミングリクエストで指定した Kinesis データストリームが正しいことを確認します。次に、Kinesis に移動し、指定したデータストリームを作成します。
- IAM_PERMISSION_REVOKED - QLDB に、指定した Kinesis データストリームにデータレコードを書き込むための十分な許可がありません。以下のアクションに対して QLDB サービス (qldb.amazonaws.com) 許可を付与する、指定した Kinesis データストリーム用のポリシーを定義していることを確認します。
  - kinesis:PutRecord
  - kinesis:PutRecords

- `kinesis:DescribeStream`
- `kinesis:ListShards`

障害のあるストリームのモニタリング

ストリーミングに障害が発生すると、ストリーミングに関する詳細と発生したエラーを示すバナーが QLDB コンソールに表示されます。`DescribeJournalKinesisStream` API オペレーションを使用して、ストリーミングのステータスと基になっているエラー原因を取得することもできます。

さらに、Amazon を使用して、ストリームの`IsImpaired`メトリクスをモニタリングするアラーム CloudWatch を作成できます。による QLDB メトリクスのモニタリングについては CloudWatch、「」を参照してくださいAmazon QLDB のディメンションとメトリクス。

QLDB でのストリーミングを使用した開発

このセクションでは、 AWS SDK または で Amazon QLDB でジャーナルストリームを作成および管理 AWS CLI するために使用できる API オペレーションの概要を説明します。また、これらのオペレーションのデモを行い、Kinesis Client Library (KCL) または AWS Lambda を使用してストリーミングコンシューマーを実装するサンプルアプリケーションも示します。

KCL を使用して Amazon Kinesis Data Streams のコンシューマーアプリケーションを構築できます。KCL は、低レベルの Kinesis Data Streams API の上で役に立つ抽象化を提供することによりコーディングを簡素化します。KCL の詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Using the Kinesis Client Library」を参照してください。

目次

QLDB ジャーナルストリーミング API

QLDB API は、アプリケーションプログラムで使用する、以下のジャーナルストリーミングオペレーションを提供します。

- `StreamJournalToKinesis` - 指定した QLDB 台帳のジャーナルストリーミングを作成します。このストリーミングは、台帳のジャーナルにコミットされたドキュメントリビジョンをすべてキャプチャし、指定した Kinesis Data Streams リソースにそのデータを配信します。
  - Kinesis Data Streams のレコード集約は、デフォルトで有効になっています。QLDB が 1 つの Kinesis Data Streams レコードで複数のデータレコードを発行できるようにします。これにより、API コールごとに送信されるレコードの数が増加します。

    レコード集約は、レコードの処理に重要な意味を持ち、ストリーミングコンシューマーで集約を解除する必要があります。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「KPL の主要な概念」および「コンシューマーの集約解除」を参照してください。
- `DescribeJournalKinesisStream` - 指定された QLDB ジャーナルストリーミングに関する詳細情報を返します。出力には、ARN、ストリーム名、現在のステータス、作成時刻、および元のストリーム作成リクエストのパラメータが含まれます。
- `ListJournalKinesisStreamsForLedger` - 指定された台帳のすべての QLDB ジャーナルストリーミング記述子のリストを返します。各ストリーム記述子の出力には、`DescribeJournalKinesisStream` によって返されるものと同じ詳細が含まれます。
- `CancelJournalKinesisStream` - 指定された QLDB ジャーナルストリーミングを終了します。ストリームをキャンセルするには、その現在のステータスが `ACTIVE` である必要があります。

  キャンセルしたストリームを再開することはできません。Kinesis Data Streams へのデータの配信を再開するには、新しい QLDB ストリーミングを作成します。

これらの API オペレーションの詳細については、Amazon QLDB API リファレンス を参照してください。

を使用したジャーナルストリームの作成と管理については AWS CLI、 AWS CLI コマンドリファレンス を参照してください。

サンプルアプリケーション

QLDB には、ジャーナルストリーミングを使用するさまざまなオペレーションのデモを行うサンプルアプリケーションがあります。これらのアプリケーションはAWS 、サンプル GitHub サイト のオープンソースです。

トピック

- 基本的な演算 (Java)
- OpenSearch サービスとの統合 (Python)

- Amazon SNS と Amazon SQS との統合 (Python)

基本的な演算 (Java)

QLDB ジャーナルストリームの基本的なオペレーションを示す Java コード例については、 GitHub リポジトリ aws-samples/-java amazon-qldb-dmv-sampleを参照してください。このサンプルアプリケーションをダウンロードしてインストールする方法については、「Amazon QLDB Java サンプルアプリケーションのインストール」を参照してください。

> **Note**
>
> アプリケーションをインストールした後、台帳を作成するために、Java チュートリアルのステップ 1 に進まないでください。ストリーミング用のこのサンプルアプリケーションでは、`vehicle-registration` の台帳が作成されます。

このサンプルアプリケーションでは、「Java チュートリアル」の完全なソースコードとその依存関係 (以下のモジュールを含む) がパッケージ化されています。

- AWS SDK for Java - 台帳、QLDB ジャーナルストリーミング、および Kinesis データストリームを含む、QLDB および Kinesis Data Streams の両方のリソースを作成および削除します。
- Java 用 Amazon QLDB ドライバー - テーブルの作成やドキュメントの挿入など、PartiQL ステートメントを使用して台帳でデータトランザクションを実行します。
- Kinesis Client Library - Kinesis データストリームのデータを消費して処理します。

コードの実行

StreamJournal クラスには、以下のオペレーションを示すチュートリアルコードが含まれています。

1. `vehicle-registration` という名前の台帳を作成し、テーブルを作成して、それらをサンプルデータとともにロードします。

   > **Note**
   >
   > このコードを実行する前に、`vehicle-registration` という名前の有効な台帳がないことを確認してください。

2. Kinesis データストリーム、QLDB が Kinesis データストリームの書き込み許可を引き受けられるようにする IAM ロール、および QLDB ジャーナルストリーミングを作成します。
3. KCL を使用して、Kinesis データストリームを処理し、各 QLDB データレコードを記録するストリーミングリーダーを起動します。
4. ストリームデータを使用して、`vehicle-registration` サンプル台帳のハッシュチェーンを検証します。
5. ストリーミングリーダーを停止し、QLDB ジャーナルストリーミングをキャンセルして台帳を削除し、Kinesis データストリームを削除することにより、すべてのリソースをクリーンアップします。

`StreamJournal` チュートリアルコードを実行するには、プロジェクトのルートディレクトリから次の Gradle コマンドを入力します。

```
./gradlew run -Dtutorial=streams.StreamJournal
```

OpenSearch サービスとの統合 (Python)

QLDB ストリームを Amazon OpenSearch Service と統合する方法を示す Python サンプルアプリケーションについては、 GitHub リポジトリ aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python を参照してください。このアプリケーションは、 AWS Lambda 関数を使用して Kinesis Data Streams コンシューマーを実装します。

リポジトリのクローンを作成するには、次の `git` コマンドを入力します。

```
git clone https://github.com/aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python.git
```

サンプルアプリケーションを実行するには、「」の「README GitHub 」を参照してください。

Amazon SNS と Amazon SQS との統合 (Python)

QLDB ストリームを Amazon Simple Notification Service (Amazon SNS) と統合する方法を示す Python サンプルアプリケーションについては、 GitHub リポジトリ aws-samples/amazon-qldb-streams-dmv-sample-lambda-python を参照してください。

このアプリケーションは、 AWS Lambda 関数を使用して Kinesis Data Streams コンシューマーを実装します。これは、Amazon SNS トピックにメッセージを送信します。トピックには、Amazon Simple Queue Service (Amazon SQS) キューがサブスクライブされています。

リポジトリのクローンを作成するには、次の git コマンドを入力します。

```
git clone https://github.com/aws-samples/amazon-qldb-streams-dmv-sample-lambda-
python.git
```

サンプルアプリケーションを実行する方法については、「」の「README GitHub 」を参照してください。

Kinesis での QLDB ストリーミングレコード

Amazon QLDB ストリーミングは、コントロール、ブロックサマリー、およびリビジョンの詳細という 3 種類のデータレコードを指定の Amazon Kinesis Data Streams リソースに書き込みます。3 つのレコードタイプはすべて、Amazon Ion 形式のバイナリ表現で記述されます。

コントロールレコードは、QLDB ストリーミングの開始と完了を示します。QLDB ストリーミングは、リビジョンがジャーナルにコミットされるたびに、関連するすべてのジャーナルブロックデータをブロックサマリーおよびリビジョンの詳細レコードに書き込みます。

3 つのレコードタイプは多相型です。これらはすべて、QLDB ストリーミング ARN、レコードタイプ、およびレコードペイロードを含む共通の最上位レコードで構成されます。この最上位レベルのレコードは、次の形式になります。

```
{
  qldbStreamArn: string,
  recordType: string,
  payload: {
    //control | block summary | revision details record
  }
}
```

recordType フィールドには、次の 3 つの値のいずれかを指定できます。

- CONTROL
- BLOCK_SUMMARY
- REVISION_DETAILS

次のセクションでは、個々のペイロードレコードの形式と内容について説明します。

> **Note**
>
> QLDB は、Amazon Ion のバイナリ表現で Kinesis Data Streams にすべてのストリーミングレコードを書き込みます。以下の例は、Ion のテキスト表現で、レコードの内容を読みやすい形式で説明しています。

トピック

コントロールレコード

QLDB ストリーミングは、開始イベントと完了イベントを示すコントロールレコードを書き込みます。次に、それぞれの `controlRecordType` のサンプルデータを含むコントロールレコードの例を示します。

- `CREATED` - 新しく作成されたストリーミングがアクティブであることを示すために、QLDB ストリーミングが Kinesis に書き込む最初のレコード。

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"CONTROL",
  payload:{
    controlRecordType:"CREATED"
  }
}
```

- `COMPLETED` - ストリーミングが指定された終了日時に達したこと示すために、QLDB ストリーミングが Kinesis に書き込む最後のレコード。ストリームをキャンセルすると、このレコードは書き込まれません。

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
```

```
  recordType:"CONTROL",
  payload:{
    controlRecordType:"COMPLETED"
  }
}
```

ブロックサマリーレコード

ブロックサマリーレコードは、ドキュメントのリビジョンがコミットされるジャーナルブロックを表します。ブロックは、トランザクション中に QLDB ジャーナルにコミットされるオブジェクトです。

ブロックサマリーレコードのペイロードには、ブロックをコミットしたトランザクションのブロックアドレス、タイムスタンプ、およびその他のメタデータが含まれます。また、ブロック内のリビジョンのサマリー属性と、それらをコミットした PartiQL ステートメントも含まれます。以下は、サンプルデータを含むブロックサマリーレコードの例です。

> **Note**
>
> このブロックサマリーの例は、情報提供のみを目的として記載されています。示されているハッシュは、実際に計算されたハッシュ値ではありません。

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"BLOCK_SUMMARY",
  payload:{
    blockAddress:{
      strandId:"ElYL30RGoqrFCbbaQn3K6m",
      sequenceNo:60807
    },
    transactionId:"9RWohCo7My4GGkxRETAJ6M",
    blockTimestamp:2019-09-18T17:00:14.601000001Z,
    blockHash:{{6Pk9KDYJd38ci09oaHxx0D2grtgh4QBBqbDS6i9quX8=}},
    entriesHash:{{r5YoH6+NXDXxgoRzPREGAWJfn73KlZE0eTfbTxZWUDU=}},
    previousBlockHash:{{K3ti0Agk7DEponywKcQCPRYVHb5RuyxdmQFTfrloptA=}},
    entriesHashList:[
      {{pbzvz6ofJC7mD2jvgfyrY/VtR01zIZHoWy8T1VcxlGo=}},
      {{k2brC23DLMercmiOWHiURaGwHu0mQtLzdNPuviE2rcs=}},
```

```
      {{hvw1EV8k4oOkIO36kblO/+UUSFUQqCanKuDGr0aP9nQ=}},
      {{ZrLbkyzDcpJ9KWsZMZqRuKUkG/czLIJ4US+K5E31b+Q=}}
    ],
    transactionInfo:{
      statements:[
        {
          statement:"SELECT * FROM Person WHERE GovId = ?",
          startTime:2019-09-18T17:00:14.587Z,
          statementDigest:{{p4Dn0DiuYD3Xm9UQQ75YLwmoMbSfJmopOmTfMnXs26M=}}
        },
        {
          statement:"INSERT INTO Person ?",
          startTime:2019-09-18T17:00:14.594Z,
          statementDigest:{{klMLkLfa5VJqk6JUPtHkQpOsDdG4HmuUaq/VaApQflU=}}
        },
        {
          statement:"INSERT INTO VehicleRegistration ?",
          startTime:2019-09-18T17:00:14.598Z,
          statementDigest:{{B0gO9BWVNrzRYFoe7t+GVLpJ6uZcLKf5t/chkfRhspI=}}
        }
      ],
      documents:{
        '7z2OpEBgVCvCtwvx4a2JGn':{
          tableName:"Person",
          tableId:"LSkFkQvkIOjCmpTZpkfpn9",
          statements:[1]
        },
        'K0FpsSLpydLDr7hi6KUzqk':{
          tableName:"VehicleRegistration",
          tableId:"Ad3A07z0ZffC7Gpso7BXyO",
          statements:[2]
        }
      }
    },
    revisionSummaries:[
      {
        hash:{{uDthuiqSy4FwjZssyCiyFd90XoPSlIwomHBdF/OrmkE=}},
        documentId:"7z2OpEBgVCvCtwvx4a2JGn"
      },
      {
        hash:{{qJID/amu0gN3dpG5Tg0FfIFTh/U5yFkfT+g/O6k5sPM=}},
        documentId:"K0FpsSLpydLDr7hi6KUzqk"
      }
    ]
```

```
  }
}
```

revisionSummaries フィールドでは、一部のリビジョンに documentId がない場合があります。これらは、ユーザーデータを含まない内部のみのシステムリビジョンです。これらのリビジョンのハッシュはジャーナルのハッシュチェーン全体の一部であるため、QLDB ストリームでは、リビジョンが対応するブロックサマリーレコードに含まれます。暗号検証には、完全なハッシュチェーンが必要です。

次のセクションで説明するように、ドキュメント ID を持つリビジョンだけが、個別のリビジョン詳細レコードで発行されます。

リビジョン詳細レコード

リビジョン詳細レコードは、ジャーナルにコミットされたドキュメントのリビジョンを表します。ペイロードには、リビジョンのコミットされたビューのすべての属性と、関連付けられたテーブル名とテーブル ID が含まれます。次に、サンプルデータを含むリビジョンレコードの例を示します。

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"REVISION_DETAILS",
  payload:{
    tableInfo:{
      tableName:"VehicleRegistration",
      tableId:"Ad3A07z0ZffC7Gpso7BXyO"
    },
    revision:{
      blockAddress:{
        strandId:"ElYL30RGoqrFCbbaQn3K6m",
        sequenceNo:60807
      },
      hash:{{qJID/amu0gN3dpG5Tg0FfIFTh/U5yFkfT+g/O6k5sPM=}},
      data:{
        VIN:"1N4AL11D75C109151",
        LicensePlateNumber:"LEWISR261LL",
        State:"WA",
        City:"Seattle",
        PendingPenaltyTicketAmount:90.25,
        ValidFromDate:2017-08-21,
        ValidToDate:2020-05-11,
        Owners:{
```

```
          PrimaryOwner:{PersonId:"7z2OpEBgVCvCtwvx4a2JGn"},
          SecondaryOwners:[]
        }
      },
      metadata:{
        id:"K0FpsSLpydLDr7hi6KUzqk",
        version:0,
        txTime:2019-09-18T17:00:14.602Z,
        txId:"9RWohCo7My4GGkxRETAJ6M"
      }
    }
  }
}
```

重複する レコードと out-of-order レコードの処理

QLDB ストリームは、Kinesis Data Streams に重複する および out-of-order レコードを発行できます。したがって、コンシューマーアプリケーションは、そのようなシナリオを特定して処理するために、独自のロジックを実装する必要があります。ブロックサマリーレコードとリビジョン詳細レコードには、この目的で使用できるフィールドが含まれます。これらのフィールドは、ダウンストリームサービスの機能と組み合わせて、一意の ID とレコードの厳密な順序の両方を示すことができます。

例えば、QLDB を OpenSearch インデックスと統合して、ドキュメントに対するフルテキスト検索機能を提供するストリームを考えてみましょう。このユースケースでは、ドキュメントの古い (out-of-order) リビジョンのインデックス作成を回避する必要があります。順序付けと重複除外を強制するには、 のドキュメント ID フィールドと外部バージョンフィールド OpenSearch、およびリビジョン詳細レコードのドキュメント ID フィールドとバージョンフィールドを使用できます。

QLDB を Amazon OpenSearch Service と統合するサンプルアプリケーションの重複排除ロジックの例については、 GitHub リポジトリ aws-samples/amazon-qldb-streaming-amazon- opensearch-service-sample-pythonを参照してください。

QLDB のストリーミング許可

Amazon QLDB ストリーミングを作成する前に、指定した Amazon Kinesis Data Streams リソースへの書き込み許可を QLDB に付与する必要があります。Kinesis ストリーミングのサーバー側の暗号化にカスタマーマネージド AWS KMS key を使用する場合、指定した対称暗号化キーを使用する許可も QLDB に付与する必要があります。Kinesis Data Streams では非対称 KMS キーはサポートされていません。

QLDB ストリーミングに必要なアクセス許可を提供するために、適切なアクセス許可ポリシーを持つ IAM サービスロールを QLDB が引き受けることができます。サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに権限を委任するロールの作成」を参照してください。

> **Note**
>
> ジャーナルストリーミングをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで `iam:PassRole` アクションを実行するための許可が必要です。QLDB ストリーミングサブリソースに対する `qldb:StreamJournalToKinesis` 許可に加えて、この許可も必要です。
>
> IAM を使用して QLDB へのアクセスを制御する方法については、「Amazon QLDB で IAM が機能する仕組み」を参照してください。QLDB ポリシーの例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

この例では、ユーザーに代わって、Kinesis データストリームにデータレコードを書き込むことを QLDB に許可するロールを作成します。詳細については、「IAM ユーザーガイド」の「AWS のサービスに権限を委任するロールの作成」を参照してください。

で QLDB ジャーナル AWS アカウント を初めてストリーミングする場合は、まず以下の操作を行って、適切なポリシーで IAM ロールを作成する必要があります。または、QLDB コンソールを使用して、ロールを自動的に作成できます。それ以外の場合は、前に作成したロールを選択できます。

トピック

- 許可ポリシーを作成する
- IAM ロールを作成する

許可ポリシーを作成する

以下のステップを実行して、QLDB ストリーミングに許可ポリシーを作成します。以下の例は、指定した Kinesis データストリームにデータレコードを書き込む許可を QLDB に付与する Kinesis Data Streams ポリシーを示しています。該当する場合、QLDB に対称暗号化 KMS キーの使用を許可するキーポリシーも示しています。

Kinesis Data Streams ポリシーの詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「IAM を使用して Amazon Kinesis Data Streams リソースへのアクセスを制御する」と

「ユーザー生成 KMS キーを使用するためのアクセス許可」を参照してください。 AWS KMS キーポリシーの詳細については、「 AWS Key Management Service デベロッパーガイド」の「 でのキーポリシー AWS KMSの使用」を参照してください。

> **Note**
>
> Kinesis データストリームと KMS キーは、QLDB 台帳と同じ AWS リージョン とアカウントにある必要があります。

JSON ポリシーエディタでポリシーを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーション列で、[ポリシー] を選択します。

   [Policies] (ポリシー) を初めて選択する場合は、[Welcome to Managed Policies] (マネージドポリシーにようこそ) ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [JSON] タブを選択します。
5. JSON ポリシードキュメントを入力します。
   - Kinesis ストリーミングのサーバー側の暗号化にカスタマーマネージド KMS キーを使用する場合は、以下のポリシードキュメントの例を使用します。このポリシーを使用するには、例の *us-east-1*、*123456789012kinesis-stream-name*、、および *1234abcd-12ab-34cd-56ef-1234567890ab* をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBStreamKinesisPermissions",
            "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
 "kinesis:ListShards" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/kinesis-
stream-name"
        },
        {
            "Sid": "QLDBStreamKMSPermission",
```

```
            "Action": [ "kms:GenerateDataKey" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

- それ以外の場合は、以下のポリシードキュメントの例を使用します。このポリシーを使用するには、例*kinesis-stream-name*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBStreamKinesisPermissions",
            "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
 "kinesis:ListShards" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/kinesis-
stream-name"
        }
    ]
}
```

6. [ポリシーの確認] を選択します。

> **Note**
>
> いつでも [Visual editor] (ビジュアルエディタ) タブと [JSON] タブを切り替えることができます。ただし、[Visual editor] (ビジュアルエディタ) タブで [Review policy] (ポリシーの確認) を変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイドの「ポリシーの再構成」を参照してください。

7. [ポリシーの確認] ページに作成するポリシーの [名前] と [説明] を入力します。ポリシーの [Summary] (概要) を参照して、ポリシーによって付与された許可を確認します。次に、[Create policy] (ポリシーの作成) を選択して作業を保存します。

IAM ロールを作成する

QLDB ストリーミングの許可ポリシーを作成したら、次に IAM ロールを作成し、ポリシーをアタッチします。

QLDB のサービスロールを作成するには (IAM コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成] を選択します。
3. 信頼できるエンティティタイプ で、AWS のサービス を選択します。
4. [サービス] または [ユースケース] で [QLDB] を選択し、次に [QLDB] ユースケースを選択します。
5. [次へ] をクリックします。
6. 前のステップで作成したポリシーの横にあるボックスをオンにします。
7. (オプション) アクセス許可の境界を設定します。このアドバンスド機能は、サービスロールで使用できますが、サービスにリンクされたロールではありません。

   a. [アクセス許可の境界の設定] セクションを開き、[アクセス許可の境界を使用してロールのアクセス許可の上限を設定する] を選択します。

      IAM には、アカウント内の AWS 管理ポリシーとカスタマー管理ポリシーのリストが含まれます。

   b. アクセス許可の境界として使用するポリシーを選択します。
8. [次へ] をクリックします。
9. このロールの目的を識別しやすいロール名またはロール名サフィックスを入力します。

> **Important**
>
> ロールに名前を付けるときは、次のことに注意してください。
>
> - ロール名は 内で一意である必要があり AWS アカウント、大文字と小文字を区別することはできません。
>
>   例えば、**`PRODROLE`** と **`prodrole`** の両方の名前でロールを作成することはできません。ロール名がポリシーまたは ARN の一部として使用される場合、ロール名は大文

字と小文字が区別されます。ただし、サインインプロセスなど、コンソールにロール名がユーザーに表示される場合、ロール名は大文字と小文字が区別されません。

- 他のエンティティがロールを参照する可能性があるため、ロールを作成した後にロール名を編集することはできません。

10. (オプション) [説明] にロールの説明を入力します。
11. (オプション) ロールのユースケースとアクセス許可を編集するには、[ステップ 1: 信頼されたエンティティを選択] または [ステップ 2: アクセス権限を追加] のセクションで [編集] を選択します。
12. (オプション) ロールの識別、整理、検索を簡単にするには、キーと値のペアとしてタグを追加します。IAM でのタグの使用に関する詳細については、『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。
13. ロールを確認したら、[Create role] (ロールを作成) を選択します。

以下の JSON ドキュメントは、特定の許可がアタッチされた IAM ロールを引き受けることを QLDB に許可する信頼ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-
east-1:123456789012:stream/myExampleLedger/*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

> **Note**
>
> この信頼ポリシーの例では、`aws:SourceArn` および `aws:SourceAccount` グローバル条件コンテキストキーを使用して、混乱した代理問題を回避する方法を示します。この信頼ポリシーを使用すると、QLDB は台帳 `123456789012` のみでアカウント `myExampleLedger` の任意の QLDB ストリーミングに対応するロールを引き受けることができます。
> 詳細については、「クロスサービスの混乱した副防止」を参照してください。

IAM ロールを作成した後、QLDB コンソールに戻り、[Create QLDB stream] (QLDB ストリームを作成) ページを更新すると、新しいロールが見つかります。

QLDB のジャーナルストリーミングの一般的なエラー

このセクションでは、ジャーナルストリーミングのリクエストに対して Amazon QLDB がスローするランタイムエラーについて説明します。

以下は、サービスによって返される一般的な例外のリストです。それぞれの例外には、特定のエラーメッセージに加え、簡単な説明と考えられる解決方法に関する推奨事項が記載されています。

AccessDeniedException

メッセージ: ユーザー: *userARN* は、次の操作を実行する権限がありません: iam:PassRole on resource: *roleARN*

IAM ロールを QLDB サービスに渡す許可がありません。QLDB には、すべてのジャーナルストリーミングリクエストに対応するロールが必要であり、このロールを QLDB に渡す許可が必要です。このロールは、指定した Amazon Kinesis Data Streams リソースでの書き込み許可を QLDB に付与します。

指定した IAM ロールリソースに対して `PassRole` API オペレーションを実行する許可を付与する IAM ポリシーを QLDB サービス (`qldb.amazonaws.com`) に定義していることを確認します。ポリシーの例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

IllegalArgumentException

メッセージ: QLDB encountered an error validating Kinesis Data Streams (Kinesis Data Streams の検証中にエラーが発生しました): Kinesis からのレスポンス: *errorCode errorMessage*

このエラーの原因として考えられるのは、存在しない Kinesis Data Streams リソースが指定されていることです。または、QLDB には、指定した Kinesis データストリームにデータレコードを書き込むための十分な許可がありません。

ストリーミングリクエストに指定した Kinesis データストリームが正しいことを確認します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「データストリームの作成および更新」を参照してください。

以下のアクションに対して QLDB サービス (`qldb.amazonaws.com`) 許可を付与する、指定した Kinesis データストリーム用のポリシーを定義していることも確認します。詳細については、「ストリームアクセス許可」を参照してください。

- `kinesis:PutRecord`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`
- `kinesis:ListShards`

IllegalArgumentException

メッセージ: Unexpected response from Kinesis Data Streams while validating the Kinesis configuration. Response from Kinesis: *`errorCode errorMessage`* (Kinesis 設定の検証中に、Kinesis Data Streams から予期しないレスポンスがありました。Kinesis からのレスポンス: errorCode errorMessage)

指定した Kinesis データストリームにデータレコードを書き込もうとしましたが、Kinesis エラーレスポンスがあり失敗しました。考えられる原因については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Amazon Kinesis Data Streams プロデューサーのトラブルシューティング」を参照してください。

IllegalArgumentException

メッセージ: Start date must not be greater than end date. (開始日は終了日よりも前の日付でなければなりません。)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、ISO 8601 の日時形式、協定世界時 (UTC) にしてください。

IllegalArgumentException

メッセージ: Start date cannot be in the future (開始日を現在よりも後の日付にすることはできません)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、`ISO 8601` の日時形式、UTC にしてください。

LimitExceededException

メッセージ: Exceeded the limit of 5 concurrently running Journal streams to Kinesis Data Streams (Kinesis Data Streams への同時実行ジャーナルストリームの 5 つの制限を超えました)

QLDB では、一度に実行できるジャーナルストリーミングのデフォルト上限、最大 5 件が適用されます。

Amazon QLDB での元帳管理

この章では、QLDB API、AWS Command Line Interface (AWS CLI)、および AWS CloudFormation を使用して、Amazon QLDB で台帳管理オペレーションを行う方法について説明します。

このタスクは AWS Management Console を使用して実行することもできます。詳細については、「コンソールを使用した Amazon QLDB へのアクセス」を参照してください。

トピック

- Amazon QLDB 台帳の基本的なオペレーション
- AWS CloudFormation を使用した Amazon QLDB リソースの作成
- Amazon QLDB リソースのタグ付け

Amazon QLDB 台帳の基本的なオペレーション

QLDB API または AWS Command Line Interface (AWS CLI) を使用して、Amazon QLDB で台帳を作成、更新、削除できます。アカウントのすべての台帳を一覧表示したり、特定の台帳に関する情報を取得したりもできます。

次のトピックでは、AWS SDK for Java と AWS CLI を使用した台帳オペレーションの一般的な手順を示す短いコード例について説明します。

トピック

- 台帳の作成
- 台帳の説明
- 台帳の更新
- 台帳アクセス許可モードの更新
- 台帳の削除
- 台帳の一覧表示

これらのオペレーションを完全なサンプルアプリケーションで実行するコード例については、以下の「ドライバーの開始方法」のチュートリアルと GitHub リポジトリを参照してください。

- Java: チュートリアル | GitHub リポジトリ

- Node.js: チュートリアル | GitHub リポジトリ
- Python: チュートリアル | GitHub リポジトリ

台帳の作成

`CreateLedger` オペレーションを使用して、AWS アカウント に台帳を作成します。これには、以下の情報を入力する必要があります。

- [Ledger name] (台帳名) - アカウントに作成する台帳の名前。この名前は現在の AWS リージョンのすべての台帳間で一意であることが必要です。

  台帳名の命名に関する制約は「Amazon QLDB でのクォータと制限」で定義されています。
- [Permissions mode] (アクセス許可モード) – 台帳に割り当てるアクセス許可モード。以下のオプションのいずれかを選択します。
  - [Allow all] (すべて許可) - 台帳の API レベル詳細度でアクセスコントロールを可能にする、レガシーアクセス許可モード。

    このモードは、この台帳の SendCommand API アクセス許可を持つユーザーが、指定された台帳の任意のテーブルで、すべての PartiQL コマンド (すなわち `ALLOW_ALL`) を実行することを許可します。このモードでは、台帳用に作成したテーブルレベルまたはコマンドレベルの IAM アクセス許可ポリシーはすべて無視されます。
  - [Standard] (標準) - (推奨) 台帳、テーブル、PartiQL コマンドの、より詳細なレベルのアクセスコントロールを可能にするアクセス許可モード。台帳データのセキュリティを最大化する、このアクセス許可モードの使用を強く推奨します。

    デフォルトでは、このモードは、この台帳内の任意のテーブルで、任意の PartiQL コマンドを実行するすべてのリクエストを拒否します。PartiQL コマンドを許可するには、台帳の SendCommand API アクセス許可に加えて、特定のテーブルリソースと PartiQL アクション用の IAM アクセス許可ポリシーを作成する必要があります。詳細については、Amazon QLDB の標準アクセス許可モードの開始方法を参照してください。
- [Deletion protection] (削除保護) - (オプション) ユーザーが台帳を削除できないようにするフラグ。台帳の作成時に指定しない場合、この機能はデフォルトで有効 (`true`) です。

  削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。無効にするには、`UpdateLedger` オペレーションを使用してフラグを `false` に設定します。
- AWS KMS key — (オプション) 保管中のデータの暗号化に使用する AWS Key Management Service (AWS KMS) のキー。次のいずれかの種類の AWS KMS keys を選択します。

- AWS 所有の KMS キー - ユーザーに代わって AWS が所有、管理している KMS キーを使用します。

  台帳の作成時にこのパラメータを定義しない場合、台帳ではデフォルトでこの種類のキーを使用します。このキーの種類を指定するときに文字列 `AWS_OWNED_KMS_KEY` を使用することもできます。このオプションには追加のセットアップは必要ありません。

- カスタマーマネージド KMS キー - 作成、所有、管理しているアカウントで、対称暗号化 KMS キーを使用します。QLDB は非対称キーをサポートしていません。

  このオプションを使用するには、KMS キーを作成するか、アカウント内の既存のキーを使用する必要があります。カスタマーマネージドキーの作成方法については、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 KMS キーの作成」を参照してください。

  カスタマーマネージド KMS キーは、ID、エイリアス、または Amazon リソースネーム (ARN) を使用して指定できます。詳細については、「AWS Key Management Service デベロッパーガイド」の「キー識別子 (KeyId)」を参照してください。

  > **Note**
  >
  > クロスリージョンキーはサポートされていません。指定した KMS キーは台帳と同じ AWS リージョン に存在する必要があります。

  詳細については、「Amazon QLDB での保管時の暗号化」を参照してください。

- [Tags] (タグ) - (オプション) タグをキーと値の組み合わせとしてアタッチすることで、メタデータを台帳に追加します。台帳にタグを追加すると、台帳の整理と識別がしやすくなります。詳細については、「Amazon QLDB リソースのタグ付け」を参照してください。

台帳は、QLDB によって作成され、ステータスが `ACTIVE` に設定されるまで、使用可能になりません。

台帳の作成 (Java)

AWS SDK for Java を使用して台帳を作成するには

1. `AmazonQLDB` クラスのインスタンスを作成します。
2. リクエスト情報を指定する `CreateLedgerRequest` クラスのインスタンスを作成します。

台帳名とアクセス許可モードを指定する必要があります。

3. リクエストオブジェクトをパラメータとして指定して、createLedger メソッドを実行します。

createLedger リクエストは、台帳に関する情報を含む CreateLedgerResult オブジェクトを返します。DescribeLedger オペレーションを使用して台帳の作成後に台帳のステータスを確認する例については、次のセクションを参照してください。

次の例は、前述のステップを示しています。

Example — 既定の構成設定を使用する

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
CreateLedgerRequest request = new CreateLedgerRequest()
        .withName(ledgerName)
        .withPermissionsMode(PermissionsMode.STANDARD);
CreateLedgerResult result = client.createLedger(request);
```

> **Note**
>
> 台帳では、指定しない場合、次のデフォルト設定が使用されます。
>
> - 削除保護 - 有効 (true)。
> - KMS キー — AWS 所有の KMS キー。

Example — 削除保護を無効にし、カスタマーマネージド KMS キーを使用し、タグをアタッチする

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();

Map<String, String> tags = new HashMap<>();
tags.put("IsTest", "true");
tags.put("Domain", "Test");

CreateLedgerRequest request = new CreateLedgerRequest()
        .withName(ledgerName)
        .withPermissionsMode(PermissionsMode.STANDARD)
        .withDeletionProtection(false)
```

```
        .withKmsKey("arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
        .withTags(tags);
CreateLedgerResult result = client.createLedger(request);
```

台帳の作成 (AWS CLI)

デフォルトの構成設定を使用して、vehicle-registration という名前の新しい台帳を作成します。

Example

```
aws qldb create-ledger --name vehicle-registration --permissions-mode STANDARD
```

> **Note**
>
> 台帳では、指定しない場合、次のデフォルト設定が使用されます。
>
> - 削除保護 - 有効 (true)。
> - KMS キー — AWS 所有の KMS キー。

または、vehicle-registration という名前の新しい台帳を作成し、その台帳に対して削除保護を無効にし、カスタマーマネージド KMS キーとタグを指定します。

Example

```
aws qldb create-ledger \
    --name vehicle-registration \
    --no-deletion-protection \
    --permissions-mode STANDARD \
    --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --tags IsTest=true,Domain=Test
```

台帳の作成 (AWS CloudFormation)

また、AWS CloudFormation テンプレートを使用して台帳を作成することもできます。詳細については、「AWS CloudFormation ユーザーガイド」内の「AWS::QLDB::Ledger」リソースを参照してください。

台帳の説明

`DescribeLedger` オペレーションを使用して、台帳に関する詳細を表示します。台帳名を指定する必要があります。`DescribeLedger` の出力は `CreateLedger` の出力と同じ形式です。次の情報が含まれています。

- [Ledger name] (台帳名) - 説明を表示する台帳の名前。
- [ARN] - 台帳の Amazon リソースネーム (ARN)。形式は以下のとおりです。

```
arn:aws:qldb:aws-region:account-id:ledger/ledger-name
```

- [Deletion protection] (削除保護) - 削除保護機能が有効になっているかどうかを示すフラグ。
- [Creation date and time] (作成日時) - 台帳が作成された日時 (エポック時刻形式)。
- [State] (状態) - 台帳の現在のステータス。これは、以下の値のいずれかになります。
  - `CREATING`
  - `ACTIVE`
  - `DELETING`
  - `DELETED`
- [Permissions mode] (アクセス許可モード) – 台帳に割り当てられるアクセス許可モード。これは、以下の値のいずれかになります。
  - `ALLOW_ALL` - 台帳の API レベル詳細度でアクセスコントロールを可能にする、レガシーアクセス許可モード。
  - `STANDARD` - 台帳、テーブル、PartiQL コマンドの、より詳細なレベルのアクセスコントロールを可能にするアクセス許可モード。
- [Encryption description] (暗号化の説明) - 台帳の保管中のデータの暗号化に関する情報。これには以下の項目が含まれます。
  - [AWS KMS key ARN] - 台帳が保管時の暗号化に使用する、カスタマーマネージド KMS キーの ARN。これが未定義の場合、台帳は、AWS 所有の KMS キーを暗号化に使用します。
  - [Encryption status] (暗号化ステータス) — 台帳の保管時の暗号化の現在のステータス。これは、以下の値のいずれかになります。
    - `ENABLED` — 暗号化は、指定されたキーを使用して完全に有効になります。
    - `UPDATING` — 指定されたキーの変更はアクティブに処理されています。

QLDB のキーの変更は非同期です。キーの変更が処理されている間も、パフォーマンスに影響を与えることなく、台帳にフルアクセスできます。キーの更新にかかる時間は、台帳のサイズによって異なります。

- KMS_KEY_INACCESSIBLE — 指定されたカスタマーマネージド KMS キーにアクセスできず、台帳に障害が発生しています。キーが無効化または削除されたか、キーの許可が取り消されました。台帳に障害が発生すると、アクセスできず、読み取りまたは書き込みリクエストも受け付けません。

  障害のある台帳は、キーの許可を復元した後、または無効化されたキーを再度有効にした後に、自動的にアクティブ状態に戻ります。ただし、カスタマーマネージド KMS キーの削除は元に戻せません。キーを削除すると、そのキーで保護されている台帳にアクセスできなくなり、データが完全に回復不可能になります。

- [アクセス不可の AWS KMS key] – エラー発生時、KMS キーが最初にアクセス不可になった日時 (エポック時刻形式)。

  KMS キーがアクセス可能な場合、これは未定義です。

詳細については、「Amazon QLDB での保管時の暗号化」を参照してください。

> **Note**
>
> QLDB 台帳を作成した後、そのステータスが CREATING から ACTIVE に変わると、使用可能になります。

台帳の説明 (Java)

AWS SDK for Java を使用して台帳を記述するには

1. AmazonQLDB クラスのインスタンスを作成します。または、CreateLedger リクエストに対してインスタンス化した AmazonQLDB クライアントの同じインスタンスを使用できます。
2. DescribeLedgerRequest クラスのインスタンスを作成し、説明を表示する台帳の名前を指定します。
3. リクエストオブジェクトをパラメータとして指定して、describeLedger メソッドを実行します。

4. `describeLedger` リクエストは、台帳に関する現在の情報を含む `DescribeLedgerResult` オブジェクトを返します。

以下のサンプルコードは、前述のステップの例です。クライアントの `describeLedger` メソッドを呼び出せば、いつでも台帳の情報を収集できます。

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
DescribeLedgerRequest request = new DescribeLedgerRequest().withName(ledgerName);
DescribeLedgerResult result = client.describeLedger(request);
System.out.printf("%s: ARN: %s \t State: %s \t CreationDateTime: %s \t
 DeletionProtection: %s
                   \t PermissionsMode: %s \t EncryptionDescription: %s",
        result.getName(),
        result.getArn(),
        result.getState(),
        result.getCreationDateTime(),
        result.getDeletionProtection(),
        result.getPermissionsMode(),
        result.getEncryptionDescription());
```

台帳の説明 (AWS CLI)

作成した `vehicle-registration` 台帳の説明を表示します。

Example

```
aws qldb describe-ledger --name vehicle-registration
```

台帳の更新

`UpdateLedger` オペレーションでは、既存台帳の次の構成設定を変更できます。

- [Deletion protection] (削除保護) - ユーザーが台帳を削除できないようにするフラグ。この機能が有効な場合は、台帳を削除するためにフラグを `false` に設定して無効にする必要があります。

  このパラメータを定義しない場合、台帳の削除保護設定は変更されません。
- [AWS KMS key] – 保管中のデータの暗号化に使用する AWS Key Management Service (AWS KMS) のキー。このパラメータを定義しない場合、台帳の KMS キーは変更されません。

> **Note**
>
> Amazon QLDB は、カスタマーマネージド AWS KMS keysのサポートを 2021 年 7 月 22 日に開始しました。ローンチ前に作成された台帳は、デフォルトでは AWS 所有のキー によって保護されますが、カスタマーマネージドキーを使用した保管時の暗号化には現在適用されません。
> 台帳の作成時間は QLDB コンソールで確認できます。

以下のいずれかのオプションを使用します。

- AWS 所有の KMS キー - ユーザーに代わって AWS が所有、管理している KMS キーを使用します。この種類のキーを使用するには、このパラメーターに文字列 `AWS_OWNED_KMS_KEY` を指定します。このオプションには追加のセットアップは必要ありません。
- カスタマーマネージド KMS キー - 作成、所有、管理しているアカウントで、対称暗号化 KMS キーを使用します。QLDB は非対称キーをサポートしていません。

  このオプションを使用するには、KMS キーを作成するか、アカウント内の既存のキーを使用する必要があります。カスタマーマネージドキーの作成方法については、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 KMS キーの作成」を参照してください。

  カスタマーマネージド KMS キーは、ID、エイリアス、または Amazon リソースネーム (ARN) を使用して指定できます。詳細については、「AWS Key Management Service デベロッパーガイド」の「キー識別子 (KeyId)」を参照してください。

  > **Note**
  >
  > クロスリージョンキーはサポートされていません。指定した KMS キーは台帳と同じ AWS リージョン に存在する必要があります。

QLDB のキーの変更は非同期です。キーの変更が処理されている間も、パフォーマンスに影響を与えることなく、台帳にフルアクセスできます。

キーは必要に応じて切り替えることができますが、キーの更新にかかる時間は、台帳のサイズによって異なります。`DescribeLedger` オペレーションを使用して保管時の暗号化のステータスを確認します。

詳細については、「Amazon QLDB での保管時の暗号化」を参照してください。

UpdateLedger の出力は CreateLedger の出力と同じ形式です。

台帳の更新 (Java)

AWS SDK for Java を使用して台帳を更新するには

1. AmazonQLDB クラスのインスタンスを作成します。
2. リクエスト情報を指定する UpdateLedgerRequest クラスのインスタンスを作成します。

   台帳名と、削除保護の新しいブール値、KMS キーの新しい文字列値を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、updateLedger メソッドを実行します。

次のコード例は、前述のステップを示しています。updateLedger リクエストは、台帳に関する情報を更新した UpdateLedgerResult オブジェクトを返します。

Example - 削除保護の無効化

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withDeletionProtection(false);
UpdateLedgerResult result = client.updateLedger(request);
```

Example - カスタマーマネージド KMS キーの使用

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withKmsKey("arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
UpdateLedgerResult result = client.updateLedger(request);
```

Example - AWS 所有 の KMS キーの使用

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
```

```
UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withKmsKey("AWS_OWNED_KMS_KEY")
UpdateLedgerResult result = client.updateLedger(request);
```

台帳の更新 (AWS CLI)

vehicle-registration 台帳に対して削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。

Example

```
aws qldb update-ledger --name vehicle-registration --no-deletion-protection
```

台帳の保管時の暗号化設定を変更して、カスタマーマネージド KMS キーを使用することもできます。

Example

```
aws qldb update-ledger --name vehicle-registration --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

または、保管時の暗号化設定を変更して、AWS 所有の KMS キーを使用することもできます。

Example

```
aws qldb update-ledger --name vehicle-registration --kms-key AWS_OWNED_KMS_KEY
```

台帳アクセス許可モードの更新

UpdateLedgerPermissionsMode オペレーションでは、既存台帳のアクセス許可を変更できます。以下のオプションのいずれかを選択します。

- [Allow all] (すべて許可) - 台帳の API レベル詳細度でアクセスコントロールを可能にする、レガシーアクセス許可モード。

  このモードは、この台帳の SendCommand API アクセス許可を持つユーザーが、指定された台帳の任意のテーブルで、すべての PartiQL コマンド (すなわち ALLOW_ALL) を実行することを許可します。このモードでは、台帳用に作成したテーブルレベルまたはコマンドレベルの IAM アクセス許可ポリシーはすべて無視されます。

- [Standard] (標準) - (推奨) 台帳、テーブル、PartiQL コマンドの、より詳細なレベルのアクセスコントロールを可能にするアクセス許可モード。台帳データのセキュリティを最大化する、このアクセス許可モードの使用を強く推奨します。

  デフォルトでは、このモードは、この台帳内の任意のテーブルで、任意の PartiQL コマンドを実行するすべてのリクエストを拒否します。PartiQL コマンドを許可するには、台帳の SendCommand API アクセス許可に加えて、特定のテーブルリソースと PartiQL アクション用の IAM アクセス許可ポリシーを作成する必要があります。詳細については、Amazon QLDB の標準アクセス許可モードの開始方法を参照してください。

> **Important**
>
> STANDARD アクセス許可モードに切り替える前に、ユーザーの混乱を避けるために、まず必要な IAM ポリシーとテーブルタグをすべて作成する必要があります。詳細については、「標準アクセス許可モードへの移行」を参照してください。

台帳アクセス許可モードの更新 (Java)

AWS SDK for Java を使用して台帳アクセス許可モードを更新するには

1. AmazonQLDB クラスのインスタンスを作成します。
2. リクエスト情報を指定する UpdateLedgerPermissionsModeRequest クラスのインスタンスを作成します。

   台帳名と、アクセス許可モードの新しい文字列値を指定する必要があります。
3. リクエストオブジェクトをパラメータとして指定して、updateLedgerPermissionsMode メソッドを実行します。

次のコード例は、前述のステップを示しています。updateLedgerPermissionsMode リクエストは、台帳に関する情報を更新した UpdateLedgerPermissionsModeResult オブジェクトを返します。

Example — 標準のアクセス許可モードを割り当てる

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerPermissionsModeRequest request = new UpdateLedgerPermissionsModeRequest()
        .withName(ledgerName)
```

```
        .withPermissionsMode(PermissionsMode.STANDARD);
UpdateLedgerPermissionsModeResult result = client.updateLedgerPermissionsMode(request);
```

台帳アクセス許可モードの更新 (AWS CLI)

vehicle-registration 台帳に STANDARD アクセス許可モードを割り当てます。

Example

```
aws qldb update-ledger-permissions-mode --name vehicle-registration --permissions-mode
 STANDARD
```

標準アクセス許可モードへの移行

STANDARD アクセス許可モードに移行するには、QLDB アクセスパターンを分析し、リソースにアクセスするために適切なアクセス許可をユーザーに付与する IAM ポリシーを追加することをお勧めします。

STANDARD アクセス許可モードに切り替える前に、まず必要な IAM ポリシーとテーブルタグをすべて作成する必要があります。そうしないと、アクセス許可モードを切り替えることで、正しい IAM ポリシーを作成するかアクセス許可モードを ALLOW_ALL に戻すまで、ユーザーが混乱する可能性があります。これらのポリシーの作成の詳細については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

また、AWS マネージドポリシーを使用して、すべての QLDB リソースへのフルアクセスを付与できます。AmazonQLDBFullAccess と AmazonQLDBConsoleFullAccess マネージドポリシーには、すべての PartiQL アクションを含むすべての QLDB アクションが含まれます。これらのポリシーの 1 つをプリンシパルにアタッチすることは、そのプリンシパルの ALLOW_ALL アクセス許可モードと等価です。詳細については、「AWS Amazon QLDB の マネージドポリシー」を参照してください。

台帳の削除

台帳とそのすべての内容を削除するには、DeleteLedger オペレーションを使用します。台帳の削除は回復不可能なオペレーションです。

台帳に対して削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。

DeleteLedger リクエストを発行すると、台帳の状態は ACTIVE から DELETING に変わります。台帳に使用されているストレージの量によっては、台帳を削除するのに時間がかかる場合があります。DeleteLedger オペレーションが完了すると、台帳は QLDB に存在しなくなります。

台帳の削除 (Java)

AWS SDK for Java を使用して台帳を削除するには

1. AmazonQLDB クラスのインスタンスを作成します。
2. DeleteLedgerRequest クラスのインスタンスを作成し、削除する台帳の名前を指定します。
3. リクエストオブジェクトをパラメータとして指定して、deleteLedger メソッドを実行します。

以下のサンプルコードは、前述のステップの例です。

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
DeleteLedgerRequest request = new DeleteLedgerRequest().withName(ledgerName);
DeleteLedgerResult result = client.deleteLedger(request);
```

台帳の削除 (AWS CLI)

vehicle-registration 台帳を削除します。

Example

```
aws qldb delete-ledger --name vehicle-registration
```

台帳の一覧表示

ListLedgers オペレーションは、現在の AWS アカウント とリージョンのすべての QLDB 台帳の概要情報を返します。

台帳の一覧表示 (Java)

AWS SDK for Java を使用してアカウント内の台帳をリストするには

1. AmazonQLDB クラスのインスタンスを作成します。

2. ListLedgersRequest クラスのインスタンスを作成します。

   前の ListLedgers 呼び出しからのレスポンスで NextToken の値を受け取った場合、結果の次のページを取得するには、このリクエストでその値を指定する必要があります。

3. リクエストオブジェクトをパラメータとして指定して、listLedgers メソッドを実行します。

4. listLedgers リクエストは ListLedgersResult オブジェクトを返します。このオブジェクトには、LedgerSummary オブジェクトのリストと、さらに結果があるかどうかを示すページ分割トークンが含まれます。

   - NextToken が空の場合、結果の最後のページが処理されており、それ以上の結果はありません。
   - NextToken が空でない場合、さらに結果があります。結果の次のページを取得するには、後続の ListLedgers 呼び出しで NextToken の値を使用します。

以下のサンプルコードは、前述のステップの例です。

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
List<LedgerSummary> ledgerSummaries = new ArrayList<>();
String nextToken = null;
do {
    ListLedgersRequest request = new ListLedgersRequest().withNextToken(nextToken);
    ListLedgersResult result = client.listLedgers(request);
    ledgerSummaries.addAll(result.getLedgers());
    nextToken = result.getNextToken();
} while (nextToken != null);
```

台帳の一覧表示 (AWS CLI)

現在の AWS アカウント とリージョンのすべての台帳を一覧表示します。

Example

```
aws qldb list-ledgers
```

AWS CloudFormation を使用した Amazon QLDB リソースの作成

Amazon QLDB は AWS CloudFormation と統合されています。これは、リソースとインフラストラクチャの作成と管理の所要時間を短縮できるように AWS リソースをモデル化して設定するためのサービスです。必要なすべての AWS リソース (QLDB 台帳など) を記述したテンプレートを作成すれば、AWS CloudFormation がお客様に代わってこれらのリソースのプロビジョニングや設定を処理します。

AWS CloudFormation を使用すると、テンプレートを再利用して QLDB リソースを同じように繰り返してセットアップできます。リソースを一度記述するだけで、同じリソースを複数の AWS アカウント とリージョンで何度でもプロビジョニングできます。

QLDB および AWS CloudFormation テンプレート

QLDB および関連サービスのリソースをプロビジョニングして設定するには、AWS CloudFormation テンプレートについて理解しておく必要があります。テンプレートは、JSONまたはYAMLでフォーマットされたテキストファイルです。これらのテンプレートには、AWS CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSON や YAML に不慣れな方は、AWS CloudFormation Designer を使えば、AWS CloudFormation テンプレートを使いこなすことができます。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation Designer とは」を参照してください。

QLDB では、AWS CloudFormation での台帳とジャーナルストリームの作成がサポートされています。台帳およびストリームの JSON テンプレートと YAML テンプレートの例を含む詳細については、「AWS CloudFormation ユーザーガイド」の以下のリソースタイプのリファレンスを参照してください。

- AWS::QLDB::Ledger のリファレンス
- AWS::QLDB::Stream のリファレンス

AWS CloudFormation の詳細はこちら

AWS CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation API リファレンス
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

Amazon QLDB リソースのタグ付け

タグとは、お客様または AWS が AWS リソースに割り当てるカスタム属性ラベルです。各タグは 2 つの部分で構成されます。

- タグキー (例: `CostCenter`、`Environment`、または `Project`)。タグキーでは、大文字と小文字が区別されます。
- タグ値として知られるオプションのフィールド (例: `111122223333` または `Production`)。タグ値を省略すると、空の文字列を使用した場合と同じになります。タグキーと同様に、タグ値は大文字と小文字が区別されます。

タグは、以下のことに役立ちます。

- AWS リソースを識別および整理します。多くの AWS のサービス ではタグ付けがサポートされているため、各種のサービスからリソースに同じタグを割り当てて、リソースの関連を示すことができます。例えば、Amazon S3 バケットに割り当てる同じタグを Amazon QLDB 台帳に割り当てることができます。
- AWS のコストの追跡。これらのタグは、AWS Billing and Cost Management ダッシュボードで有効にします。AWS では、タグを使用してコストを分類し、毎月のコスト配分レポートを提供します。詳細については、AWS Billing ユーザーガイド の「コスト配分タグの使用」を参照してください。
- AWS リソースへのアクセスを AWS Identity and Access Management (IAM) で制御します。詳細については、このデベロッパーガイドの「QLDB での属性ベースのアクセスコントロール (ABAC)」および「IAM ユーザーガイド」の「IAM タグを使用したアクセスの制御」を参照してください。

タグの使用方法のヒントについては、AWS の回答ブログのAWS タグ付け戦略記事を参照してください。

以下のセクションでは、Amazon QLDB のタグに関する詳細が説明されています。

トピック

Amazon QLDB でサポートされるリソース

Amazon QLDB の以下のリソースがタグ付けをサポートしています。

- 台帳
- テーブル
- ジャーナルストリーム

タグを追加および管理する方法については、「タグの管理」を参照してください。

タグの命名規則と使用規則

Amazon QLDB リソースでのタグの使用には、以下の基本的な命名規則と使用規則が適用されます。

- 各リソースには、最大 50 個のタグを設定できます。
- タグキーは、リソースごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- タグキーの最大長は UTF-8 で 128 Unicode 文字です。
- タグ値の最大長は UTF-8 で 256 Unicode 文字です。
- 使用できる文字は、UTF-8 対応の文字、数字、スペースと、文字 (. : + = @ _ / -) (ハイフン) です。
- タグのキーと値は大文字と小文字が区別されます。ベストプラクティスとして、タグを大文字にするための戦略を決定し、その戦略をすべてのリソースタイプにわたって一貫して実装します。たとえば、`Costcenter`、`costcenter`、`CostCenter` のいずれを使用するかを決定し、すべてのタグに同じ規則を使用します。大文字と小文字の扱いについて、同様のタグに整合性のない規則を使用することは避けてください。
- `aws:` プレフィックスは AWS 用に限定されています。タグに `aws:` というプレフィックスが付いたタグキーがある場合、タグのキーまたは値を編集、削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限には計算されません。

タグの管理

タグは、リソースの `Key` および `Value` プロパティで構成されています。このようなプロパティの値を追加、編集、削除するには、Amazon QLDB コンソール、AWS CLI、または QLDB API を使用します。AWS Resource Groups タグエディタを使用してタグを管理することもできます。

タグの使用については、以下の API オペレーションを参照してください。

- ListTagsForResource (Amazon QLDB API リファレンス)
- TagResource (Amazon QLDB API リファレンス)
- UntagResource (Amazon QLDB API リファレンス)

QLDB のタグ付けパネルを使用するには (コンソール)

1. AWS Management Console にサインインして、Amazon QLDB コンソール (https://console.aws.amazon.com/qldb) を開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. [Ledgers (台帳)] リストで、タグを管理する台帳の名前を選択します。
4. 台帳の詳細ページの [Tags (タグ)] カードで、[Manage tags (タグの管理)] を選択します。
5. [Manage tags (タグの管理)] ページで、台帳に対して、必要に応じてタグを追加、編集、または削除できます。タグのキーと値が希望どおりになったら、[Save (保存)] を選択します。

作成時のリソースのタグ付け

タグ付けをサポートする QLDB リソースの場合、リソースの作成中にタグを定義するには、AWS Management Console、AWS CLI、または QLDB API を使用します。作成時にリソースにタグ付けすることで、リソース作成後にカスタムタグ付けスクリプトを実行する必要がなくなります。

リソースにタグ付けすると、それらのタグに基づいてリソースへのアクセスを制御できます。たとえば、特定のタグを持つテーブルリソースにのみフルアクセス権を付与できます。JSON ポリシーの例については、「テーブルタグに基づくすべてのアクションへのフルアクセス」を参照してください。

> **Note**
>
> テーブルリソースとストリーミングリソースは、ルート台帳リソースのタグを継承しません。

テーブルタグは、`CREATE TABLE` PartiQL ステートメントで指定することでも定義できます。詳細については、「テーブルのタグ付け」を参照してください。

Amazon QLDB のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ — AWS は、 AWS のサービス で実行されるインフラストラクチャを保護する責任を担います AWS クラウド。また、 は、お客様が安全に使用できるサービス AWS も提供します。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon QLDB に適用されるコンプライアンスプログラムについては、「コンプライアンスプログラムによる対象範囲内のAWS サービス」を参照してください。
- クラウドのセキュリティ — お客様の責任は AWS のサービス 、使用する によって決まります。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、QLDB を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために QLDB を設定する方法を示します。また、QLDB リソースのモニタリングや保護 AWS のサービス に役立つ他の の使用方法についても説明します。

トピック

Amazon QLDB のデータ保護

責任 AWS 共有モデル、Amazon QLDB でのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された記事「AWS 責任共有モデルおよび GDPR」を参照してください。

データ保護の目的で、 認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これは、コンソール、API、または AWS のサービス SDK を使用して QLDB AWS CLIまたは他の を使用する場合も同様です。 AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

> **Note**
>
> 機密情報におけるタグまたは自由形式のフィールドの回避に関するこのガイダンスでは、台帳内に保存されるデータではなく、QLDB 台帳リソースのメタデータについて言及しています。QLDB 台帳リソース内に保存されたデータは、請求ログや診断ログには使用されません。

トピック

Amazon QLDB での保管時の暗号化

Amazon QLDB に保存されているすべてのデータは、保管時にデフォルトで完全に暗号化されます。QLDB 保管時の暗号化は、 AWS Key Management Service () の暗号化キーを使用して保管中のすべての台帳データを暗号化することで、セキュリティを強化しますAWS KMS。この機能は、機密データの保護における負担と複雑な作業を減らすのに役立ちます。保管時に暗号化することで、セキュリティを重視した台帳アプリケーションを構築して、暗号化のコンプライアンスと規制の厳格な要件を満たすことができます。

保管時の暗号化は、QLDB 台帳の保護に使用される暗号化キー AWS KMS を管理するために と統合されます。の詳細については AWS KMS、「 AWS Key Management Service デベロッパーガイド」の「 のAWS Key Management Service 概念」を参照してください。

QLDB では、台帳リソース AWS KMS key ごとに のタイプを指定できます。新しい台帳を作成するとき、または既存の台帳を更新するときに、次のタイプの KMS キーのいずれかを選択して台帳データを保護できます。

- AWS 所有のキー– デフォルトの暗号化タイプ。キーは QLDB により所有されます (追加料金なし)。
- カスタマーマネージドキー – キーは AWS アカウント に保存され、ユーザーによって作成、所有、管理されます。キーを完全に制御できます (AWS KMS 料金が適用されます) 。

> **Note**
>
> Amazon QLDB は、2021 年 7 月 22 AWS KMS keys 日にカスタマー管理のサポートを開始しました。起動前に作成された台帳は AWS 所有のキー デフォルトで によって保護されますが、現在、カスタマーマネージドキーを使用した保管時の暗号化の対象にはなりません。台帳の作成時間は QLDB コンソールで確認できます。

台帳にアクセスすると、QLDB はデータを透過的に復号化します。 AWS 所有のキー とカスタマーマネージドキーはいつでも切り替えることができます。暗号化されたデータの使用あるいは管理のためにコードやアプリケーションを変更する必要はありません。

新しい台帳を作成するときに暗号化キーを指定したり、、QLDB API AWS Management Console、または AWS Command Line Interface () を使用して既存の台帳の暗号化キーを変更したりできます AWS CLI。詳細については、「Amazon QLDB でカスタマーマネージドキーを使用する」を参照してください。

> **Note**
>
> デフォルトでは、Amazon QLDB は追加料金なしで AWS 所有のキー を使用して保管時の暗号化を自動的に有効にします。ただし、カスタマーマネージドキーの使用には AWS KMS 料金が適用されます。料金については、「AWS Key Management Service の料金」を参照してください。
>
> 保管時の QLDB 暗号化は、QLDB AWS リージョン が利用可能なすべての で使用できます。

トピック

- 保管時の暗号化: Amazon QLDB での仕組み
- Amazon QLDB でカスタマーマネージドキーを使用する

保管時の暗号化: Amazon QLDB での仕組み

QLDB の保管時の暗号化では、256 ビットの Advanced Encryption Standard (AES-256) を使用してデータを暗号化します。この機能は、基になるストレージへの不正アクセスからデータを保護するのに役立ちます。QLDB 台帳に保存されるすべてのデータは、保管時にデフォルトで暗号化されます。サーバー側の暗号化は透過的です。つまり、アプリケーションに対する変更は必要ありません。

保管時の暗号化は AWS Key Management Service （ AWS KMS) と統合され、QLDB 台帳の保護に使用される暗号化キーを管理します。新しい台帳を作成するとき、または既存の台帳を更新するときに、次のタイプの AWS KMS キーのいずれかを選択できます。

- AWS 所有のキー– デフォルトの暗号化タイプ。キーは QLDB により所有されます (追加料金なし)。
- カスタマーマネージドキー – キーは AWS アカウント に保存され、ユーザーによって作成、所有、管理されます。キーを完全に制御できます (AWS KMS 料金が適用されます)。

トピック

AWS 所有のキー

AWS 所有のキー は に保存されません AWS アカウント。これらは、複数の で使用するために が AWS 所有および管理する KMS キーのコレクションの一部です AWS アカウント。 AWS のサービスは AWS 所有のキー を使用してデータを保護します。

AWS 所有のキーを作成または管理する必要はありません。ただし、 を表示または追跡したり AWS 所有のキー、その使用を監査したりすることはできません。の月額料金や使用料は請求されず AWS 所有のキー、アカウントのクォータにも AWS KMS カウントされません。

詳細については、「AWS Key Management Service デベロッパーガイド」の 「AWS 所有のキー」を参照してください。

カスタマーマネージドキー

カスタマーマネージドキーは、ユーザーが作成、所有、管理する の KMS キー AWS アカウント です。ユーザーは、この KMS キーに関する完全なコントロール権を持ちます。QLDB では、対称暗号化 KMS キーのみをサポートしています。

カスタマーマネージドキーを使用して次の機能を取得します。

- キーポリシー、IAM ポリシー、およびキーへのアクセスを制御するための許可を設定し管理する

- キーの有効化と無効化を行う
- キーの暗号化マテリアルをローテーションする
- キーのタグとエイリアスを作成する
- キーの削除をスケジュールする
- 独自のキーマテリアルをインポートする、または所有し管理しているカスタムキーストアを使用する
- AWS CloudTrail および Amazon CloudWatch Logs を使用して、QLDB が AWS KMS ユーザーに代わって に送信するリクエストを追跡する

詳細については、「AWS Key Management Service デベロッパーガイド」の「カスタマーマネージドキー」を参照してください。

カスタマーマネージドキーには API コールごとに料金が発生し、これらの KMS キーには AWS KMS クォータが適用されます。詳細については、「AWS KMS resource or request quotas」(KMS リソースクォータまたはリクエストクォータ) を参照してください。

カスタマーマネージドキーを台帳の KMS キーとして指定すると、ジャーナルストレージとインデックス付きストレージの両方のすべての台帳データが、同じカスタマーマネージドキーで保護されます。

アクセスできないカスタマーマネージドキー

カスタマーマネージドキーを無効にしたり、キーの削除をスケジュールしたり、キーに対する許可を取り消したりすると、台帳の暗号化のステータスは `KMS_KEY_INACCESSIBLE` になります。この状態のとき、台帳には障害が発生し、台帳は読み取りリクエストまたは書き込みリクエストを受け付けないようになります。アクセスできないキーを使用すると、すべてのユーザーと QLDB サービスがデータを暗号化または復号化することができなくなり、台帳で読み取り操作および書き込み操作を実行できなくなります。台帳に対するアクセスを維持し、データ損失を防止するには、QLDB が KMS キーにアクセスできる必要があります。

> **Important**
>
> 障害のある台帳は、キーの許可を復元した後、または無効化されたキーを再度有効にした後に、自動的にアクティブ状態に戻ります。
> ただし、カスタマーマネージドキーの削除は元に戻せません。キーを削除すると、そのキーで保護されている台帳にアクセスできなくなり、データが完全に回復不可能になります。

台帳の暗号化ステータスを確認するには、 AWS Management Console または DescribeLedger API オペレーションを使用します。

Amazon QLDB で AWS KMSの許可を使用する方法

QLDB でカスタマーマネージドキーを使用するには許可が必要です。カスタマーマネージドキーで保護された台帳を作成すると、QLDB は にCreateGrantリクエストを送信してユーザーに代わって許可を作成します AWS KMS。の許可 AWS KMS は、QLDB に顧客 の KMS キーへのアクセスを許可するために使用されます AWS アカウント。詳細については、「AWS Key Management Service デベロッパーガイド」の「許可の使用」を参照してください。

QLDB では、以下の AWS KMS オペレーションでカスタマーマネージドキーを使用するのに許可が必要です。

- DescribeKey – 指定された対称暗号化 KMS キーが有効であることを確認します。
- GenerateDataKey – QLDB が台帳に保管中のデータを暗号化するために使用する一意の対称データキーを生成します。
- Decrypt — カスタマーマネージドキーで暗号化されたデータキーを復号化します。
- Encrypt — カスタマーマネージドキーを使用して、プレーンテキストを暗号文に暗号化します。

いつでも許可を取り消して、カスタマーマネージドキーに対するサービスからのアクセス権を削除できます。これを行うと、キーはアクセス不可能になり、QLDB はカスタマーマネージドキーによって保護されている台帳データへのアクセスを失います。この状態のとき、台帳には障害が発生し、キーの許可を復元するまで台帳は読み取りリクエストまたは書き込みリクエストを受け付けないようになります。

AWS KMSでの許可の復元

カスタマーマネージドキーに対する許可を復元し、QLDB で台帳へのアクセスを回復するには、台帳を更新して同じ KMS キーを指定します。手順については、「既存の台帳の AWS KMS key を更新する」を参照してください。

保管時の暗号化に関する考慮事項

QLDB で保管時の暗号化を使用する場合は、以下の点を考慮してください。

- 保管時のサーバー側の暗号化は、デフォルトですべての QLDB 台帳データで有効になり、無効にできません。台帳内のデータのサブセットのみを暗号化することはできません。

- 保管時の暗号化は、永続的ストレージメディアの静的 (保管時) のデータのみを暗号化します。転送中のデータあるいは使用中のデータでデータの安全性が考慮される場合には、次のように追加の対策を実行する必要がある場合があります。
  - 送信中のデータ: 送信中、QLDB 内のすべてのデータが暗号化されます。デフォルトでは、QLDB との通信において、Secure Sockets Layer (SSL)/Transport Layer Security (TLS) 暗号化を使用してネットワークトラフィックを保護する HTTPS プロトコルが使用されます。
  - 使用中のデータ: クライアント側の暗号化を使用することで、QLDB にデータを送信する前にデータを保護します。

台帳に対してカスタマーマネージドキーを実装する方法については、「Amazon QLDB でカスタマーマネージドキーを使用する」を参照してください。

Amazon QLDB でカスタマーマネージドキーを使用する

、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または QLDB API を使用して、Amazon QLDB の新しい台帳と既存の台帳 AWS KMS key の を指定できます。以下のトピックでは、QLDB でのカスタマーマネージドキーの使用状況を管理および監視する方法について説明します。

トピック

- 前提条件
- 新しい台帳に AWS KMS key を指定する
- 既存の台帳の AWS KMS key を更新する
- AWS KMS keysのモニタリング

前提条件

カスタマーマネージドキーで QLDB 台帳を保護する前に、まず AWS Key Management Service () でキーを作成する必要がありますAWS KMS。また、QLDB が AWS KMS key ユーザーに代わって許可を作成できるようにするキーポリシーを指定する必要があります。

カスタマーマネージドキーの作成

カスタマーマネージドキーを作成するには、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 KMS キーの作成」のステップに従います。QLDB は非対称キーをサポートしていません。

キーポリシーの設定

キーポリシーは、 でカスタマーマネージドキーへのアクセスを制御する主な方法です AWS KMS。すべてのカスタマーマネージドキーには、厳密に 1 つのキーポリシーが必要です。キーポリシードキュメントのステートメントでは、KMS キーの使用を許可するユーザーとその使用方法を決定します。詳細については、「 でのキーポリシーの使用 AWS KMS」を参照してください。

カスタマーマネージドキーを作成する際に、キーポリシーを指定することができます。既存のカスタマーマネージドキーのキーポリシーを変更するには、「キーポリシーの変更」を参照してください。

QLDB がカスタマーマネージドキーを使用できるようにするには、キーポリシーに次の AWS KMS アクションのアクセス許可を含める必要があります。

- kms:CreateGrant – カスタマーマネージドキーに許可を追加します。指定された KMS キーに対するアクセスコントロールを許可します。

  指定されたカスタマーマネージドキーで台帳を作成または更新すると、QLDB は、必要な権限オペレーションへのアクセスを可能にする権限を作成します。権限オペレーションには以下が含まれます。

  - GenerateDataKey
  - Decrypt
  - 暗号化
- kms:DescribeKey – カスタマーマネージドキーに関する詳細情報を返します。QLDB は、この情報を使用してキーを検証します。

キーポリシーの例

以下は、QLDB で使用できるキーポリシーの例です。このポリシーでは、アカウント 111122223333 から QLDB を使用して、リソース `arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab` で `DescribeKey` オペレーションと `CreateGrant` オペレーションを呼び出す権限があるプリンシパルを許可しています。

このポリシーを使用するには、以下の例の *`us-east-1`*、*`111122223333`*、および *`1234abcd-12ab-34cd-56ef-1234567890ab`* を自分の情報と置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid" : "Allow access to principals authorized to use Amazon QLDB",
            "Effect" : "Allow",
            "Principal" : {
                "AWS" : "*"
            },
            "Action" : [
                "kms:DescribeKey",
                "kms:CreateGrant"
            ],
            "Resource" : "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "Condition" : {
                "StringEquals" : {
                    "kms:ViaService" : "qldb.us-east-1.amazonaws.com",
                    "kms:CallerAccount" : "111122223333"
                }
            }
        }
    ]
}
```

新しい台帳に AWS KMS key を指定する

QLDB コンソールまたは AWS CLIを使用して新しい台帳を作成するときに KMS キーを指定するには、次のステップを実行します。

カスタマーマネージドキーは、ID、エイリアス、または Amazon リソースネーム (ARN) を使用して指定できます。詳細については、「 AWS Key Management Service デベロッパーガイド」の「キー識別子 (KeyId) 」を参照してください。

> **Note**
>
> クロスリージョンキーはサポートされていません。指定された KMS キーは台帳 AWS リージョン と同じ にある必要があります。

台帳の作成 (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/qldb で Amazon QLDB コンソールを開きます。

2. [Create Ledger (台帳の作成)] を選択します。
3. [Create Ledger (台帳の作成)] ページで、以下の操作を実行します。
   - 台帳情報 – 現在の AWS アカウント とリージョンのすべての台帳間で一意の台帳名を入力します。
   - [Permissions mode] (アクセス許可モード) – 台帳に割り当てるアクセス許可モードを選択します。
     - [Allow all] (すべて許可)
     - [Standard] (スタンダード) (推奨)
   - [Encrypt data at rest] (保管中のデータの暗号化) – 保管時の暗号化に使用する KMS キーのタイプを選択します。
     - AWS 所有の KMS キーを使用する – ユーザーに代わって が所有および管理する KMS キー AWS を使用します。これはデフォルトのオプションであり、追加のセットアップは必要ありません。
     - 別の AWS KMS キーを選択する – 作成、所有、管理するアカウントで対称暗号化 KMS キーを使用します。

       AWS KMS コンソールを使用して新しいキーを作成するには、 AWS KMS キーの作成 を選択します。詳細については、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 KMS キーを作成する」を参照してください。

       既存の KMS キーを使用するには、ドロップダウンリストからキーを選択するか、KMS キー ARN を指定します。
4. すべての設定が正しいことを確認したら、[Create ledger (台帳の作成)] を選択します。

   QLDB の台帳には、台帳のステータスが [Active] (アクティブ) になるとアクセスできるようになります。これには数分間かかる場合があります。

台帳の作成 (AWS CLI)

AWS CLI を使用して、デフォルトキー AWS 所有のキー またはカスタマーマネージドキーを使用して QLDB に台帳を作成します。

Example – デフォルトの AWS 所有のキーを使用して台帳を作成するには

```
aws qldb create-ledger --name my-example-ledger --permissions-mode STANDARD
```

Example – カスタマーマネージドキーを使用して台帳を作成するには

```
aws qldb create-ledger \
    --name my-example-ledger \
    --permissions-mode STANDARD \
    --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

既存の台帳の AWS KMS key を更新する

QLDB コンソールまたは を使用して AWS CLI 、既存の台帳の KMS キーをいつでも AWS 所有のキー またはカスタマーマネージドキーに更新することもできます。

> **Note**
>
> Amazon QLDB は、2021 年 7 月 22 AWS KMS keys 日にカスタマー管理のサポートを開始しました。起動前に作成された台帳は AWS 所有のキー デフォルトで によって保護されますが、現在、カスタマーマネージドキーを使用した保管時の暗号化の対象にはなりません。台帳の作成時間は QLDB コンソールで確認できます。

QLDB のキーの変更は非同期です。キーの変更が処理されている間も、パフォーマンスに影響を与えることなく、台帳にフルアクセスできます。キーの更新にかかる時間は、台帳のサイズによって異なります。

カスタマーマネージドキーは、ID、エイリアス、または Amazon リソースネーム (ARN) を使用して指定できます。詳細については、「 AWS Key Management Service デベロッパーガイド」の「キー識別子 (KeyId) 」を参照してください。

> **Note**
>
> クロスリージョンキーはサポートされていません。指定された KMS キーは台帳 AWS リージョン と同じ にある必要があります。

台帳の更新 (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/qldb で Amazon QLDB コンソールを開きます。

2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. 台帳のリストで、更新する台帳を選択し、[Edit ledger] (台帳を編集) を選択します。
4. [Edit ledger] (台帳を編集) ページで、保管時の暗号化に使用する KMS キーのタイプを選択します。

   - AWS 所有の KMS キーを使用する – ユーザーに代わって が所有および管理する KMS キー AWS を使用します。これはデフォルトのオプションであり、追加のセットアップは必要ありません。
   - 別の AWS KMS キーを選択する – 作成、所有、管理するアカウントで対称暗号化 KMS キーを使用します。

     AWS KMS コンソールを使用して新しいキーを作成するには、 AWS KMS キーの作成 を選択します。詳細については、「AWS Key Management Service デベロッパーガイド」の「対称暗号化 KMS キーを作成する」を参照してください。

     既存の KMS キーを使用するには、ドロップダウンリストからキーを選択するか、KMS キー ARN を指定します。
5. [Confirm changes] (変更の確認) を選択します。

台帳の更新 (AWS CLI)

AWS CLI を使用して、QLDB の既存の台帳をデフォルト AWS 所有のキー キーまたはカスタマーマネージドキーで更新します。

Example – デフォルトの AWS 所有のキーを使用して台帳を更新するには

```
aws qldb update-ledger --name my-example-ledger --kms-key AWS_OWNED_KMS_KEY
```

Example – カスタマーマネージドキーを使用して台帳を更新するには

```
aws qldb update-ledger \
    --name my-example-ledger \
    --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

AWS KMS keysのモニタリング

カスタマーマネージドキーを使用して Amazon QLDB 台帳を保護する場合は、 AWS CloudTrailまたは Amazon CloudWatch Logs を使用して、QLDB が AWS KMS ユーザーに代わって に送信するリク

エストを追跡できます。詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMS keysのモニタリング」を参照してください。

以下の例は、オペレーション CreateGrant、、GenerateDataKey、Decrypt、Encryptおよびの CloudTrail ログエントリですDescribeKey。

CreateGrant

台帳を保護するためにカスタマーマネージドキーを指定すると、QLDB は AWS KMS ユーザーに代わって にCreateGrantリクエストを送信し、KMS キーへのアクセスを許可します。また、台帳を削除すると、QLDB が RetireGrant オペレーションを使用して許可を削除します。

QLDB が作成する権限付与は台帳ごとに固有となります。CreateGrant リクエストのプリンシパルは、テーブルを作成したユーザーです。

CreateGrant 演算を記録するイベントは、次のようなサンプルイベントになります。このパラメータには、カスタマーマネージドキーの Amazon リソースネーム (ARN)、プリンシパルの被付与者と廃止プリンシパル (QLDB サービス)、およびこの権限付与の範囲内のオペレーションが含まれます。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:sample-user",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/sample-user",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-06-04T21:37:11Z"
            }
        },
        "invokedBy": "qldb.amazonaws.com"
```

```
    },
    "eventTime": "2021-06-04T21:40:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "granteePrincipal": "qldb.us-west-2.amazonaws.com",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt"
        ],
        "retiringPrincipal": "qldb.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "b3c83f999187ccc0979ef2ff86a1572237b6bba309c0ebce098c34761f86038a"
    },
    "requestID": "e99188d7-3b82-424e-b63e-e086d848ed60",
    "eventID": "88dc7ba5-4952-4d36-9ca8-9ab5d9598bab",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKey

台帳を保護するためにカスタマーマネージドキーを指定すると、QLDB は一意のデータキーを作成します。台帳のカスタマーマネージドキー AWS KMS を指定するGenerateDataKeyリクエストを に送信します。

GenerateDataKey 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは、QLDB サービスアカウントです。このパラメータには、カスタマーマネージドキーの ARN、32 バイトの長さを必要とするデータキー指定子、および内部キー階層ノードを識別する暗号化コンテキストが含まれます。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:01Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "numberOfBytes": 32,
        "encryptionContext": {
            "key-hierarchy-node-id": "LY4HWMnkeZWKYi6MlitVJC",
            "key-hierarchy-node-version": "1"
        }
    },
    "responseElements": null,
    "requestID": "786977c9-e77c-467a-bff5-9ad5124a4462",
    "eventID": "b3f082cb-3e75-454e-bf0a-64be13075436",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
```

```
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "26688de5-0b1c-43d3-bc4f-a18029b08446"
}
```

Decrypt

台帳にアクセスすると、QLDB は Decrypt オペレーションを呼び出して台帳の保存されたデータキーを復号化し、台帳内の暗号化されたデータにアクセスできるようにします。

Decrypt 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは、QLDB サービスアカウントです。このパラメータには、カスタマーマネージドキーの ARN、および内部キー階層ノードを識別する暗号化コンテキストが含まれます。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:56Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "key-hierarchy-node-id": "LY4HWMnkeZWKYi6MlitVJC",
            "key-hierarchy-node-version": "1"
        }
    },
    "responseElements": null,
    "requestID": "28f2dd18-3cc1-4fe2-82f7-5154f4933ebf",
    "eventID": "603ad5d4-4744-4505-9c21-bd4a6cbd4b20",
    "readOnly": true,
    "resources": [
```

```
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "7b6ce3e3-a764-42ec-8f90-5418c97ec411"
}
```

暗号化

QLDB は Encrypt オペレーションを呼び出し、カスタマーマネージドキーを使用してプレーンテキストを暗号文に暗号化します。

Encrypt 演算を記録するイベントは、次のようなサンプルイベントになります。ユーザーは、QLDB サービスアカウントです。このパラメータには、カスタマーマネージドキーの ARN、および台帳の一意の内部 ID を指定する暗号化コンテキストが含まれます。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:01Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Encrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionContext": {
            "LedgerId": "F6qRNziJLUXA4Vy2ZUv8YY"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
```

```
    "responseElements": null,
    "requestID": "b2daca7d-4606-4302-a2d7-5b3c8d30c64d",
    "eventID": "b8aace05-2e37-4fed-ae6f-a45a1c6098df",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "ce420ab0-288e-4b4f-aee8-541e18a28aa5"
}
```

DescribeKey

QLDB は `DescribeKey` オペレーションを呼び出して、指定した KMS キーが AWS アカウントおよびリージョンに存在するかどうかを判断します。

`DescribeKey` 演算を記録するイベントは、次のようなサンプルイベントになります。プリンシパルは、KMS キーを AWS アカウント 指定した のユーザーです。このパラメータには、カスタマーマネージドキーの ARN が含まれます。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:sample-user",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/sample-user",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
```

```
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-06-04T21:37:11Z"
            }
        },
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "a30586af-c783-4d25-8fda-33152c816c36",
    "eventID": "7a9caf07-2b27-44ab-afe4-b259533ebb88",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

Amazon QLDB での転送時の暗号化

Amazon QLDB は、Secure Sockets Layer (SSL)/Transport Layer Security (TLS) を使ってネットワークトラフィックを保護する HTTPS プロトコルを使用する、セキュアな接続のみを受け入れます。転送時の暗号化では、QLDB との間で送受信するときにデータを暗号化することによって、データ保護

のレイヤーを追加します。組織のポリシー、業界や政府の規制、またはコンプライアンス要件によって、ネットワークを介したデータ転送時にアプリケーションのデータセキュリティを高めるために転送時の暗号化の使用が求められることがあります。

QLDB では、選択されたリージョンで FIPS エンドポイントも提供します。標準の AWS エンドポイントとは異なり、FIPS エンドポイントでは連邦情報処理標準 (FIPS) 140-2 に準拠した TLS ソフトウェアライブラリを使用しています。このエンドポイントは、米国政府とやり取りをする企業で必要とされる場合があります。詳細については、「AWS 全般のリファレンス」の「FIPS エンドポイント」を参照してください。QLDB で使用可能なリージョンとエンドポイントの完全なリストについては、「Amazon QLDB エンドポイントとクォータ」を参照してください。

Amazon QLDB のための Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM の管理者は、誰を認証 (サインイン) し、誰に QLDB リソースの使用を許可する (アクセス許可を持たせる) かを制御できます。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、QLDB で行う作業によって異なります。

サービスユーザー – ジョブを実行するために QLDB サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。作業を実行するためにさらに多くの QLDB 機能を使用するとき、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。QLDB の機能にアクセスできない場合は、「Amazon QLDB アイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 - 社内の QLDB リソースを担当している場合は、通常、QLDB へのフルアクセスがあります。サービスのユーザーがどの QLDB 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。お客様の会社で QLDB で IAM を利用する方法の詳細については、「Amazon QLDB で IAM が機能する仕組み」を参照してください。

IAM 管理者 – IAM 管理者には、QLDB へのアクセスを管理するポリシーの作成方法の詳細を理解することが推奨されます。IAM で使用できる QLDB アイデンティティベースのポリシーの例を表示するには、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッドアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧めします。詳細については、『AWS IAM Identity Center ユーザーガイド』の「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWSにおける多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、『IAM ユーザーガイド』の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリ、または ID ソースを通じて提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID がにアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用できるようにすることもできます。IAM Identity Center の詳細については、『AWS IAM Identity Center ユーザーガイド』の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めしま

す。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、『IAM ユーザーガイド』の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッドアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」 を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。

- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスにリンクされたロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) — IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに権限を委任するロールの作成」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールの権限を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロ

ファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、『IAM ユーザーガイド』の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、ID またはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS としてに保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれていま

す。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、『IAM ユーザーガイド』の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、『Amazon Simple Storage Service デベロッパーガイド』の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal`

フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティの許可の境界」を参照してください。

- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、『AWS Organizations ユーザーガイド』の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

Amazon QLDB で IAM が機能する仕組み

IAM を使用して QLDB へのアクセスを管理する前に、QLDB で利用できる IAM の機能について学びます。

Amazon QLDB で使用できる IAM の機能

IAM 機能	QLDB のサポート
アイデンティティベースのポリシー	Yes

IAM 機能	QLDB のサポート
リソースベースのポリシー	No
ポリシーアクション	Yes
ポリシーリソース	Yes
ポリシー条件キー	Yes
ACL	No
ABAC (ポリシー内のタグ)	はい
一時的な認証情報	Yes
プリンシパル権限	いいえ
サービスロール	あり
サービスリンクロール	いいえ

QLDB およびその他の がほとんどの IAM 機能と AWS のサービス 連携する方法の概要を把握するには、IAM ユーザーガイドのAWS のサービス 「IAM と連携する 」を参照してください。

QLDB のアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されている

ユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

QLDB のアイデンティティベースのポリシーの例

QLDB のアイデンティティベースのポリシーの例を表示するには、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

QLDB 内のリソースベースのポリシー

リソースベースのポリシーのサポート	No

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

QLDB のポリシーアクション

ポリシーアクションに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

QLDB アクションのリストを確認するには、「サービス認証リファレンス」の「Amazon QLDB で定義されるアクション」を参照してください。

QLDB のポリシーアクションは、アクションの前に以下のプレフィックスを使用します。

```
qldb
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "qldb:action1",
      "qldb:action2"
]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "qldb:Describe*"
```

台帳で PartiQL ステートメントを実行して QLDB トランザクションデータ API (QLDB セッション) と対話するには、次のように SendCommand アクションにアクセス許可を付与する必要があります。

```
"Action": "qldb:SendCommand"
```

STANDARD アクセス許可モードの台帳の場合は、「PartiQL アクセス許可のリファレンス」を参照して、各 PartiQL コマンドに必要な追加のアクセス許可を確認してください。

QLDB のアイデンティティベースのポリシーの例を表示するには、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

QLDB のポリシーリソース

ポリシーリソースに対するサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

`Resource` JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、`Resource` または `NotResource`要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

QLDB リソースのタイプとその ARN のリストを確認するには、「サービス認証リファレンス」の「Amazon QLDB で定義されるリソース」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「Amazon QLDB で定義されるアクション」を参照してください。

QLDB では、プライマリリソースは台帳です。QLDB では、追加のリソースタイプ (テーブルおよびストリーム) もサポートしています。ただし、既存の台帳のコンテキストでのみ、テーブルやストリームを作成できます。

QLDB テーブルは、台帳のジャーナルからの順序付けられていないドキュメントリビジョンのコレクションのマテリアライズドビューです。STANDARD アクセス許可モードの台帳では、このテーブルリソースで PartiQL ステートメントを実行するために、アクセス許可を付与する IAM ポリシーを作成する必要があります。テーブルリソースに対するアクセス許可を使用すると、テーブルの現在の状態にアクセスするステートメントを実行できます。組み込みの `history()` 関数を使用してテーブル

のリビジョン履歴のクエリを実行することもできます。詳細については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

> **Note**
>
> CREATE TABLE ステートメントは、一意の ID と指定されたテーブル名を持つテーブルを作成します。指定するテーブル名は、すべてのアクティブなテーブルの中で一意である必要があります。ただし、QLDB ではテーブルを非アクティブ化できるため、同じテーブル名を共有する非アクティブなテーブルが複数存在する可能性があります。したがって、テーブルリソース ARN は、ユーザー定義のテーブル名ではなく、システムによって割り当てられた一意の ID を参照します。

各台帳では、システム定義のカタログリソースも提供され、このリソースでクエリを実行して台帳内のすべてのテーブルとインデックスを一覧表示できます。QLDB データオブジェクトモデルの詳細については、「Amazon QLDB の重要な概念と用語」を参照してください。

このリソースには、次の表に示すように、一意の ARN が関連付けられています。

リソースタイプ	ARN
ledger	`arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}`
table	`arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}/table/${TableId}`
catalog	`arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}/information_schema/user_tables`
stream	`arn:${Partition}:qldb:${Region}:${Account}:stream/${LedgerName}/${StreamId}`

例えば、ステートメントで myExampleLedger リソースを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
```

複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": [
      "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger1",
      "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger2"
]
```

QLDB のアイデンティティベースのポリシーの例を表示するには、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

QLDB のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、 AWS は AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

QLDB の条件キーのリストを確認するには、「サービス認証リファレンス」の「Amazon QLDB の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon QLDB で定義されるアクション」を参照してください。

PartiQLDropIndex アクションおよび PartiQLDropTable アクションは、qldb:Purge 条件キーをサポートします。この条件キーは PartiQL DROP ステートメントで指定された purge の値でアクセスをフィルタします。ただし、QLDB で現在サポートされているのは purge = true for DROP INDEX ステートメントと purge = false for DROP TABLE ステートメントのみです。他の QLDB アクションでは、いくつかのグローバルコンディションキーがサポートされています。

QLDB のアイデンティティベースのポリシーの例を表示するには、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

QLDB のアクセスコントロールリスト (ACL)

ACL のサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

QLDB での属性ベースのアクセスコントロール (ABAC)

ABAC のサポート (ポリシー内のタグ)	はい

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合に操作を許可するように ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値ははいです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、『IAM ユーザーガイド』の「ABAC とは?」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセス制御 (ABAC) を使用する」を参照してください。

QLDB リソースのタグ付けの詳細については、「Amazon QLDB リソースのタグ付け」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグに基づく QLDB 台帳の更新」を参照してください。

QLDB での一時的な認証情報の使用

一時的な認証情報のサポート	はい

一部の は、一時的な認証情報を使用してサインインすると機能 AWS のサービス しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドのAWS のサービス「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ロールへの切り替え (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して にアクセスします AWS。 AWS 長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

QLDB のクロスサービスプリンシパル許可

転送アクセスセッション (FAS) をサポート	いいえ

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS の

サービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

QLDB のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに権限を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールの許可を変更すると、QLDB の機能が破損する可能性があります。QLDB が指示する場合以外は、サービスロールを編集しないでください。

次のセクションで説明するように、QLDB は `ExportJournalToS3` および `StreamJournalToKinesis` API オペレーションのサービスロールをサポートしています。

QLDB で IAM ロールを選択

QLDB でジャーナルブロックをエクスポートまたはストリーミングする場合、ユーザーに代わって特定の宛先にオブジェクトを書き込むことを QLDB に許可するロールを選択する必要があります。以前に作成したサービスロールがある場合、QLDB により、選択できるロールのリストが表示されます。エクスポート用に指定した Simple Storage Service (Amazon S3) バケットに書き込むか、ストリーム用に指定した Amazon Kinesis Data Streams リソースに書き込むためのアクセスを許可するロールを選択することが重要です。詳細については、QLDB のジャーナルエクスポート権限または QLDB のストリーミング許可を参照してください。

> **Note**
>
> ジャーナルエクスポートまたはストリームをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで `iam:PassRole` アクションを実行するためのアクセス許可が

必要です。これは、QLDB 台帳リソースでの `qldb:ExportJournalToS3` の実行、または QLDB ストリームサブリソースでの `qldb:StreamJournalToKinesis` の実行に対するアクセス許可に追加されます。

QLDB のサービスリンクロール

サービスにリンクされたロールのサポート	いいえ

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの権限を表示できますが、編集することはできません。

サービスリンクロールの作成または管理の詳細については、「IAM と提携するAWS のサービス」を参照してください。表の中から、[Service-linked role (サービスリンクロール)] 列に `Yes` と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] リンクを選択します。

Amazon QLDB の標準アクセス許可モードの開始方法

このセクションを参照して、Amazon QLDB の標準アクセス許可モードを開始します。このセクションでは、PartiQL アクションについて AWS Identity and Access Management (IAM) でアイデンティティベースのポリシーを作成する際に役立つリファレンステーブルと、QLDB のテーブルリソースを示します。また、IAM でアクセス許可ポリシーを作成するための step-by-step チュートリアルと、テーブル ARN を検索し、QLDB でテーブルタグを作成する手順も含まれています。

トピック

- STANDARD アクセス許可モード
- PartiQL アクセス許可のリファレンス
- テーブル ID と ARN の検索
- テーブルのタグ付け
- クイックスタートチュートリアル: アクセス許可ポリシーの作成

STANDARD アクセス許可モード

QLDB では、台帳リソースの STANDARD アクセス許可モードがサポートされるようになりました。標準アクセス許可モードでは、台帳、テーブル、および PartiQL コマンドをより詳細なレベルでアクセスコントロールすることができます。デフォルトでは、このモードは、台帳内の任意のテーブルで、任意の PartiQL コマンドを実行するすべてのリクエストを拒否します。

> **Note**
>
> 以前は、台帳で使用できる唯一のアクセス許可モードは ALLOW_ALL でした。この ALLOW_ALL モードでは、台帳の API レベルの詳細度によるアクセスコントロールが有効になり、QLDB 台帳では引き続きサポートされますが、推奨はされません。このモードでは、SendCommand API アクセス許可があるユーザーは、アクセス許可ポリシーで指定された台帳の任意のテーブルですべての PartiQL コマンドを実行することができます (したがって、PartiQL コマンドを「すべて許可」します)。
> 既存の台帳のアクセス許可モードを ALLOW_ALL から STANDARD に変更できます。詳細については、「標準アクセス許可モードへの移行」を参照してください。

標準モードでコマンドを許可するには、特定のテーブルリソースと PartiQL アクションに対するアクセス許可ポリシーを IAM で作成する必要があります。これは、台帳に対する SendCommand API アクセス許可に追加されます。このモードのポリシーを容易にするために、QLDB では、PartiQL コマンドに対する IAM アクションのセット、および QLDB テーブルの Amazon リソースネーム (ARN) が導入されました。QLDB データオブジェクトモデルの詳細については、「Amazon QLDB の重要な概念と用語」を参照してください。

PartiQL アクセス許可のリファレンス

次の表に、各 QLDB PartiQL コマンド、コマンドを実行するために に許可を付与する必要がある対応する IAM アクション、および許可を付与できる AWS リソースを示します。ポリシーの Action フィールドでアクションを指定し、ポリシーの Resource フィールドでリソースの値を指定します。

> **Important**
>
> - この PartiQL コマンドにアクセス許可を付与する IAM ポリシーは、STANDARD アクセス許可モードが台帳に割り当てられている場合にのみ、台帳に適用されます。このようなポリシーは、ALLOW_ALL アクセス許可モードの台帳には適用できません。

台帳を作成または更新するときにアクセス許可モードを指定する方法については、「Amazon QLDB 台帳の基本的なオペレーション」、または「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」を参照してください。

- 台帳で任意の PartiQL コマンドを実行するには、台帳リソースに対する SendCommand API アクションにもアクセス許可を付与する必要があります。これは、次の表に示す PartiQL アクションおよびテーブルリソースに追加されています。詳細については、「データトランザクションの実行」を参照してください。

Amazon QLDB PartiQL コマンドと必要なアクセス許可

Command	必要なアクセス許可 (IAM アクション)	リソース	依存アクション
CREATE TABLE	qldb:PartiQLCreateTable	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*	qldb:TagResource (作成時のタグ付け用)
DROP TABLE	qldb:PartiQLDropTable	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
テーブルの削除の取り消し	qldb:PartiQLUndropTable	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
CREATE INDEX	qldb:PartiQLCreateIndex	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	

Command	必要なアクセス許可 (IAM アクション)	リソース	依存アクション
DROP INDEX	qldb:PartiQLDropIndex	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
DELETE FROM-REMOVE (ドキュメント全体)	qldb:PartiQLDelete	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	qldb:PartiQLSelect
INSERT	qldb:PartiQLInsert	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
UPDATE FROM (INSERT、EMOVE、または SET)	qldb:PartiQLUpdate	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	qldb:PartiQLSelect
REDACT_REVISION (ストアドプロシージャ)	qldb:PartiQLRedact	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	

Command	必要なアクセス許可 (IAM アクション)	リソース	依存アクション
SELECT FROM table_name	qldb:PartiQLSelect	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
SELECT FROM information_schema.user_tables	qldb:PartiQLSelect	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /information_schema/user_tables	
SELECT FROM history(table_name)	qldb:PartiQLHistoryFunction	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	

これらの PartiQL コマンドにアクセス許可を付与する IAM ポリシードキュメントの例については、「クイックスタートチュートリアル: アクセス許可ポリシーの作成」または「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

テーブル ID と ARN の検索

テーブル ID を見つけるには、 を使用する AWS Management Console か、テーブル information_schema.user_tables をクエリします。コンソールでテーブルの詳細を表示する、または、このシステムカタログテーブルのクエリを実行するには、システムカタログリソースに対する SELECT アクセス許可が必要です。例えば、Vehicle テーブルのテーブル ID を検索するには、次のステートメントを実行できます。

```
SELECT * FROM information_schema.user_tables
```

```
WHERE name = 'Vehicle'
```

このクエリは、次の例のような形式で結果を返します。

```
{
    tableId: "Au1EiThbt8s0z9wM26REZN",
    name: "Vehicle",
    indexes: [
        { indexId: "Djg2nt0yIs2GY0T29Kud1z", expr: "[VIN]", status: "ONLINE" },
        { indexId: "4tPW3fUhaVhDinRgKRLhGU", expr: "[LicensePlateNumber]", status:
 "BUILDING" }
    ],
    status: "ACTIVE"
}
```

テーブルで PartiQL ステートメントを実行するためのアクセス許可を付与するには、次の ARN 形式でテーブルリソースを指定します。

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/table/${table-id}
```

以下に、テーブル ID Au1EiThbt8s0z9wM26REZN のテーブル ARN の例を示します。

```
arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/Au1EiThbt8s0z9wM26REZN
```

コンソールを使用する場合

QLDB コンソールを使用してテーブル ARN を検索することもできます。

テーブルの ARN を検索するには (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/qldb で Amazon QLDB コンソールを開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. [Ledgers] (台帳) のリストで、検索するテーブル ARN の台帳名を選択します。
4. 台帳の詳細ページの [Tables] (テーブル) タブで、検索する ARN のテーブル名を見つけます。ARN をコピーするには、横のコピーアイコン () を選択します。

テーブルのタグ付け

テーブルリソースにタグ付けすることができます。既存のテーブルのタグを管理するには、 AWS Management Console または API オペレーション `TagResource`、`UntagResource`、および を使用しますListTagsForResource。詳細については、「Amazon QLDB リソースのタグ付け」を参照してください。

> **Note**
>
> テーブルリソースは、ルート台帳リソースのタグを継承しません。
> 作成時のテーブルのタグ付けは、現在 STANDARD 許可モードの台帳のみでサポートされています。

テーブルの作成中に、QLDB コンソールを使用するか、`CREATE TABLE` PartiQL ステートメントで指定して、テーブルタグを定義することもできます。次の例では、`Vehicle` という名前のテーブルをタグ `environment=production` を付けて作成します。

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

作成時にテーブルにタグを付けるには、`qldb:PartiQLCreateTable` および `qldb:TagResource` アクションの両方にアクセスする必要があります。

作成時にリソースにタグ付けすることで、リソース作成後にカスタムタグ付けスクリプトを実行する必要がなくなります。テーブルにタグを付けると、それらのタグに基づいてテーブルへのアクセスを制御できます。例えば、特定のタグを持つテーブルにのみフルアクセスを付与できます。JSON ポリシーの例については、「テーブルタグに基づくすべてのアクションへのフルアクセス」を参照してください。

コンソールを使用する場合

QLDB コンソールを使用して、テーブルの作成中にテーブルタグを定義することもできます。

作成時にテーブルにタグを付けるには (コンソール)

1. にサインインし AWS Management Console、https://console.aws.amazon.com/qldb で Amazon QLDB コンソールを開きます。
2. ナビゲーションペイン内で [Ledgers (台帳)] を選択します。
3. [Ledgers] (台帳) のリストで、テーブルを作成する台帳の名前を選択します。

4. 台帳の詳細ページの [Tables] (テーブル) タブで、[Create table] (テーブルを作成) を選択します。
5. [Create table] (テーブルを作成) ページで、次の操作を行います。
   - [Table name] (テーブル名) – テーブル名を入力します。
   - [Tags] (タグ) – タグをキーと値の組み合わせとしてアタッチすることで、メタデータをテーブルに追加します。テーブルにタグを追加すると、テーブルの整理と識別がしやすくなります。

     [Add tag (タグの追加)] を選択し、必要に応じて、任意のキーと値の組み合わせを入力します。
6. すべての設定が正しいことを確認したら、[テーブルを作成] を選択します。

クイックスタートチュートリアル: アクセス許可ポリシーの作成

このチュートリアルでは、STANDARD アクセス許可モードの Amazon QLDB 台帳に対して IAM でアクセス許可ポリシーを作成する手順について説明します。その後、ユーザー、グループ、またはロールに権限を割り当てることができます。

PartiQL コマンドおよびテーブルリソースにアクセス許可を付与する IAM ポリシードキュメントの例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

トピック

- 前提条件
- 読み取り専用ポリシーの作成
- フルアクセスポリシーの作成
- 特定のテーブルの読み取り専用ポリシーの作成
- 権限の割り当て

前提条件

作業を始める前に、次の操作を実行してください。

1. まだ設定していない場合はAmazon QLDB へのアクセス、 AWS 「」の設定手順に従ってください。これらのステップには、 へのサインアップ AWS と管理ユーザーの作成が含まれます。
2. 新しい台帳を作成し、台帳の STANDARD アクセス許可モードを選択します。その方法については、「コンソールの開始方法」の「ステップ 1: 新しい台帳を作成する」、または「Amazon QLDB 台帳の基本的なオペレーション」を参照してください。

読み取り専用ポリシーの作成

JSON ポリシーエディタを使用して、標準アクセス許可モードの台帳のすべてのテーブルに対する読み取り専用ポリシーを作成するには、次の手順を実行します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーション列で、[ポリシー] を選択します。

   [Policies] (ポリシー) を初めて選択する場合は、[Welcome to Managed Policies] (マネージドポリシーにようこそ) ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [JSON] タブを選択します。
5. 以下の JSON ポリシードキュメントをコピーして貼り付けます。このポリシー例では、台帳のすべてのテーブルに読み取り専用アクセス権を付与しています。

   このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
      ]
    }
```

```
  ]
}
```

6. [ポリシーの確認] を選択します。

> **Note**
>
> いつでも [Visual editor] (ビジュアルエディタ) タブと [JSON] タブを切り替えることができます。ただし、[Visual editor] (ビジュアルエディタ) タブで [Review policy] (ポリシーの確認) を変更または選択した場合、IAM はポリシーを再構成してビジュアルエディタに合わせて最適化することがあります。詳細については、IAM ユーザーガイドの「ポリシーの再構成」を参照してください。

7. [ポリシーの確認] ページに作成するポリシーの [名前] と [説明] を入力します。ポリシーの [Summary] (概要) を参照して、ポリシーによって付与された許可を確認します。次に、[Create policy] (ポリシーの作成) を選択して作業を保存します。

フルアクセスポリシーの作成

標準アクセス許可モードの QLDB 台帳内のすべてのテーブルに対するフルアクセスポリシーを作成するには、次の手順を実行します。

- 次のポリシードキュメントを使用して、前のステップを繰り返します。このポリシー例では、ワイルドカード (*) を使用して、すべての PartiQL アクションと台帳のすべてのリソースがカバーされるようにすることで、台帳のすべてのテーブルに対するすべての PartiQL コマンドにアクセス権を付与しています。

> **Warning**
>
> これは、ワイルドカード文字 (*) を使用して、すべての PartiQL アクションを許可する例です。これには、QLDB 台帳のすべてのテーブルに対する管理オペレーションと読み取り/書き込みオペレーションが含まれます。代わりに、付与する各アクションと、そのユーザー、ロール、またはグループが必要とするアクションのみを明示的に指定することをお勧めします。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLFullPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQL*"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
      ]
    }
  ]
}
```

特定のテーブルの読み取り専用ポリシーの作成

標準アクセス許可モードの QLDB 台帳内の特定のテーブルに対する読み取り専用アクセスポリシーを作成するには、次の手順を実行します。

1. を使用する AWS Management Console か、システムカタログテーブル をクエリして、テーブルの ARN を検索しますinformation_schema.user_tables。手順については、「テーブル ID と ARN の検索」を参照してください。
2. テーブル ARN を使用して、テーブルへの読み取り専用アクセスを許可するポリシーを作成します。これを行うには、次のポリシードキュメントを使用して、前のステップを繰り返します。

   このポリシー例では、指定されたテーブルにのみ、読み取り専用アクセス権が付与されています。この例では、テーブル ID は Au1EiThbt8s0z9wM26REZN です。このポリシーを使用するには、例の *us-*

east-1、*123456789012myExampleLedger*、、Au*Au1EiThbt8s0z9wM26REZN* をユーザー自身の情報に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
table/Au1EiThbt8s0z9wM26REZN"
      ]
    }
  ]
}
```

権限の割り当て

QLDB アクセス権限ポリシーを作成したら、次のように権限を割り当てます。

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

Amazon QLDB のアイデンティティベースのポリシー例

デフォルトでは、IAM ユーザーおよびロールには、QLDB リソースを作成または変更するアクセス許可はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

QLDB が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認証リファレンス」の「Amazon QLDB のアクション、リソース、および条件キー」を参照してください。

目次

    - 読み取り専用アクセス
    - 特定のテーブルへの読み取り専用アクセス
    - テーブル作成のアクセス許可
    - リクエストタグに基づいたテーブル作成のアクセス許可
- Simple Storage Service (Amazon S3) バケットへのジャーナルのエクスポート
- Kinesis Data Streams へのジャーナルのストリーミング
- タグに基づく QLDB 台帳の更新

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰かが QLDB リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細については、『IAM ユーザーガイド』の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、 などの特定の を介してサービスアクションが使用される場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、『IAM ユーザーガイド』の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサ

ポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。

- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、『IAM ユーザーガイド』の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

QLDB コンソールの使用

Amazon QLDB コンソールにアクセスするには、許可の最小限のセットが必要です。これらのアクセス許可により、 の QLDB リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが QLDB コンソールとそのすべての機能にフルアクセスできるようにするには、エンティティに次の AWS 管理ポリシーをアタッチします。詳細については、「AWS Amazon QLDB の マネージドポリシー」、および「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

```
AmazonQLDBConsoleFullAccess
```

クエリ履歴のアクセス許可

QLDB のアクセス許可に加えて、一部のコンソール機能では、Database Query Metadata Service (サービスプレフィックス: dbqms) に対するアクセス許可が必要です。これは、QLDB やその他の AWS のサービスなどのコンソールクエリエディタで最近使用したクエリや保存したクエリを管理する内部専用のサービスです。DBQMS API アクションの全リストについては、「サービス認証リファレンス」の「データベースクエリメタデータサービス」を参照してください。

クエリ履歴のアクセス許可を許可するには、 AWS 管理ポリシー AmazonQLDBConsoleFullAccess を使用できます。このポリシーでは、ワイルドカード (dbqms:*) を使用して、すべてのリソースに対するすべての DBQMS アクションを許可します。

または、カスタム IAM ポリシーを作成して、次の DBQMS アクションを含めることもできます。QLDB コンソールの PartiQL クエリエディタには、クエリ履歴機能にこれらのアクションを使用する権限が必要です。

```
dbqms:CreateFavoriteQuery
dbqms:CreateQueryHistory
dbqms:DeleteFavoriteQueries
dbqms:DeleteQueryHistory
dbqms:DescribeFavoriteQueries
dbqms:DescribeQueryHistory
dbqms:UpdateFavoriteQuery
```

コンソールへのクエリ履歴権限を除いたフルアクセス許可

クエリ履歴権限なしで QLDB コンソールへのフルアクセスを許可するには、すべての DBQMS アクションを除外するカスタム IAM ポリシーを作成できます。例えば、次のポリシードキュメントでは、サービスプレフィックス で始まるアクションを除いて、 AWS 管理ポリシー AmazonQLDBConsoleFullAccess によって付与されるのと同じアクセス許可を許可しますdbqms。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "qldb:CreateLedger",
        "qldb:UpdateLedger",
        "qldb:UpdateLedgerPermissionsMode",
        "qldb:DeleteLedger",
        "qldb:ListLedgers",
        "qldb:DescribeLedger",
        "qldb:ExportJournalToS3",
        "qldb:ListJournalS3Exports",
        "qldb:ListJournalS3ExportsForLedger",
        "qldb:DescribeJournalS3Export",
        "qldb:CancelJournalKinesisStream",
        "qldb:DescribeJournalKinesisStream",
        "qldb:ListJournalKinesisStreamsForLedger",
        "qldb:StreamJournalToKinesis",
        "qldb:GetBlock",
        "qldb:GetDigest",
        "qldb:GetRevision",
        "qldb:TagResource",
```

```
        "qldb:UntagResource",
        "qldb:ListTagsForResource",
        "qldb:SendCommand",
        "qldb:ExecuteStatement",
        "qldb:ShowCatalog",
        "qldb:InsertSampleData",
        "qldb:PartiQLCreateIndex",
        "qldb:PartiQLDropIndex",
        "qldb:PartiQLCreateTable",
        "qldb:PartiQLDropTable",
        "qldb:PartiQLUndropTable",
        "qldb:PartiQLDelete",
        "qldb:PartiQLInsert",
        "qldb:PartiQLUpdate",
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "kinesis:ListStreams",
        "kinesis:DescribeStream"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "qldb.amazonaws.com"
        }
      }
    }
  ]
}
```

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

データトランザクションの実行

台帳で PartiQL ステートメントを実行して QLDB トランザクションデータ API (QLDB セッション) と対話するには、SendCommand API アクションにアクセス許可を付与する必要があります。次の JSON ドキュメントは、myExampleLedger という台帳に対してのみ、SendCommand API アクションのアクセス許可を付与するポリシーの例です。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        }
    ]
}
```

myExampleLedger が ALLOW_ALL アクセス許可モードを使用する場合、このポリシーは、台帳の任意のテーブルに対してすべての PartiQL コマンドを実行するアクセス許可を付与します。

AWS 管理ポリシーを使用して、すべての QLDB リソースへのフルアクセスを許可することもできます。詳細については、「AWS Amazon QLDB の マネージドポリシー」を参照してください。

PartiQL アクションとテーブルリソースに対する標準アクセス許可

STANDARD アクセス許可モードの台帳の場合は、適切な PartiQL アクセス許可を付与する例として、次の IAM ポリシードキュメントを参照できます。各 PartiQL コマンドに必要なアクセス許可のリストについては、「PartiQL アクセス許可のリファレンス」を参照してください。

トピック

- すべてのアクションへのフルアクセス
- テーブルタグに基づくすべてのアクションへのフルアクセス
- 読み取り/書き込みアクセス
- 読み取り専用アクセス

- 特定のテーブルへの読み取り専用アクセス
- テーブル作成のアクセス許可
- リクエストタグに基づいたテーブル作成のアクセス許可

すべてのアクションへのフルアクセス

次の JSON ポリシードキュメントでは、myExampleLedger のすべてのテーブルですべての PartiQL コマンドを使用できるように、フルアクセスを付与しています。このポリシーは、台帳で ALLOW_ALL アクセス許可モードを使用するのと同じ効果を発揮します。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLFullPermissions",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLCreateIndex",
                "qldb:PartiQLDropIndex",
                "qldb:PartiQLCreateTable",
                "qldb:PartiQLDropTable",
                "qldb:PartiQLUndropTable",
                "qldb:PartiQLDelete",
                "qldb:PartiQLInsert",
                "qldb:PartiQLUpdate",
                "qldb:PartiQLRedact",
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
```

```
            ]
        }
    ]
}
```

テーブルタグに基づくすべてのアクションへのフルアクセス

次の JSON ポリシードキュメントでは、テーブルリソースタグに基づいた条件を使用して、myExampleLedger のすべてのテーブルですべての PartiQL コマンドを使用できるようにフルアクセスを付与しています。アクセス許可は、テーブルタグ environment に値 development がある場合にのみ付与されます。

> **Warning**
>
> これは、ワイルドカード文字 (*) を使用して、すべての PartiQL アクションを許可する例です。これには、QLDB 台帳のすべてのテーブルに対する管理オペレーションと読み取り/書き込みオペレーションが含まれます。代わりに、付与する各アクションと、そのユーザー、ロール、またはグループが必要とするアクションのみを明示的に指定することをお勧めします。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLFullPermissionsBasedOnTags",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQL*"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
```

```
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
            ],
            "Condition": {
                "StringEquals": { "aws:ResourceTag/environment": "development" }
            }
        }
    ]
}
```

読み取り/書き込みアクセス

次の JSON ポリシードキュメントでは、myExampleLedger のすべてのテーブルのデータを選択、挿入、更新、および削除するアクセス許可を付与しています。このポリシーでは、テーブルやインデックスの作成や削除など、データの秘匿化やスキーマの変更を行うアクセス許可は付与されません。

> **Note**
>
> UPDATE ステートメントでは、変更するテーブルに対する qldb:PartiQLUpdate アクションと qldb:PartiQLSelect アクション両方のアクセス許可が必要です。UPDATE ステートメントを実行すると、更新オペレーションに加えて読み取りオペレーションも実行されます。両方のアクションを要求すれば、テーブルの内容の読み取りが許可されているユーザーにのみ UPDATE アクセス許可が付与されるようになります。
> 同様に、DELETE ステートメントでは、qldb:PartiQLDelete アクションと qldb:PartiQLSelect アクションの両方へのアクセス許可が必要です。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
```

```
        {
            "Sid": "QLDBPartiQLReadWritePermissions",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLDelete",
                "qldb:PartiQLInsert",
                "qldb:PartiQLUpdate",
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
            ]
        }
    ]
}
```

読み取り専用アクセス

次の JSON ポリシードキュメントでは、myExampleLedger のすべてのテーブルに読み取り専用アクセス許可を付与しています。このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLReadOnlyPermissions",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
```

```
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
            ]
        }
    ]
}
```

特定のテーブルへの読み取り専用アクセス

次の JSON ポリシードキュメントでは、myExampleLedger の特定のテーブルに読み取り専用アクセス許可を付与しています。この例では、テーブル ID は Au1EiThbt8s0z9wM26REZN です。

このポリシーを使用するには、例の *us-east-1*、*123456789012myExampleLedger*、、Au*Au1EiThbt8s0z9wM26REZN* をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLReadOnlyPermissionsOnTable",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
table/Au1EiThbt8s0z9wM26REZN"
            ]
        }
    ]
}
```

テーブル作成のアクセス許可

次の JSON ポリシードキュメントでは、myExampleLedger のテーブルを作成するアクセス許可を付与しています。qldb:PartiQLCreateTable アクションには、テーブルリソースタイプに対するアクセス許可が必要です。ただし、CREATE TABLE ステートメントを実行する時点で、新しいテーブルのテーブル ID は未知です。したがって、qldb:PartiQLCreateTable アクセス許可を付与するポリシーでは、テーブル ARN でワイルドカード (*) を使用してリソースを指定する必要があります。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLCreateTablePermission",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLCreateTable"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*"
            ]
        }
    ]
}
```

リクエストタグに基づいたテーブル作成のアクセス許可

次の JSON ポリシードキュメントでは、aws:RequestTag コンテキストキーに基づいた条件を使用して、myExampleLedger のテーブルを作成するアクセス許可を付与しています。アクセス許可は、リクエストタグ environment に値 development がある場合にのみ付与されます。作成時にテーブルにタグを付けるには、qldb:PartiQLCreateTable および qldb:TagResource アク

ションの両方にアクセスする必要があります。作成時にテーブルにタグを付ける方法については、「テーブルのタグ付け」を参照してください。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLCreateTablePermission",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLCreateTable",
                "qldb:TagResource"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*"
            ],
            "Condition": {
                "StringEquals": { "aws:RequestTag/environment": "development" }
            }
        }
    ]
}
```

Simple Storage Service (Amazon S3) バケットへのジャーナルのエクスポート

ステップ 1: QLDB ジャーナルのエクスポートのアクセス許可

次の例では、QLDB 台帳リソースで `qldb:ExportJournalToS3` アクションを実行する AWS アカウント アクセス許可をユーザーに付与します。また、QLDB サービスに渡す IAM ロールリソースに対して `iam:PassRole` アクションを実行するためのアクセス許可も付与しています。これはすべてのジャーナルエクスポートリクエストに必要です。

このポリシーを使用するには、例の *us-east-1*、*123456789012myExampleLedger*、、および *qldb-s3-export* をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportPermission",
            "Effect": "Allow",
            "Action": "qldb:ExportJournalToS3",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "IAMPassRolePermission",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::123456789012:role/qldb-s3-export",
            "Condition": {
               "StringEquals": {
                  "iam:PassedToService": "qldb.amazonaws.com"
               }
            }
        }
    ]
}
```

ステップ 2: Simple Storage Service (Amazon S3) バケットのアクセス許可

次の例では、IAM ロールを使用して、Simple Storage Service (Amazon S3) バケットの 1 つ、DOC-EXAMPLE-BUCKET に書き込むアクセス権を QLDB に付与しています。これは、どの QLDB ジャーナルエクスポートでも必要です。

`s3:PutObject` アクセス許可を付与することに加えて、このポリシーは、オブジェクトのアクセスコントロールリスト (ACL) のアクセス許可を設定するための `s3:PutObjectAcl` アクセス許可も付与します。

このポリシーを使用するには、以下の例の DOC-EXAMPLE-BUCKET を Simple Storage Service (Amazon S3) バケット名と置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "QLDBJournalExportS3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        }
    ]
}
```

次に、このアクセス許可ポリシーを、QLDB が Simple Storage Service (Amazon S3) バケットにアクセスするために引き受けることができる IAM ロールにアタッチします。次の JSON ドキュメントは、アカウント 123456789012 のみの QLDB リソースの IAM ロールを QLDB が引き受けることを許可する信頼ポリシーの例です。

このポリシーを使用するには、以下の例の *us-east-1* および *123456789012* を自分の情報と置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

Kinesis Data Streams へのジャーナルのストリーミング

ステップ 1: QLDB ジャーナルストリームのアクセス許可

次の例では、台帳内のすべての QLDB ストリームサブリソースに対して `qldb:StreamJournalToKinesis`アクションを実行する AWS アカウント アクセス許可をユーザーに付与します。また、QLDB サービスに渡す IAM ロールリソースに対して `iam:PassRole` アクションを実行するためのアクセス許可も付与しています。これは、すべてのジャーナルストリームリクエストに必要です。

このポリシーを使用するには、例*`qldb-kinesis-stream`*の *`us-east-1`*、*`123456789012myExampleLedger`*、、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalStreamPermission",
            "Effect": "Allow",
            "Action": "qldb:StreamJournalToKinesis",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
        },
        {
            "Sid": "IAMPassRolePermission",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::123456789012:role/qldb-kinesis-stream",
            "Condition": {
               "StringEquals": {
                  "iam:PassedToService": "qldb.amazonaws.com"
               }
            }
        }
    ]
}
```

ステップ 2: Kinesis Data Streams のアクセス許可

次の例では、IAM ロールを使用して、Amazon Kinesis データストリーム にデータレコードを書き込むためのアクセス権を QLDB に付与します*`stream-for-qldb`*。これは、どの QLDB ジャーナルストリームでも必要です。

このポリシーを使用するには、例*stream-for-qldb*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBStreamKinesisPermissions",
            "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
 "kinesis:ListShards" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/stream-for-qldb"
        }
    ]
}
```

次に、このアクセス許可ポリシーを、QLDB が Kinesis データストリームにアクセスするために引き受けることができる IAM ロールにアタッチします。次の JSON ドキュメントは、台帳 myExampleLedger のみのアカウント 123456789012 の QLDB ストリームの IAM ロールを QLDB が引き受けることを許可する信頼ポリシーの例です。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-
east-1:123456789012:stream/myExampleLedger/*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
```

```
        }
    ]
}
```

タグに基づく QLDB 台帳の更新

アイデンティティベースのポリシーの条件を使用して、タグに基づいて QLDB リソースへのアクセスをコントロールできます。この例では、台帳の更新を許可するポリシーを作成する方法を示しています。ただし、台帳タグ Owner にそのユーザーのユーザー名の値がある場合のみ、アクセス許可は付与されます。このポリシーでは、このアクションをコンソールで実行するために必要なアクセス許可も付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListLedgersInConsole",
            "Effect": "Allow",
            "Action": "qldb:ListLedgers",
            "Resource": "*"
        },
        {
            "Sid": "UpdateLedgerIfOwner",
            "Effect": "Allow",
            "Action": "qldb:UpdateLedger",
            "Resource": "arn:aws:qldb:*:*:ledger/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

このポリシーをアカウントの ユーザーにアタッチできます。richard-roe という名前のユーザーが QLDB 台帳を更新しようとする場合は、台帳にタグ Owner=richard-roe または owner=richard-roe が付いている必要があります。それ以外の場合、そのユーザーのアクセスは拒否されます。条件キー名では大文字と小文字は区別されないため、条件タグキー Owner は Owner と owner に一致します。詳細については、IAM ユーザーガイドのIAM JSON ポリシー要素: 条件を参照してください。

クロスサービスの混乱した副防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。

サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。混乱した代理問題を防ぐために、 は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを持つすべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシー内では aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、Amazon QLDB が別のサービスに付与する、リソースへのアクセス許可を制限することをお勧めします。グローバル条件コンテキストキーの両方を使用しており、それらが同じポリシーステートメントで使用される場合、aws:SourceAccount 値と aws:SourceArn 値のアカウントが同じアカウント ID を使用する必要があります。

次の表は、ExportJournalToS3 および StreamsJournalToKinesis QLDB API オペレーションでの aws:SourceArn の可能な値の一覧です。これらのオペレーションは、 AWS Security Token Service （ AWS STS) を呼び出して、指定した IAM ロールを引き受けるため、このセキュリティ問題の範囲内です。

API オペレーション	呼び出されたサービス	aws:SourceArn
ExportJournalToS3	AWS STS (AssumeRole)	QLDB がアカウントの任意の QLDB リソースのロールを引き受けることを許可します。 `arn:aws:qldb: us-east-1 :123456789012 :*` 現在、QLDB では、このワイルドカード ARN をジャーナルエクスポートでのみサポートしています。

API オペレーション	呼び出されたサービス	aws:SourceArn

StreamsJournalToKinesis	AWS STS (AssumeRole)	QLDB が特定の QLDB ストリーミングのロールを引き受けることを許可します。
		`arn:aws:qldb: us-east-1 :123456789012 :stream/myExample Ledger /IiPT4brpZCqCq3f4MTHbYy`
		注意: ARN でストリーム ID を指定できるのは、ストリームリソースが作成された後のみです。この ARN を使用すると、単一の QLDB ストリームに対してのみロールを使用できるようにできます。
		QLDB が台帳の任意の QLDB ストリームのロールを引き受けることを許可します。
		`arn:aws:qldb: us-east-1 :123456789012 :stream/myExample Ledger /*`
		QLDB がアカウントの任意の QLDB ストリームのロールを引き受けることを許可します。
		`arn:aws:qldb: us-east-1 :123456789012 :stream/*`
		QLDB がアカウントの任意の QLDB リソースのロールを引き受けることを許可します。
		`arn:aws:qldb: us-east-1 :123456789012 :*`

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、aws:SourceArn グローバルコンテキスト条件キーで、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します (例: arn:aws:qldb:us-east-1:*123456789012*:*)。

次の IAM ロールの信頼ポリシーの例では、aws:SourceArn および aws:SourceAccount グローバル条件コンテキストキーを使用して、「混乱した代理」問題を回避する方法を示します。この信頼

ポリシーを使用すると、QLDB は台帳 123456789012 のみでアカウント myExampleLedger の任意の QLDB ストリーミングに対応するロールを引き受けることができます。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "qldb.amazonaws.com"
    },
    "Action": [ "sts:AssumeRole" ],
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

AWS Amazon QLDB の マネージドポリシー

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、多くの一般的なユースケースに対するアクセス許可を付与するように設計されているため、ユーザー、グループ、ロールへのアクセス許可の割り当てを開始できます。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケース別にカスタマーマネージドポリシーを定義して、マネージドポリシーを絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) が更新されます。 は、新しい AWS のサービス が起動されたとき、

または既存のサービスで新しい API AWS オペレーションが使用可能になったときに、 AWS 管理ポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

これらの管理ポリシーの QLDB API オペレーションの詳細については、 AWS 「」を参照してくださいAmazon QLDB API リファレンス。

トピック

- AWS マネージドポリシー: AmazonQLDBReadOnly
- AWS マネージドポリシー: AmazonQLDBFullAccess
- AWS マネージドポリシー: AmazonQLDBConsoleFullAccess
- QLDB の AWS マネージドポリシーの更新

AWS マネージドポリシー: AmazonQLDBReadOnly

AmazonQLDBReadOnly ポリシーを使用して、すべての QLDB リソースに読み取り専用アクセス許可を付与します。このポリシーは IAM アイデンティティにアタッチできます。

許可の詳細

このポリシーには、`qldb` サービスに対する以下のアクセス許可が含まれます。

- プリンシパルに対し、すべての QLDB リソースとそのタグを記述し、一覧表示することを許可します。これらのリソースには、台帳、Simple Storage Service (Amazon S3) のエクスポートジョブ、Kinesis Data Streams へのストリームが含まれます。
- プリンシパルに対し、任意の台帳のジャーナルからブロック、ダイジェスト、またはリビジョンを取得して、そのデータを暗号的に検証することを許可します。
- プリンシパルは、任意の台帳の任意のテーブルで任意の PartiQL コマンドを実行することは許可されていません。

AWS マネージドポリシー: AmazonQLDBFullAccess

AmazonQLDBFullAccess ポリシーを使用して、QLDB API または を介してすべての QLDB リソースに完全な管理アクセス許可を付与します AWS CLI。このポリシーは IAM アイデンティティにアタッチできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `qldb`
  - プリンシパルに対し、すべての QLDB リソースとそのタグを作成、記述、一覧表示、および管理することを許可します。これらのリソースには、台帳、Simple Storage Service (Amazon S3) のエクスポートジョブ、Kinesis Data Streams へのストリームが含まれます。
  - プリンシパルに対し、QLDB ドライバーまたは QLDB シェルを使用して、任意の台帳のすべてのテーブルですべての PartiQL コマンドを実行することを許可します。
  - プリンシパルに対し、任意の台帳のジャーナルからブロック、ダイジェスト、またはリビジョンを取得して、そのデータを暗号的に検証することを許可します。
- `iam` – プリンシパルに対し、アカウントの任意の IAM ロールリソースを QLDB サービスに渡すことを許可します。これは、ジャーナルのどのエクスポートとストリーミングリクエストにも必要です。

AWS マネージドポリシー: AmazonQLDBConsoleFullAccess

AmazonQLDBConsoleFullAccess ポリシーを使用して、、QLDB API AWS Management Console、または を介してすべての QLDB リソースに完全な管理アクセス許可を付与します AWS CLI。このポリシーは IAM アイデンティティにアタッチできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `qldb`
  - プリンシパルに対し、すべての QLDB リソースとそのタグを作成、記述、一覧表示、および管理することを許可します。これらのリソースには、台帳、Simple Storage Service (Amazon S3) のエクスポートジョブ、Kinesis Data Streams へのストリームが含まれます。
  - プリンシパルに対し、QLDB コンソール、QLDB ドライバー、または QLDB シェルを使用して、任意の台帳のすべてのテーブルですべての PartiQL コマンドを実行することを許可します。
  - プリンシパルに対し、QLDB コンソールを使用して任意の台帳にサンプルアプリケーションデータを挿入することを許可します。
  - プリンシパルに対し、任意の台帳のジャーナルからブロック、ダイジェスト、またはリビジョンを取得して、そのデータを暗号的に検証することを許可します。

- dbqms – プリンシパルに対し、Database Query Metadata Service のすべてのアクションを使用することを許可します。これは、QLDB コンソールでは、PartiQL クエリエディタ用に最近保存されたクエリを作成、記述、管理するために必要な内部専用のサービスです。
- kinesis – プリンシパルに対し、Amazon Kinesis Data Streams リソースの記述と一覧表示を許可します。このリソースは、QLDB ストリームリソースがデータを書き込むことができるターゲット宛先です。
- iam – プリンシパルに対し、アカウントの任意の IAM ロールリソースを QLDB サービスに渡すことを許可します。これは、ジャーナルのどのエクスポートとストリーミングリクエストにも必要です。

QLDB の AWS マネージドポリシーの更新

このサービスがこれらの変更の追跡を開始した以降の QLDB の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知については、QLDB の「リリース履歴」ページの RSS フィードにサブスクライブしてください。

変更	説明	日付
AmazonQLDBFullAccess、AmazonQLDBConsoleFullAccess – 既存のポリシーの更新	QLDB は、プリンシパルがすべての台帳のドキュメントリビジョンを STANDARD 権限モードで秘匿化できるようにする新しい権限を追加しました。	2022 年 11 月 4 日
AmazonQLDBFullAccess、AmazonQLDBConsoleFullAccess – 既存のポリシーの更新	QLDB では、プリンシパルに対し、アカウントの任意の IAM ロールリソースを QLDB サービスに渡すことを許可する新しいアクセス許可が追加されました。これは、ジャーナルのどのエクスポートとストリーミングリクエストにも必要です。	2021 年 9 月 2 日

変更	説明	日付
AmazonQLDBReadOnly – 既存のポリシーの更新	QLDB では、以前は 2 回リストされていた重複する `qldb:GetBlock` アクションが削除され、また `"Effect"` フィールドの順序が変更されて `"Action"` フィールドの前に表示されるようになりました。	2021 年 7 月 1 日

変更	説明	日付
AmazonQLDBFullAccess、AmazonQLDBConsoleFullAccess – 既存のポリシーの更新	QLDB では、プリンシパルに対し、すべての台帳のアクセス許可モードを更新し、新しい STANDARD アクセス許可モードのすべての台帳ですべての PartiQL コマンドを実行することを許可する新しいアクセス許可が追加されました。 STANDARD アクセス許可モードは、PartiQL コマンドのテーブルレベルのアクセスコントロールと詳細度をサポートします。新しいアクセス許可モードを容易にするために、QLDB では、PartiQL コマンドタイプに対する IAM アクションのセット、および QLDB テーブルリソースの Amazon リソースネーム (ARN) が導入されました。これらの 2 つのポリシーが更新され、STANDARD 台帳にフルアクセスを付与する新しい PartiQL アクションが含まれるようになりました。	2021 年 5 月 27 日
QLDB が変更の追跡を開始しました	QLDB が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 3 月 1 日

Amazon QLDB アイデンティティとアクセスのトラブルシューティング

次の情報は、QLDB と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

QLDB でアクションを実行する権限がない

からアクションを実行する権限がないと AWS Management Console 通知された場合は、管理者に連絡してサポートを依頼する必要があります。管理者とは、サインイン認証情報を提供した担当者です。

以下のエラー例は、`mateojackson` ユーザーがコンソールを使用して架空の *`myExampleLedger`* リソースに関する詳細情報を表示しようとしているが、架空の `qldb:`*`DescribeLedger`* 許可がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 qldb:DescribeLedger on resource: myExampleLedger
```

この場合、Mateo は、`qldb:`*`DescribeLedger`* アクションを使用して *`myExampleLedger`* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

iam を実行する権限がありません。PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して QLDB にロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して QLDB でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡す権限がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

ジャーナルのエクスポートまたはストリーム操作に固有のこのエラーに関するトラブルシューティングのガイダンスについては、「Amazon QLDB のトラブルシューティング」を参照してください。

自分の 以外のユーザーに QLDB リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- QLDB がこれらの機能をサポートしているかどうかを確認するには、「Amazon QLDB で IAM が機能する仕組み」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティー に提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権限」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、『IAM ユーザーガイド』の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon QLDB でのログ記録とモニタリング

モニタリングは、Amazon QLDB および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。ただし、QLDB のモニタリングを開始する前に、以下の質問に対する回答を反映したモニタリング計画を作成する必要があります。

- モニタリングの目的は何ですか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまなタイミングと負荷条件でパフォーマンスを測定することにより、お客様の環境で通常の QLDB のパフォーマンスのベースラインを確定します。QLDB のモニタリングでは、過去のモニタリングデータを保存し、現在のパフォーマンスデータと比較することで、パフォーマンスの通常パターンと異常パターンを特定し、問題に対処する方法を考案できます。

ベースラインを確立するには、少なくとも、次の項目をモニタリングする必要があります。

- 読み取りおよび書き込み I/O とストレージ。請求目的で台帳の消費パターンを追跡できます。
- コマンドレイテンシー。データオペレーションを実行するときの台帳のパフォーマンスを追跡できます。
- 例外。リクエストがエラーになったかどうかを判断できます。

トピック

モニタリングツール

AWS には、Amazon QLDB のモニタリングに使用できるさまざまなツールが用意されています。これらのツールの一部はモニタリングを行うように設定できますが、一部のツールは手動による介入が必要です。モニタリングタスクをできるだけ自動化することをお勧めします。

トピック

自動モニタリングツール

以下の自動化されたモニタリングツールを使用して、QLDB を監視し、問題が発生したときにレポートできます。

- Amazon CloudWatch アラーム – 指定した期間にわたって 1 つのメトリクスを監視し、複数の期間にわたって特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) トピックまたは Amazon EC2 Auto Scaling ポリシーに送信される通知です。 CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変更され、指定された期間維持されている必要があります。詳細については、「Amazon によるモニタリング CloudWatch」を参照してください。
- Amazon CloudWatch Logs – またはその他のソースからのログファイルをモニタリング、保存 AWS CloudTrail 、およびアクセスします。詳細については、「Amazon ユーザーガイド」の「ログファイルのモニタリング」を参照してください。 CloudWatch
- Amazon CloudWatch Events – イベントを照合し、1 つ以上のターゲット関数またはストリームにルーティングして、変更を行い、状態情報をキャプチャして、是正措置を講じます。詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch イベントとは」を参照してください。 CloudWatch
- AWS CloudTrail ログモニタリング – アカウント間でログファイルを共有し、 CloudTrail ログファイルを CloudWatch ログに送信してリアルタイムでモニタリングし、Java でログ処理アプリケーションを書き込み、 による配信後にログファイルが変更されていないことを検証します CloudTrail。詳細については、「 ユーザーガイド」の CloudTrail 「ログファイルの使用AWS CloudTrail 」を参照してください。

手動モニタリングツール

QLDB のモニタリングでもう 1 つ重要な点は、 CloudWatch アラームでカバーされない項目を手動でモニタリングすることです。QLDB CloudWatch、 Trusted Advisor、およびその他の AWS Management Console ダッシュボードには、 AWS 環境の状態 at-a-glance が表示されます。Amazon QLDB のログファイルを確認することもお勧めします。

- QLDB ダッシュボードでは以下について確認できます。
  - 読み取りおよび書き込み I/O
  - ジャーナルおよびインデックス付きストレージ
  - コマンドレイテンシー
  - 例外
- CloudWatch ホームページには以下が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  さらに、 CloudWatch を使用して次の操作を実行できます。
  - 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成します
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
  - すべての AWS リソースメトリクスを検索して参照する
  - 問題があることを通知するアラームを作成/編集する

Amazon によるモニタリング CloudWatch

Amazon QLDB は CloudWatch、Amazon QLDB から raw データを収集して読み取り可能な near-real-time メトリクスに処理する を使用してモニタリングできます。これらの統計は 2 週間記録されるため、履歴情報にアクセスしてウェブアプリケーションまたはサービスのパフォーマンスをより的確に把握できます。デフォルトでは、QLDB メトリクスデータは 1 CloudWatch 分または 15 分間隔で に自動的に送信されます。詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch、Amazon CloudWatch Events、Amazon CloudWatch Logs とは」を参照してください。 CloudWatch

トピック

- QLDB メトリクスの使用方法

- Amazon QLDB のメトリクスとディメンション
- Amazon QLDB をモニタリングする CloudWatch アラームの作成

QLDB メトリクスの使用方法

QLDB によってレポートされるメトリクスが提供する情報は、さまざまな方法で分析できます。以下のリストは、メトリクスの一般的な利用方法をいくつか示しています。ここで紹介するのは開始するための提案事項です。すべてを網羅しているわけではありません。

- 指定した期間にわたって `JournalStorage` と `IndexedStorage` をモニタリングして、台帳が消費しているディスク容量を追跡できます。
- 指定した期間にわたって `ReadIOs` と `WriteIOs` をモニタリングして、台帳が処理しているリクエストの数を追跡できます。
- `CommandLatency` をモニタリングして、データオペレーションに対する台帳のパフォーマンスを追跡し、最もレイテンシーが長くなるコマンドのタイプを分析できます。

Amazon QLDB のメトリクスとディメンション

Amazon QLDB を操作すると、次のメトリクスとディメンションが に送信されます CloudWatch。ストレージメトリクスは 15 分ごとにレポートされ、その他のすべてのメトリクスは 1 分ごとに集計されてレポートされます。QLDB のメトリクスを表示するには、以下の手順を使用できます

CloudWatch コンソールを使用してメトリクスを表示するには

メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 必要に応じて、リージョンを変更します。ナビゲーションバーで、 AWS リソースがあるリージョンを選択します。詳細については、「リージョンとエンドポイント」を参照してください。
3. ナビゲーションペインで メトリクスを選択します。
4. [All metrics] (すべてのメトリクス) タブで、[QLDB] を選択します。

を使用してメトリクスを表示するには AWS CLI

- コマンドプロンプトで、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace "AWS/QLDB"
```

CloudWatch は、QLDB の次のメトリクスを表示します。

Amazon QLDB のディメンションとメトリクス

Amazon QLDB が Amazon に送信するメトリクスとディメンション CloudWatch をここに一覧表示します。

QLDB のメトリクス

メトリクス	説明
JournalStorage	台帳のジャーナルによって使用されているディスク容量の合計 (15 分間隔でレポート)。ジャーナルには、データへのすべての変更に関する完全、不変、検証可能な履歴が含まれています。 単位: Bytes ディメンション: LedgerName
IndexedStorage	台帳のテーブル、インデックス、インデックス付き履歴によって使用されているディスク容量の合計 (15 分間隔でレポート)。インデックス付きストレージは、高パフォーマンスなクエリ用に最適化された台帳データで構成されています。 単位: Bytes ディメンション: LedgerName
ReadIOs	読み取り I/O リクエストの数。1 分間隔でレポートされます。これにより、データトランザクション、検証リクエスト、ジャーナルのエクスポート、ジャーナルストリームなど、あらゆる種類の読み取りオペレーションがキャプチャされます。 単位: Count

メトリクス	説明
	ディメンション: LedgerName
WriteIOs	書き込み I/O リクエストの数。1 分間隔でレポートされます。 単位: Count ディメンション: LedgerName
CommandLatency	データオペレーションにかかった時間。1 分間隔でレポートされます。 単位: Milliseconds ディメンション: CommandType, LedgerName
IsImpaired	Kinesis Data Streams へのジャーナルストリームに障害があるかどうかを示すフラグ。1 分間隔で報告されます。値 1 は、ストリームが障害状態であることを示し、0 はそうでないことを示します。 単位: Boolean (0 または 1) ディメンション: LedgerName, StreamId
OccConflictExceptions	OccConflictException を生成する QLDB へのリクエストの数。オプティミスティック同時実行制御 (OCC) については、「Amazon QLDB 同時実行モデル」を参照してください。 単位: Count
Session4xxExceptions	HTTP 4xx エラーを生成する QLDB へのリクエストの数。 単位: Count

メトリクス	説明
Session5xxExceptions	HTTP 5xx エラーを生成する QLDB へのリクエストの数。 単位: Count
SessionRateExceededExceptions	SessionRateExceededException を生成する QLDB へのリクエストの数。 単位: Count

QLDB のメトリクスのディメンション

QLDB のメトリクスはアカウント、台帳名、ストリーム ID、またはコマンドタイプの値によって修飾されます。 CloudWatch コンソールを使用して、次の表の任意のディメンションに沿って QLDB データを取得できます。

ディメンション	説明
LedgerName	このディメンションは、特定の台帳にデータを制限します。この値は、現在の AWS リージョン と現在の の任意の台帳名にすることができます AWS アカウント。
StreamId	このディメンションは、特定のジャーナルストリームにデータを制限します。この値は、現在の AWS リージョン と現在の の台帳の任意のストリーム ID にすることができます AWS アカウント。
CommandType	このディメンションは、以下の QLDB データ API コマンドのいずれかにデータを制限します。 • AbortTransaction • CommitTransaction • EndSession • ExecuteStatement • FetchPage

ディメンション	説明
	• `StartSession` • `StartTransaction` QLDB がこれらのコマンドを使用してデータオペレーションを管理する方法については、「ドライバーによるセッション管理」を参照してください。

Amazon QLDB をモニタリングする CloudWatch アラームの作成

CloudWatch アラームの状態が変更されたときに Amazon Simple Notification Service (Amazon SNS) メッセージを送信する Amazon アラームを作成できます。指定した期間中、1 つのアラームが 1 つのメトリクスを監視します。このアラームは、複数の期間にわたる一定のしきい値とメトリクスの値の関係性に基づき、1 つ以上のアクションを実行します。アクションは、Amazon SNS のトピックまたは自動スケーリングのポリシーに送信される通知です。

アラームは、持続する状態の変化に対してのみアクションを呼び出します。 CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。

CloudWatch アラームの作成の詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch アラームの使用」を参照してください。 CloudWatch

CloudWatch イベントによる Amazon QLDB の自動化

Amazon CloudWatch Events を使用すると、 を自動化 AWS のサービス し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。からのイベント AWS のサービス は、ほぼリアルタイムで CloudWatch イベントに配信されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。自動的にトリガーできるオペレーションには、以下が含まれます。

- AWS Lambda 関数の呼び出し
- Amazon EC2 Run Command の呼び出し
- Amazon Kinesis Data Streams へのイベントの中継
- AWS Step Functions ステートマシンのアクティブ化
- Amazon SNS トピックまたは Amazon SQS キューの通知

Amazon QLDB は、 内の台帳リソースの状態が AWS アカウント 変化するたびに、イベントを CloudWatch Events に報告します。現在、イベントは QLDB 台帳リソースに対してのみ保証 at-least-once されたベースで発行されます。

以下の例に示しているのは、台帳の状態が DELETING に変わったときに QLDB がレポートしたイベントです。

```
{
    "version" : "0",
    "id" : "2f6557eb-e361-54ef-0f9f-99dd9f171c62",
    "detail-type" : "QLDB Ledger State Change",
    "source" : "aws.qldb",
    "account" : "123456789012",
    "time" : "2019-07-24T21:59:17Z",
    "region" : "us-east-1",
    "resources" : ["arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"],
    "detail" : {
        "ledgerName" : "exampleLedger",
        "state" : "DELETING"
    }
}
```

QLDB で CloudWatch Events を使用する例には、以下が含まれますが、これらに限定されません。

- 新しい台帳が最初に CREATING 状態で作成され、最終的に ACTIVE になるたびに、Lambda 関数をアクティブにする。
- 台帳の状態が DELETING に変わり、次に DELETED に変わったときに Amazon SNS トピックを通知する。

詳細については、「Amazon CloudWatch Events ユーザーガイド」を参照してください。

を使用した Amazon QLDB API コールのログ記録 AWS CloudTrail

Amazon QLDB は AWS CloudTrail、QLDB のユーザー、ロール、または によって実行されたアクションを記録するサービスである と統合 AWS のサービス されています。 は、QLDB のすべてのリソース管理 API コールをイベントとして CloudTrail キャプチャします。キャプチャされた呼び出しには、QLDB コンソールの呼び出しと、QLDB API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、QLDB の CloudTrail イベントなど、Amazon Simple Storage Service (Amazon S3) バケットへのイベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、イベント履歴 で CloudTrail コンソールで最新のイベントを表示できます。 によって収

集された情報を使用して CloudTrail、QLDB に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の設定と有効化の方法など CloudTrail、 の詳細については、AWS CloudTrail 「 ユーザーガイド」を参照してください。

の QLDB 情報 CloudTrail

CloudTrail アカウントを作成する AWS アカウント と、 で が有効になります。QLDB でサポートされているアクティビティが発生すると、そのアクティビティは CloudTrail イベント履歴 の他の AWS のサービス イベントとともにイベントに記録されます。 で最近のイベントを表示、検索、ダウンロードできます AWS アカウント。詳細については、「イベント履歴 を含む CloudTrail イベントの表示」を参照してください。

QLDB のイベントなど AWS アカウント、 のイベントの継続的な記録については、証跡を作成します。証跡により CloudTrail 、 はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、他の を設定 AWS のサービス して、CloudTrail ログで収集されたイベントデータをさらに分析し、それに基づく対応を行うことができます。

詳細については、『AWS CloudTrail ユーザーガイド:』の以下のトピックを参照してください。

- 証跡作成の概要
- CloudTrail がサポートするサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信
- 複数のアカウントからの CloudTrail ログファイルの受信

すべての QLDB リソース管理および非トランザクションデータ API アクションは、 によってログに記録 CloudTrail され、Amazon QLDB API リファレンス に記載されています。例えば、`CreateLedger`、、および `DeleteLedger`アクションを呼び出すと`DescribeLedger`、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます:

- リクエストが、ルートと ユーザー認証情報のどちらを使用して送信されたか

- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか
- リクエストが別の によって行われたかどうか AWS のサービス

詳細については、CloudTrail userIdentity 要素」を参照してください。

QLDB ログファイルエントリの概要

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、これらのアクションを示す CloudTrail ログエントリを示しています。

- `CreateLedger`
- `DescribeLedger`
- `ListTagsForResource`
- `TagResource`
- `UntagResource`
- `ListLedgers`
- `GetDigest`
- `GetBlock`
- `GetRevision`
- `ExportJournalToS3`
- `DescribeJournalS3Export`
- `ListJournalS3ExportsForLedger`
- `ListJournalS3Exports`
- `DeleteLedger`

```
{
  "endTime": 1561497717208,
  "startTime": 1561497687254,
  "calls": [
```

```
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:27Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "CreateLedger",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "Name": "CloudtrailTest",
          "PermissionsMode": "ALLOW_ALL"
        },
        "responseElements": {
          "CreationDateTime": 1.561497687403E9,
          "Arn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
          "State": "CREATING",
          "Name": "CloudtrailTest"
        },
        "requestID": "3135aec7-978f-11e9-b313-1dd92a14919e",
        "eventID": "bf703ff9-676f-41dd-be6f-5f666c9f7852",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:27Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"CreateLedger\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":
{\"Name\":\"CloudtrailTest\",\"PermissionsMode\":\"ALLOW_ALL\"},\"responseElements
\":{\"CreationDateTime\":1.561497687403E9,\"Arn\":\"arn:aws:qldb:us-
east-2:123456789012:ledger/CloudtrailTest\",\"State\":\"CREATING\",\"Name\":
\"CloudtrailTest\"},\"requestID\":\"3135aec7-978f-11e9-b313-1dd92a14919e\",\"eventID\":
\"bf703ff9-676f-41dd-be6f-5f666c9f7852\",\"readOnly\":false,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
```

```
      "name": "CreateLedger",
      "request": [
        "com.amazonaws.services.qldb.model.CreateLedgerRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "tags": null,
          "permissionsMode": "ALLOW_ALL"
        }
      ],
      "requestId": "3135aec7-978f-11e9-b313-1dd92a14919e"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:43Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "DescribeLedger",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
        },
        "responseElements": null,
        "requestID": "3af51ba0-978f-11e9-8ae6-837dd17a19f8",
        "eventID": "be128e61-3e38-4503-83de-49fdc7fc0afb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity
\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-
user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",
\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",
\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate
\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":
\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId
\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:43Z
\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"DescribeLedger\",
\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":
\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-
```

```
Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation
\",\"requestParameters\":{\"name\":\"CloudtrailTest\"},\"responseElements
\":null,\"requestID\":\"3af51ba0-978f-11e9-8ae6-837dd17a19f8\",\"eventID\":
\"be128e61-3e38-4503-83de-49fdc7fc0afb\",\"readOnly\":true,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "DescribeLedger",
      "request": [
        "com.amazonaws.services.qldb.model.DescribeLedgerRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest"
        }
      ],
      "requestId": "3af51ba0-978f-11e9-8ae6-837dd17a19f8"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "TagResource",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest",
          "Tags": {
            "TagKey": "TagValue"
          }
        },
        "responseElements": null,
        "requestID": "3b1d6371-978f-11e9-916c-b7d64ec76521",
        "eventID": "6101c94a-7683-4431-812b-9a91afb8c849",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
```

{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"TagResource\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"resourceArn\":\"arn
%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest\",\"Tags\":{\"TagKey
\":\"TagValue\"}},\"responseElements\":null,\"requestID\":\"3b1d6371-978f-11e9-916c-
b7d64ec76521\",\"eventID\":\"6101c94a-7683-4431-812b-9a91afb8c849\",\"readOnly\":false,
\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "TagResource",
      "request": [
        "com.amazonaws.services.qldb.model.TagResourceRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
          "tags": {
            "TagKey": "TagValue"
          }
        }
      ],
      "requestId": "3b1d6371-978f-11e9-916c-b7d64ec76521"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListTagsForResource",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest"
        },
        "responseElements": null,
        "requestID": "3b56c321-978f-11e9-8527-2517d5bfa8fd",
        "eventID": "375e57d7-cf94-495a-9a48-ac2192181c02",
        "readOnly": true,

```
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity
\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-
user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",
\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",
\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate
\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId
\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin
\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":
\"2019-06-25T21:21:44Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName
\":\"ListTagsForResource\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":
\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6
 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/
Oracle_Corporation\",\"requestParameters\":{\"resourceArn\":\"arn%3Aaws%3Aqldb
%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest\"},\"responseElements\":null,
\"requestID\":\"3b56c321-978f-11e9-8527-2517d5bfa8fd\",\"eventID\":\"375e57d7-
cf94-495a-9a48-ac2192181c02\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",
\"recipientAccountId\":\"123456789012\"}",
      "name": "ListTagsForResource",
      "request": [
        "com.amazonaws.services.qldb.model.ListTagsForResourceRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest"
        }
      ],
      "requestId": "3b56c321-978f-11e9-8527-2517d5bfa8fd"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "UntagResource",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "tagKeys": "TagKey",
```

```
          "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest"
        },
        "responseElements": null,
        "requestID": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9",
        "eventID": "bcdcdca3-699f-4363-b092-88242780406f",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"UntagResource\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"tagKeys\":
\"TagKey\",\"resourceArn\":\"arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest\"},\"responseElements\":null,\"requestID\":\"3b87e59b-978f-11e9-8b9a-
bb6dc3a800a9\",\"eventID\":\"bcdcdca3-699f-4363-b092-88242780406f\",\"readOnly\":false,
\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "UntagResource",
      "request": [
        "com.amazonaws.services.qldb.model.UntagResourceRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
          "tagKeys": [
            "TagKey"
          ]
        }
      ],
      "requestId": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
```

```
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListLedgers",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": null,
        "responseElements": null,
        "requestID": "3bafb877-978f-11e9-a6de-dbe6464b9dec",
        "eventID": "6ebe7d49-af59-4f29-aaa2-beffe536e20c",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"ListLedgers\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":null,
\"responseElements\":null,\"requestID\":\"3bafb877-978f-11e9-a6de-dbe6464b9dec\",
\"eventID\":\"6ebe7d49-af59-4f29-aaa2-beffe536e20c\",\"readOnly\":true,\"eventType\":
\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ListLedgers",
      "request": [
        "com.amazonaws.services.qldb.model.ListLedgersRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "maxResults": null,
          "nextToken": null
        }
      ],
      "requestId": "3bafb877-978f-11e9-a6de-dbe6464b9dec"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
```

```
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:49Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "GetDigest",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
        },
        "responseElements": null,
        "requestID": "3cddd8a1-978f-11e9-a6de-dbe6464b9dec",
        "eventID": "a5cb60db-e6c5-4f5e-a5fc-0712249622b3",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:49Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"GetDigest\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"name\":
\"CloudtrailTest\"},\"responseElements\":null,\"requestID\":\"3cddd8a1-978f-11e9-a6de-
dbe6464b9dec\",\"eventID\":\"a5cb60db-e6c5-4f5e-a5fc-0712249622b3\",\"readOnly\":true,
\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "GetDigest",
      "request": [
        "com.amazonaws.services.qldb.model.GetDigestRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "digestTipAddress": null
        }
      ],
      "requestId": "3cddd8a1-978f-11e9-a6de-dbe6464b9dec"
    },
```

```
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:50Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "GetBlock",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "BlockAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
          },
          "name": "CloudtrailTest",
          "DigestTipAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
          }
        },
        "responseElements": null,
        "requestID": "3eaea09f-978f-11e9-bdc2-c1e55368155e",
        "eventID": "1f7da83f-d829-4e35-953d-30b925ceee66",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:50Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"GetBlock\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"BlockAddress
\":{\"IonText\":\"{strandId:\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:0}\"},
\"name\":\"CloudtrailTest\",\"DigestTipAddress\":{\"IonText\":\"{strandId:
\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:0}\"}},\"responseElements\":null,
\"requestID\":\"3eaea09f-978f-11e9-bdc2-c1e55368155e\",\"eventID\":\"1f7da83f-
d829-4e35-953d-30b925ceee66\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",
\"recipientAccountId\":\"123456789012\"}",
```

```
      "name": "GetBlock",
      "request": [
        "com.amazonaws.services.qldb.model.GetBlockRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "blockAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
          },
          "digestTipAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
          }
        }
      ],
      "requestId": "3eaea09f-978f-11e9-bdc2-c1e55368155e"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:55Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "GetRevision",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "BlockAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          },
          "name": "CloudtrailTest",
          "DocumentId": "8UyXvDw6ApoFfVOA2HPfUE",
          "DigestTipAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          }
        },
        "responseElements": null,
        "requestID": "41e19139-978f-11e9-aaed-dfe1dafe37ab",
        "eventID": "43bf2661-5046-41ec-a1d3-87706954aa10",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
```

```
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:55Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"GetRevision\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"BlockAddress
\":{\"IonText\":\"{strandId:\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:1}\"},\"name
\":\"CloudtrailTest\",\"DocumentId\":\"8UyXvDw6ApoFfVOA2HPfUE\",\"DigestTipAddress
\":{\"IonText\":\"{strandId:\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:1}\"}},
\"responseElements\":null,\"requestID\":\"41e19139-978f-11e9-aaed-dfe1dafe37ab\",
\"eventID\":\"43bf2661-5046-41ec-a1d3-87706954aa10\",\"readOnly\":true,\"eventType\":
\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "GetRevision",
      "request": [
        "com.amazonaws.services.qldb.model.GetRevisionRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "blockAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          },
          "documentId": "8UyXvDw6ApoFfVOA2HPfUE",
          "digestTipAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          }
        }
      ],
      "requestId": "41e19139-978f-11e9-aaed-dfe1dafe37ab"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ExportJournalToS3",
```

```
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "InclusiveStartTime": 1.561497687254E9,
          "name": "CloudtrailTest",
          "S3ExportConfiguration": {
            "Bucket": "cloudtrailtests-123456789012-us-east-2",
            "Prefix": "CloudtrailTestsJournalExport",
            "EncryptionConfiguration": {
              "ObjectEncryptionType": "SSE_S3"
            }
          },
          "ExclusiveEndTime": 1.561497715795E9
        },
        "responseElements": {
          "ExportId": "BabQhsmJRYDCGMnA2xYBDG"
        },
        "requestID": "423815f8-978f-11e9-afcf-55f7d0f3583d",
        "eventID": "1b5abdc4-52fa-435f-857e-8995ef7a19b7",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"ExportJournalToS3\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"InclusiveStartTime
\":1.561497687254E9,\"name\":\"CloudtrailTest\",\"S3ExportConfiguration\":{\"Bucket\":
\"cloudtrailtests-123456789012-us-east-2\",\"Prefix\":\"CloudtrailTestsJournalExport\",
\"EncryptionConfiguration\":{\"ObjectEncryptionType\":\"SSE_S3\"}},\"ExclusiveEndTime
\":1.561497715795E9},\"responseElements\":{\"ExportId\":\"BabQhsmJRYDCGMnA2xYBDG
\"},\"requestID\":\"423815f8-978f-11e9-afcf-55f7d0f3583d\",\"eventID\":
\"1b5abdc4-52fa-435f-857e-8995ef7a19b7\",\"readOnly\":false,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ExportJournalToS3",
      "request": [
```

```
        "com.amazonaws.services.qldb.model.ExportJournalToS3Request",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "inclusiveStartTime": 1561497687254,
          "exclusiveEndTime": 1561497715795,
          "s3ExportConfiguration": {
            "bucket": "cloudtrailtests-123456789012-us-east-2",
            "prefix": "CloudtrailTestsJournalExport",
            "encryptionConfiguration": {
              "objectEncryptionType": "SSE_S3",
              "kmsKeyArn": null
            }
          }
        }
      ],
      "requestId": "423815f8-978f-11e9-afcf-55f7d0f3583d"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "DescribeJournalS3Export",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest",
          "exportId": "BabQhsmJRYDCGMnA2xYBDG"
        },
        "responseElements": null,
        "requestID": "427ebbbc-978f-11e9-8888-e9894c9c4bb9",
        "eventID": "ca8ffc88-16ff-45f5-9042-d94fadb389c3",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
```

{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"DescribeJournalS3Export\",\"awsRegion\":
\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3
 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"name
\":\"CloudtrailTest\",\"exportId\":\"BabQhsmJRYDCGMnA2xYBDG\"},\"responseElements
\":null,\"requestID\":\"427ebbbc-978f-11e9-8888-e9894c9c4bb9\",\"eventID\":
\"ca8ffc88-16ff-45f5-9042-d94fadb389c3\",\"readOnly\":true,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "DescribeJournalS3Export",
      "request": [
        "com.amazonaws.services.qldb.model.DescribeJournalS3ExportRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "exportId": "BabQhsmJRYDCGMnA2xYBDG"
        }
      ],
      "requestId": "427ebbbc-978f-11e9-8888-e9894c9c4bb9"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListJournalS3ExportsForLedger",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
        },
        "responseElements": null,
        "requestID": "429ca40c-978f-11e9-8c4b-d13a8018a286",
        "eventID": "34f0e76b-58a5-45be-881c-786d22e34e96",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },

```
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"ListJournalS3ExportsForLedger\",
\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":
\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-
Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation
\",\"requestParameters\":{\"name\":\"CloudtrailTest\"},\"responseElements
\":null,\"requestID\":\"429ca40c-978f-11e9-8c4b-d13a8018a286\",\"eventID\":
\"34f0e76b-58a5-45be-881c-786d22e34e96\",\"readOnly\":true,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ListJournalS3ExportsForLedger",
      "request": [
        "com.amazonaws.services.qldb.model.ListJournalS3ExportsForLedgerRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "maxResults": null,
          "nextToken": null
        }
      ],
      "requestId": "429ca40c-978f-11e9-8c4b-d13a8018a286"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListJournalS3Exports",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": null,
        "responseElements": null,
        "requestID": "42cc1814-978f-11e9-befb-f5dbaa142118",
        "eventID": "4c24d7d6-810c-4cf4-884e-00482278b6ce",
        "readOnly": true,
```

```
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"ListJournalS3Exports\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":null,
\"responseElements\":null,\"requestID\":\"42cc1814-978f-11e9-befb-f5dbaa142118\",
\"eventID\":\"4c24d7d6-810c-4cf4-884e-00482278b6ce\",\"readOnly\":true,\"eventType\":
\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ListJournalS3Exports",
      "request": [
        "com.amazonaws.services.qldb.model.ListJournalS3ExportsRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "maxResults": null,
          "nextToken": null
        }
      ],
      "requestId": "42cc1814-978f-11e9-befb-f5dbaa142118"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:57Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "DeleteLedger",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
        },
        "responseElements": null,
```

```
        "requestID": "42f439b9-978f-11e9-8b2c-69ef598d66e9",
        "eventID": "429f5163-cba5-4d86-bd7e-f606e057c6cf",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:57Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"DeleteLedger\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters
\":{\"name\":\"CloudtrailTest\"},\"responseElements\":null,\"requestID\":
\"42f439b9-978f-11e9-8b2c-69ef598d66e9\",\"eventID\":\"429f5163-cba5-4d86-bd7e-
f606e057c6cf\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":
\"123456789012\"}",
      "name": "DeleteLedger",
      "request": [
        "com.amazonaws.services.qldb.model.DeleteLedgerRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest"
        }
      ],
      "requestId": "42f439b9-978f-11e9-8b2c-69ef598d66e9"
    }
  ]
}
```


Amazon QLDB のコンプライアンス検証

サードパーティーの監査者は、以下を含むがこれらに限定されない複数の コンプライアンスプログラムの一環として Amazon QLDB のセキュリティと AWS コンプライアンスを評価します。

- System and Organization Controls (SOC)

- Payment Card Industry (PCI)
- 国際標準化機構（ISO）
- 政府情報システムのためのセキュリティ評価制度 (ISMAP)
- Health Insurance Portability and Accountability Act (HIPAA)

> **Note**
>
> これは、Amazon QLDB 認定資格の包括的なリストではありません。

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

  > **Note**
  >
  > すべての AWS のサービス が HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。

- AWS カスタマーコンプライアンスガイド — コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon QLDB の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。物理的に分離および分離された複数のアベイラビリティーゾーン AWS リージョン を提供し、低レイテンシー、高スループット、高冗長ネットワークで接続されます。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

AWS リージョン およびアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

ストレージの耐久性

QLDB ジャーナルストレージは、トランザクションコミット時の複数のアベイラビリティーゾーンへの同期レプリケーション機能を備えています。これにより、ジャーナルストレージのアベイラビリティーゾーン全体に障害が発生しても、データの整合性やアクティブなサービスを維持する能力が損なわれることはありません。さらに、QLDB ジャーナルには、フォールトトレラントストレージへの非同期アーカイブ機能があります。この機能は、複数のアベイラビリティーゾーンで同時にストレージ障害が発生するという可能性が非常に低いイベント時の災害対策をサポートします。

QLDB インデックス付きストレージは、複数のアベイラビリティーゾーンへのレプリケーションによってバックアップされます。これにより、インデックス付きストレージのアベイラビリティーゾーン全体に障害が発生しても、データの整合性やアクティブなサービスを維持する能力が損なわれることはありません。

データ耐久性機能

グローバル AWS インフラストラクチャに加えて、QLDB には、データの耐障害性とバックアップのニーズをサポートするために以下の機能が用意されています。

QLDB サービスの機能

オンデマンドジャーナルエクスポート

QLDB は、オンデマンドジャーナルエクスポート機能を提供します。台帳から Amazon S3 バケットにジャーナルブロックをエクスポートすることで、ジャーナルのコンテンツにアクセスします。このデータは、データ保持、分析、監査などのさまざまな目的に使用できます。詳細については、「Amazon QLDB からのジャーナルデータのエクスポート」を参照してください。

バックアップと復元

エクスポートの自動復元は、現時点ではサポートされていません。エクスポートは、追加のデータ冗長性を定義した頻度で作成する基本的な機能を提供します。ただし、復元シナリオはアプリケーションによって異なります。エクスポートされたレコードは、同じまたは類似の取り込み方法を使用して新しい台帳に書き戻されると想定されます。

QLDB は、現時点では専用のバックアップおよび関連する復元機能を提供していません。

ジャーナルストリーム

QLDB は、連続ジャーナルストリーム機能も提供します。QLDB ジャーナルストリームを Amazon Kinesis ストリーミングプラットフォームと統合して、リアルタイムのジャーナルデー

タを処理できます。詳細については、「Amazon QLDB からのジャーナルデータのストリーミング」を参照してください。

QLDB の設計上の特徴

QLDB は、論理的な破損からの回復性があるように設計されています。QLDB ジャーナルは不変であり、コミットされたすべてのトランザクションがジャーナルに確実に保持されます。さらに、コミットされたドキュメントの変更はすべて記録されます。これにより、台帳データへの意図しない変更をpoint-in-time可視化できます。

QLDB は、現時点では、論理的破損シナリオに対する自動回復機能は提供していません。

Amazon QLDB のインフラストラクチャセキュリティ

マネージドサービスである Amazon QLDB は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティの手順で保護されています。

が AWS 公開した API コールを使用して、ネットワーク経由で QLDB にアクセスします。クライアントで Transport Layer Security (TLS) 1.0 以降がサポートされている必要があります。TLS 1.2 以降が推奨されています。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、IAM プリンシパルに関連付けられているプログラム認証情報を使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

また、QLDB で仮想プライベートクラウド (VPC) エンドポイントを使用することもできます。インターフェイス VPC エンドポイントを使用することで、Amazon VPC リソースは、パブリックインターネットにさらされることなく、プライベート IP アドレスを使用して QLDB にアクセスできるようになります。詳細については、「インターフェイスエンドポイント (AWS PrivateLink) を使用して Amazon QLDB にアクセスします。」を参照してください。

インターフェイスエンドポイント (AWS PrivateLink) を使用して Amazon QLDB にアクセスします。

を使用して AWS PrivateLink 、VPC と Amazon QLDB の間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を使用せずに、VPC 内にあるかのように QLDB にアクセスできます。VPC のインスタンスは、パブリック IP アドレスがなくても QLDB にアクセスできます。

このプライベート接続を確立するには、 AWS PrivateLinkを利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、QLDB 宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については、『AWS PrivateLink ガイド』の「AWS のサービス でアクセスする」を参照してください。

トピック

- QLDB に関する考慮事項
- QLDB 用のインターフェイスエンドポイントの作成
- インターフェイスエンドポイントのエンドポイントポリシーを作成する
- QLDB の インターフェイスエンドポイントの可用性

QLDB に関する考慮事項

QLDB のインターフェイスエンドポイントを設定する前に、「AWS PrivateLink ガイド」の「考慮事項」を確認してください。

> **Note**
>
> QLDB は、インターフェイスエンドポイントを介した QLDB セッショントランザクションデータ API に対する呼び出しのみをサポートします。この API には SendCommandオペレーションのみが含まれます。台帳の STANDARD 権限モードでは、この API の特定の PartiQL アクションの権限を制御できます。

QLDB 用のインターフェイスエンドポイントの作成

Amazon VPC コンソールまたは AWS Command Line Interface () を使用して、QLDB のインターフェイスエンドポイントを作成できますAWS CLI。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成」を参照してください。

以下のサービス名を使用して、QLDB のインターフェイスエンドポイントを作成します。

```
com.amazonaws.region.qldb.session
```

インターフェイスエンドポイントのプライベート DNS を有効にすると、デフォルトのリージョン DNS 名を使用して、QLDB への API リクエストを実行できます。例えば `session.qldb.us-east-1.amazonaws.com` です。

インターフェイスエンドポイントのエンドポイントポリシーを作成する

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーでは、インターフェイスエンドポイント経由での QLDB へのフルアクセスが許可されています。VPC から QLDB への許可されたアクセスをコントロールするには、カスタムエンドポイントポリシーをインターフェイスエンドポイントにアタッチします。

エンドポイントポリシーは、以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、ユーザー、ロール)。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、AWS PrivateLink ガイドのControl access to services using endpoint policies (エンドポイントポリシーを使用してサービスへのアクセスをコントロールする)を参照してください。

また、ユーザー、グループ、またはロールにアタッチされたポリシーの `Condition` フィールドを使用して、指定された インターフェイスエンドポイントからのアクセスのみを許可することもできます。エンドポイントポリシーと IAM ポリシーを一緒に使用すると、指定された台帳の特定の QLDB アクションへのアクセスを、指定されたインターフェイスエンドポイントに制限できます。

エンドポイントポリシーの例: 特定の QLDB 台帳へのアクセスを制限する

以下は、QLDB のカスタムエンドポイントポリシーの例です。このポリシーをインターフェイスエンドポイントにアタッチすると、指定された台帳リソースのすべてのプリンシパルに対し

て、SendCommand アクションと PartiQL 読み取り専用アクションへのアクセス権が許可されます。この例では、台帳が STANDARD 権限モードでなければなりません。

このポリシーを使用するには、例*myExampleLedger*の *us-east-1*、*123456789012*、 をユーザー自身の情報に置き換えます。

```
{
   "Statement": [
      {
         "Sid": "QLDBSendCommandPermission",
         "Principal": "*",
         "Effect": "Allow",
         "Action": "qldb:SendCommand",
         "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
      },
      {
         "Sid": "QLDBPartiQLReadOnlyPermissions",
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "qldb:PartiQLSelect",
            "qldb:PartiQLHistoryFunction"
         ],
         "Resource": [
            "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
            "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
         ]
      }
   ]
}
```

IAM ポリシーの例: 特定のインターフェイスエンドポイントからのみ QLDB 台帳へのアクセスを制限する

以下は、QLDB の IAM アイデンティティベースポリシーの例です。このポリシーをユーザー、ロール、またはグループにアタッチすると、指定されたインターフェイスエンドポイントからのみ台帳リソースへの SendCommand アクセスが許可されます。

このポリシーを使用するには、例の *us-east-1*、*123456789012myExampleLedger*、、*vpce-1a2b3c4d* をユーザー自身の情報に置き換えます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AccessFromSpecificInterfaceEndpoint",
         "Effect": "Deny",
         "Action": "qldb:SendCommand",
         "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger",
         "Condition": {
            "StringNotEquals": {
               "aws:sourceVpce": "vpce-1a2b3c4d"
            }
         }
      }
   ]
}
```

QLDB の インターフェイスエンドポイントの可用性

Amazon QLDB は、QLDB が利用可能なすべての AWS リージョン のポリシーを使用したインターフェイスエンドポイントをサポートしています。利用可能なリージョンの完全なリストについては、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。

Amazon QLDB のトラブルシューティング

次のセクションでは、Amazon QLDB を使用する際に発生する可能性のある一般的なエラーを集約したリストと、それらのトラブルシューティング方法を示します。

IAM アクセスに関するトラブルシューティングガイダンスについては、「Amazon QLDB アイデンティティとアクセスのトラブルシューティング」を参照してください。

PartiQL ステートメントのチューニングのベストプラクティスについては、「クエリパフォーマンスの最適化」を参照してください。

トピック

- QLDB ドライバーを使ったトランザクションの実行
- ジャーナルデータのエクスポート
- ジャーナルデータのストリーミング
- ジャーナルデータの検証

QLDB ドライバーを使ったトランザクションの実行

このセクションでは、Amazon QLDB ドライバーを使用して台帳で PartiQL トランザクションを実行したときに返される一般的な例外について説明します。この機能の詳細については、「ドライバーの開始方法」を参照してください。ドライバーを設定および使用するためのベストプラクティスについては、「ドライバーに関する推奨事項」を参照してください。

それぞれの例外には、特定のエラーメッセージに加え、簡単な説明と考えられる解決方法に関する推奨事項が記載されています。

CapacityExceededException

メッセージ: 処理能力の超過

台帳の処理能力を超えたため、Amazon QLDB はリクエストを拒否しました。QLDB は、サービスのヘルスとパフォーマンスを維持するために、台帳ごとに内部スケーリング制限を適用します。この制限は、各リクエストのワークロードサイズによって異なります。例えば、非インデックス修飾クエリによるテーブルスキャンなど、非効率なデータトランザクションを実行する場合、リクエストのワークロードが増大する可能性があります。

リクエストを再試行する前に、待機することをお勧めします。アプリケーションでこの例外が継続的に発生する場合は、ステートメントを最適化し、台帳に送信するリクエストの割合やボリュームを減らします。ステートメントの最適化の例としては、トランザクションごとに実行するステートメントの数を減らすことや、テーブルインデックスのチューニングなどがあります。ステートメントの最適化方法およびテーブルスキャンの回避方法については、「クエリパフォーマンスの最適化」を参照してください。

最新バージョンの QLDB ドライバーを使用することもお勧めします。このドライバーには、エクスポネンシャルバックオフとジッターを使用するデフォルトの再試行ポリシーがあり、こうした例外発生時に自動的に再試行します。エクスポネンシャルバックオフは、再試行間の待機時間を累進的に長くして、連続的な エラー応答に対処するという概念に基づいています。

InvalidSessionException

メッセージ: トランザクション *transactionId* の有効期限が切れています

トランザクションが最大有効期間を超えました。トランザクションは、コミットされるまでに最大 30 秒間実行できます。このタイムアウト制限を超えると、トランザクションに行ったすべての操作が拒否され、QLDB はセッションを破棄します。この制限は、クライアントがトランザクションを開始し、トランザクションをコミットまたはキャンセルしないことで、セッションがリークするのを防ぎます。

この例外がアプリケーションでよく発生する場合、トランザクションの実行に時間がかかりすぎている可能性があります。トランザクションの実行に 30 秒以上かかる場合は、トランザクションを高速化するようにステートメントを最適化します。ステートメントの最適化の例としては、トランザクションごとに実行するステートメントの数を減らすことや、テーブルインデックスのチューニングなどがあります。詳細については、「クエリパフォーマンスの最適化」を参照してください。

InvalidSessionException

メッセージ: セッション *sessionId* の有効期限が切れています

最大合計存続期間を超えたため、QLDB はセッションを破棄しました。QLDB は、アクティブなトランザクションに関係なく、13 ~ 17 分後にセッションを破棄します。ハードウェア障害、ネットワーク障害、アプリケーションの再起動など、さまざまな理由でセッションが失われたり、障害が生じたりすることがあります。したがって、QLDB はセッションに最大有効期間を適用し、クライアントソフトウェアがセッション障害に対して回復力を持つようにします。

この例外が発生した場合は、新しいセッションを取得し、トランザクションを再試行することをお勧めします。また、最新バージョンの QLDB ドライバーを使用することをお勧めします。このドライバーは、アプリケーションに代わってセッションプールとそのヘルスを管理します。

InvalidSessionException

メッセージ: そのようなセッションはありません

クライアントは、存在しないセッションを使用して QLDB でトランザクションを実行しようとしました。以前に存在していたセッションをクライアントが使用していると仮定すると、次のいずれかが原因でセッションは存在しなくなる可能性があります。

- セッションが内部サーバーの障害 (つまり、HTTP 応答コード 500 のエラー) に関与している場合、QLDB は、カスタマーが不確実な状態のセッションでトランザクションを実行することを許可するのではなく、セッションを完全に破棄することを選択することがあります。その後、そのセッションでの再試行は、このエラーで失敗します。
- 期限切れのセッションは、最終的に QLDB に破棄されます。その後、そのセッションの使用を継続しようとすると、最初の `InvalidSessionException` エラーではなく、このエラーが発生します。

この例外が発生した場合は、新しいセッションを取得し、トランザクションを再試行することをお勧めします。また、最新バージョンの QLDB ドライバーを使用することをお勧めします。このドライバーは、アプリケーションに代わってセッションプールとそのヘルスを管理します。

RateExceededException

メッセージ: レートを超えました

QLDB は、呼び出し元のアイデンティティに基づいてクライアントをスロットルしました。QLDB は、トークンバケットスロットリングアルゴリズムを使用して、リージョンごと、アカウント単位でスロットリングを適用します。これは、サービスのパフォーマンスを向上し、QLDB のすべてのお客様に平等にご利用いただくために行われるものです。たとえば、`StartSessionRequest` オペレーションを使用して多数の同時セッションを取得しようとすると、スロットリングが発生する可能性があります。

アプリケーションの正常性を維持し、さらなるスロットリングを軽減するために、エクスポネンシャルバックオフおよびジッターを使用してこの例外を再試行できます。エクスポネンシャルバックオフは、再試行間の待機時間を累進的に長くして、連続的なエラー応答に対処するという概念に基づいています。最新バージョンの QLDB ドライバーを使用することをお勧めします。このドライバーには、エクスポネンシャルバックオフとジッターを使用するデフォルトの再試行ポリシーがあり、こうした例外発生時に自動的に再試行します。

最新バージョンの QLDB ドライバーは、アプリケーションが StartSessionRequest コールのために QLDB によって継続的にスロットリングしている場合に役立ちます。このドライバーは、トランザクション間で再利用されるセッションのプールを維持します。これにより、アプリケーションが行う StartSessionRequest コールの回数を減らすことができます。API スロットリング制限の引き上げをリクエストするには、AWS Support センターにお問い合わせください。

LimitExceededException

メッセージ: セッション制限を超えました

台帳が、アクティブなセッション数のクォータ (制限とも呼ばれる) を超えました。このクォータは Amazon QLDB でのクォータと制限 で定義されています。台帳のアクティブセッション数は結果的に一貫しており、常時クォータ付近で実行している台帳の場合、この例外が定期的に表示される場合があります。

アプリケーションの正常性を維持するために、この例外を再試行することをお勧めします。この例外を回避するには、すべてのクライアントで単一台帳で使用する同時セッションが 1,500 を超えないようにしてください。例えば、Java 用 Amazon QLDB ドライバーの maxConcurrentTransactions メソッドを使用して、ドライバーインスタンスで使用可能なセッションの最大数を設定できます。

QldbClientException

メッセージ: ストリームされた結果は、親トランザクションが開いている場合にのみ有効です

トランザクションは閉じられており、QLDB から結果を取得するために使用することはできません。トランザクションは、コミットまたはキャンセルされると終了します。

この例外は、クライアントが Transaction オブジェクトを直接操作していて、トランザクションのコミットまたはキャンセル後に QLDB から結果を取得しようとした場合に発生します。この問題を軽減するには、クライアントはトランザクションを閉じる前にデータを読み取る必要があります。

ジャーナルデータのエクスポート

このセクションでは、台帳から Amazon S3 バケットにジャーナルデータを エクスポートするときに QLDB が返す可能性のある一般的な例外について説明します。この機能の詳細については、「Amazon QLDB からのジャーナルデータのエクスポート」を参照してください。

それぞれの例外には、特定のエラーメッセージに加え、簡単な説明と考えられる解決方法に関する推奨事項が記載されています。

AccessDeniedException

メッセージ: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN* (ユーザー: userARN にはリソース roleARN に対して iam:PassRole を実行する許可がありません)

IAM ロールを QLDB サービスに渡す許可がありません。QLDB には、すべてのジャーナルエクスポートリクエストに対応するロールが必要であり、このロールを QLDB に渡す許可が必要です。このロールにより、指定した Amazon S3 バケットへの書き込みアクセス許可が QLDB に付与されます。

指定した IAM ロールリソースに対して `PassRole` API オペレーションを実行する許可を付与する IAM ポリシーを QLDB サービス (`qldb.amazonaws.com`) に定義していることを確認します。ポリシーの例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

IllegalArgumentException

メッセージ: QLDB encountered an error validating S3 configuration (QLDB で S3 設定の検証中にエラーが発生しました): *errorCode errorMessage*

このエラーの原因として考えられるのは、指定したバケットが Simple Storage Service (Amazon S3) に存在しないことです。または、指定した Simple Storage Service (Amazon S3) バケットにオブジェクトを書き込むための権限が QLDB にないと考えられます。

エクスポートジョブリクエストで指定した S3 バケット名が正しいか確認します。バケットの命名規則の詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの制約と制限」を参照してください。

また、指定したバケットに対する `PutObject` および `PutObjectAcl` アクセス許可を QLDB サービス (`qldb.amazonaws.com`) に付与するポリシーを定義していることを確認します。詳細については、「エクスポートアクセス許可」を参照してください。

IllegalArgumentException

メッセージ: Unexpected response from Amazon S3 while validating the S3 configuration. (S3 設定の検証中に、Amazon S3 から不測のレスポンスがありました。) S3 からのレスポンス: *errorCode errorMessage*

指定した S3 バケットにジャーナルエクスポートデータを書き込もうとしましたが、Simple Storage Service (Amazon S3) エラーレスポンスがあり失敗しました。考えられる原因の詳細につ

いては、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 のトラブルシューティング」を参照してください。

IllegalArgumentException

メッセージ: Amazon S3 bucket prefix must not exceed 128 characters (Amazon S3 バケットプレフィックスは 128 文字以下でなければなりません)

ジャーナルエクスポートのリクエストで指定したプレフィックスが 128 文字を超えています。

IllegalArgumentException

メッセージ: Start date must not be greater than end date (開始日は終了日よりも前の日付でなければなりません)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、ISO 8601 の日時形式、協定世界時 (UTC) にしてください。

IllegalArgumentException

メッセージ: End date cannot be in future (終了日を現在よりも後の日付にすることはできません)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、`ISO 8601` の日時形式、UTC にしてください。

IllegalArgumentException

メッセージ: The supplied object encryption setting (S3EncryptionConfiguration) is not compatible with an AWS Key Management Service (AWS KMS) key

`ObjectEncryptionType` が `NO_ENCRYPTION` または `SSE_S3` の `KMSKeyArn` が指定されています。`SSE_KMS` のオブジェクトの暗号化タイプに対してのみ、カスタマーマネージド AWS KMS key を指定できます。Simple Storage Service (Amazon S3) のサーバー側の暗号化オプションの詳細については、「Amazon S3 デベロッパーガイド」の「サーバー側の暗号化を使用したデータの保護」を参照してください。

LimitExceededException

メッセージ: Exceeded the limit of 2 concurrently running Journal export jobs (ジャーナルエクスポートジョブの同時実行の上限数 2 を超えています)

QLDB では、デフォルトで、ジャーナルエクスポートジョブの同時実行は最大 2 件までという制限が適用されます。

ジャーナルデータのストリーミング

このセクションでは、台帳から Amazon Kinesis Data Streams へのジャーナルデータのストリーミング時に QLDB から返される可能性のある例外について説明します。この機能の詳細については、「Amazon QLDB からのジャーナルデータのストリーミング」を参照してください。

それぞれの例外には、特定のエラーメッセージに加え、簡単な説明と考えられる解決方法に関する推奨事項が記載されています。

AccessDeniedException

メッセージ: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN* (ユーザー: userARN にはリソース roleARN に対して iam:PassRole を実行する許可がありません)

IAM ロールを QLDB サービスに渡す許可がありません。QLDB には、すべてのジャーナルストリーミングリクエストに対応するロールが必要であり、このロールを QLDB に渡す許可が必要です。このロールは、指定した Amazon Kinesis Data Streams リソースでの書き込み許可を QLDB に付与します。

指定した IAM ロールリソースに対して `PassRole` API オペレーションを実行する許可を付与する IAM ポリシーを QLDB サービス (`qldb.amazonaws.com`) に定義していることを確認します。ポリシーの例については、「Amazon QLDB のアイデンティティベースのポリシー例」を参照してください。

IllegalArgumentException

メッセージ: QLDB encountered an error validating Kinesis Data Streams (Kinesis Data Streams の検証中にエラーが発生しました): Kinesis からのレスポンス: *errorCode errorMessage*

このエラーの原因として考えられるのは、存在しない Kinesis Data Streams リソースが指定されていることです。または、QLDB には、指定した Kinesis データストリームにデータレコードを書き込むための十分な許可がありません。

ストリーミングリクエストに指定した Kinesis データストリームが正しいことを確認します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「データストリームの作成および更新」を参照してください。

以下のアクションに対して QLDB サービス (`qldb.amazonaws.com`) 許可を付与する、指定した Kinesis データストリーム用のポリシーを定義していることも確認します。詳細については、「ストリームアクセス許可」を参照してください。

- `kinesis:PutRecord`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`
- `kinesis:ListShards`

IllegalArgumentException

メッセージ: Unexpected response from Kinesis Data Streams while validating the Kinesis configuration. Response from Kinesis: *`errorCode errorMessage`* (Kinesis 設定の検証中に、Kinesis Data Streams から予期しないレスポンスがありました。Kinesis からのレスポンス: errorCode errorMessage)

指定した Kinesis データストリームにデータレコードを書き込もうとしましたが、Kinesis エラーレスポンスがあり失敗しました。考えられる原因については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Amazon Kinesis Data Streams プロデューサーのトラブルシューティング」を参照してください。

IllegalArgumentException

メッセージ: Start date must not be greater than end date. (開始日は終了日よりも前の日付でなければなりません。)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、ISO 8601 の日時形式、協定世界時 (UTC) にしてください。

IllegalArgumentException

メッセージ: Start date cannot be in the future (開始日を現在よりも後の日付にすることはできません)

`InclusiveStartTime` と `ExclusiveEndTime` を両方とも、`ISO 8601` の日時形式、UTC にしてください。

LimitExceededException

メッセージ: Exceeded the limit of 5 concurrently running Journal streams to Kinesis Data Streams (Kinesis Data Streams への同時実行ジャーナルストリームの 5 つの制限を超えました)

QLDB では、一度に実行できるジャーナルストリーミングのデフォルト上限、最大 5 件が適用されます。

ジャーナルデータの検証

このセクションでは、台帳でジャーナルデータを検証するときに QLDB が返す可能性のある一般的な例外を示します。この機能の詳細については、「Amazon QLDB でのデータ検証」を参照してください。

各例外には、固有のエラーメッセージ、エラーをスローする可能性のある API オペレーション、簡単な説明、および考えられる解決策の案が含まれています。

IllegalArgumentException

メッセージ: 提供された Ion 値は無効なため解析できません。

API オペレーション: `GetDigest`, `GetBlock`, `GetRevision`

リクエストを再試行する前に、有効な Amazon Ion 値が指定されているか確認してください。

IllegalArgumentException

メッセージ: 指定されたブロックアドレスは無効です。

API オペレーション: `GetDigest`, `GetBlock`, `GetRevision`

リクエストを再試行する前に、有効なブロックアドレスが指定されていることを確認してください。ブロックアドレスは、`strandId` と `sequenceNo` という 2 つのフィールドを含む Amazon Ion 構造です。

例: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`

IllegalArgumentException

メッセージ: The sequence number of the provided digest tip address is beyond the strand's latest committed record. (指定されたダイジェストティップアドレスのシーケンス番号が、ストランドの最新のコミット済みレコードを超えています。)

API オペレーション: `GetDigest`, `GetBlock`, `GetRevision`

指定するダイジェストティップアドレスのシーケンス番号は、ジャーナルストランドの最新のコミット済みレコードのシーケンス番号以下でなければなりません。リクエストを再試行する前に、ダイジェストティップアドレスに有効なシーケンス番号を指定していることを確認してください。

IllegalArgumentException

メッセージ: The Strand ID of the provided block address is not valid. (指定されたブロックアドレスのストランド ID が無効です。)

API オペレーション: `GetDigest, GetBlock, GetRevision`

指定するブロックアドレスのストランド ID は、ジャーナルのストランド ID に一致するものでなければなりません。リクエストを再試行する前に、ブロックアドレスに有効なストランド ID を指定していることを確認してください。

IllegalArgumentException

メッセージ: The sequence number of the provided block address is beyond the strand's latest committed record. (指定されたブロックアドレスのシーケンス番号が、ストランドの最新のコミット済みレコードを超えています。)

API オペレーション: `GetBlock, GetRevision`

指定するブロックアドレスのシーケンス番号は、ストランドの最新のコミット済みレコードのシーケンス番号以下でなければなりません。リクエストを再試行する前に、有効なシーケンス番号を持つブロックアドレスが指定されているか確認してください。

IllegalArgumentException

メッセージ: 指定されたブロックアドレスのストランド ID は、指定されたダイジェストティップアドレスのストランド ID と一致するものでなければなりません。

API オペレーション: `GetBlock, GetRevision`

ドキュメントリビジョンまたはブロックは、指定するダイジェストと同じジャーナルストランドに存在する場合に限り検証できます。

IllegalArgumentException

メッセージ: 指定されたブロックアドレスのシーケンス番号は、指定されたダイジェストティップアドレスのシーケンス番号以下でなければなりません。

API オペレーション: `GetBlock, GetRevision`

ドキュメントリビジョンまたはブロックは、指定されたダイジェストでカバーされている場合に限り検証できます。言い換えると、ダイジェストティップアドレスより前のジャーナルにコミットされている場合に限り検証できます。

IllegalArgumentException

メッセージ: 指定されたドキュメント ID が、指定されたブロックアドレスのブロックに存在しません。

API オペレーション: `GetRevision`

指定するドキュメント ID は、指定するブロック内に存在するものでなければなりません。リクエストを再試行する前に、これら 2 つのパラメータが一貫しているか確認してください。

Amazon QLDB の PartiQL リファレンス

Amazon QLDB は、PartiQL クエリ言語のサブセットをサポートしています。以下のトピックでは、PartiQL の QLDB 実装について説明します。

> **Note**
>
> - QLDB は、すべての PartiQL オペレーションをサポートしているわけではありません。
> - QLDB のすべての PartiQL ステートメントは、Amazon QLDB でのクォータと制限 で定義されているトランザクション制限の対象となります。
> - このリファレンスでは、QLDB コンソールまたは QLDB シェルで手動で実行する PartiQL ステートメントの基本的な構文と使用例を示します。QLDB ドライバーを使用して同様のステートメントをプログラムで実行する方法を示すコード例については、「ドライバーの開始方法」のチュートリアルを参照してください。

トピック

PartiQL とは何ですか?

PartiQL は、構造化データ、半構造化データ、ネストされたデータを含む複数のデータストア間で、SQL 互換のクエリアクセスを提供します。PartiQL は、Amazon 内で広く使用されており、現在、QLDB を含む多くの AWS のサービス の一部として利用できます。

PartiQL の仕様とコアクエリ言語のチュートリアルについては、PartiQL ドキュメントを参照してください。

PartiQL は、SQL-92 を拡張することで、Amazon Ion データ形式のドキュメントをサポートしています。Amazon Ion の詳細については、「Amazon QLDB の Amazon Ion データ形式リファレンス」を参照してください。

Amazon QLDB の PartiQL

QLDB で PartiQL クエリを実行するには、次のいずれかを使用します。

- QLDB の AWS Management Console の PartiQL エディタ
- コマンドライン QLDB シェル
- クエリをプログラムで実行する、AWS 提供の QLDB ドライバー

これらの方法を使用して QLDB にアクセスする方法については、「Amazon QLDB へのアクセス」を参照してください。

特定のテーブルで各 PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

QLDB における PartiQL のクイックヒント

以下は、QLDB で PartiQL を使用するためのヒントとベストプラクティスの簡単な要約です。

- 同時実行性とトランザクション制限を理解する - SELECT クエリを含むすべてのステートメントはオプティミスティック同時実行制御 (OCC、Optimistic Concurrency Control) 競合およびトランザクション制限 (30 秒のトランザクションタイムアウトなど) の対象になります。
- インデックスの使用 - 高基数インデックスを使用し、ターゲットとなるクエリを実行して、ステートメントを最適化し、すべてのテーブルスキャンを回避します。詳細については、「クエリパフォーマンスの最適化」を参照してください。

- 等価述語の使用 - インデックス付きルックアップには等価演算子 (= または IN) が必要です。不等価演算子 (<、>、LIKE、BETWEEN) はインデックス付きルックアップの対象にならず、すべてのテーブルスキャンが実行されます。
- 内部結合のみを使用する - QLDB は現在、内部結合のみをサポートしています。ベストプラクティスとして、結合するテーブルごとにインデックス付けされたフィールドで結合します。結合基準と等価述語の両方に高基数インデックスを選択します。

Amazon QLDB の PartiQL リファレンスの規則

このセクションでは、Amazon QLDB の PartiQL リファレンスに記載されている PartiQL のコマンド、関数および式の構文を記述する際の規則について説明します。これらの規則を PartiQL クエリ言語自体の構文とセマンティクスと混同しないでください。

文字	説明
CAPS	大文字の単語はキーワードです。
[ ]	角括弧はオプションの引数または句を表します。角括弧に複数の引数が含まれる場合は、任意の個数の引数を選択できることを示します。また、複数行に角括弧で囲まれた引数がある場合、QLDB パーサーは、引数が構文で示される順番どおりに出現するものと想定します。
|	縦線は、どちらかの引数を選択できることを示します。
red italics	赤色のイタリック体の単語は、プレースホルダを示します。赤色のイタリック体の単語に代えて適切な値を挿入します。
. . .	省略符号は、先行する要素の繰り返しが可能であることを示します。
'	一重引用符に囲まれた値は、一重引用符の入力が必要であることを示します。PartiQL では、一重引用符は Amazon Ion 構造内の文字列値、またはフィールド名を表します。
"	二重引用符に囲まれた値は、二重引用符の入力が必要であることを示します。PartiQL では、二重引用符は引用符で囲まれた識別子を表します。
`	バックティックに囲まれた値は、バックティックの入力が必要であることを示します。PartiQL では、バックティックは Ion リテラル値を表します。

Amazon QLDB のデータ型

Amazon QLDB は Amazon Ion 形式でドキュメントを保存します。Amazon Ion は、データシリアル化フォーマット (テキスト形式およびバイナリエンコード形式の両方で表記される) で、JSON のスーパーセットです。次の表に、QLDB ドキュメントで使用できる Ion データ型を示します。

データ型	説明
`null`	汎用 null 値
`bool`	ブール値
`int`	任意の大きさの符号付き整数
`decimal`	任意精度の 10 進数エンコード実数
`float`	バイナリエンコード浮動小数点数 (64 ビット IEEE)
`timestamp`	任意精度の日付/時刻/タイムゾーン
`string`	Unicode 文字リテラル
`symbol`	Unicode 記号のアトム (識別子)
`blob`	ユーザー定義エンコードのバイナリデータ
`clob`	ユーザー定義エンコードのテキストデータ
`struct`	名前と値のペアの順序なしコレクション
`list`	異種値の順序付きコレクション

Ion の主要なデータ型、そのすべての説明、値の形式の詳細のリストについては、Amazon の GitHub サイトにある Ion の仕様書を参照してください。

Amazon QLDB のドキュメント

Amazon QLDB は、データレコードをドキュメントとして保存します。このドキュメントは、テーブルに挿入される Amazon Ion `struct` オブジェクトです。Ion の仕様については、Amazon Ion の GitHub サイトを参照してください。

トピック

- Ion ドキュメントの構造
- PartiQL と Ion の型マッピング
- ドキュメント ID

Ion ドキュメントの構造

JSON と同様に、QLDB ドキュメントは、以下の構造の名前と値のペアで構成されます。

```
{
  name1: value1,
  name2: value2,
  name3: value3,
  ...
  nameN: valueN
}
```

名前は記号トークンで、値は無制限です。名前と値の各ペアはフィールドと呼ばれます。フィールドの値は、コンテナ型 (ネストされた構造、リスト、構造のリスト) を含む、任意の Ion データ型 にすることができます。

また JSON と同様に、`struct` は中括弧 (`{...}`) で、`list` は角括弧 (`[...]`) で表されます。次の例は、さまざまな型の値を含む Amazon QLDB コンソールの使用開始方法 のサンプルデータのドキュメントです。

```
{
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    PendingPenaltyTicketAmount: 90.25,
    ValidFrom: 2017-08-21T,
    ValidTo: 2020-05-11T,
```

```
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    }
}
```

> **Important**
>
> Ion では、二重引用符は文字列値を表し、引用符で囲まれていないシンボルはフィールド名を表します。ただし、PartiQL では、一重引用符は文字列とフィールド名の両方を表します。
> 構文のこの違いにより、PartiQL クエリ言語で SQL 互換性を維持し、Amazon Ion データ形式で JSON 互換性を維持できます。QLDB での PartiQL の構文とセマンティクスの詳細については、「PartiQL での Ion のクエリ」を参照してください。

PartiQL と Ion の型マッピング

QLDB では、PartiQL は SQL の型システムを継承して、Ion データモデルに対応しています。このマッピングについて以下に説明します。

- SQL スカラー型は Ion で以下の例のようにマッピングされます。例:
  - `CHAR` と `VARCHAR` は、Ion の `string` 型にマッピングされる Unicode シーケンスです。
  - `NUMBER` は、Ion `decimal` タイプにマッピングされます。
- Ion の `struct` 型は、テーブルの行を従来表す SQL タプルと同等です。
  - ただし、オープンコンテンツでスキーマなしの場合、本来順序付きの SQL タプルに依存するクエリ (`SELECT *` の順序付きの出力など) はサポートされません。
- `NULL` に加えて、PartiQL には `MISSING` 型があります。これは `NULL` の特殊化であり、欠けているフィールドあることを示します。この型が必要なのは、Ion の `struct` フィールドがスパースであるためです。

ドキュメント ID

QLDB では、テーブルに挿入する各ドキュメントにドキュメント ID を割り当てます。システムによって割り当てられた ID はすべて汎用一意 ID (UUID、Universally Unique Identifier) であり、それぞれ Base62 エンコード文字列で表されます (例: `3Qv67yjXEwB9SjmvkuG6Cp`)。詳細については、「Amazon QLDB で割り当てられる一意の ID」を参照してください。

ドキュメントの各リビジョンは、ドキュメント ID と 0 から始まるバージョン番号を組み合わせてそれぞれ識別されます。

ドキュメント ID およびバージョンフィールドはドキュメントのメタデータに含まれており、コミット済みビュー (テーブルのシステム定義のビュー) でクエリを実行できます。QLDB におけるビューの詳細については、「重要な概念」を参照してください。メタデータの詳細については、「ドキュメントのメタデータのクエリの実行」を参照してください。

Amazon QLDB での PartiQL による Ion のクエリ

Amazon QLDB でデータをクエリする場合、ステートメントは PartiQL 形式で記述されますが、結果は Amazon Ion で返されます。PartiQL は SQL と互換性を持つようになっていますが、Ion は JSON の拡張機能です。そのため、クエリステートメントでのデータの表記方法と、クエリ結果の表示方法に構文上の違いが生じます。

このセクションでは、QLDB コンソールまたは QLDB シェルを使用して PartiQL ステートメントを手動で実行するための基本的な構文とセマンティクスについて説明します。

> **Tip**
>
> PartiQL クエリをプログラムで実行する場合、ベストプラクティスはパラメータ化されたステートメントを使用することです。ステートメントで疑問符 (?) をバインド変数プレースホルダーとして使用して、これらの構文ルールを回避できます。この方法は、よりセキュアで効率的でもあります。
> 詳細については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。
>
> - Java: クイックスタートチュートリアル | クックブックリファレンス
> - .NET: クイックスタートチュートリアル | クックブックリファレンス
> - Go: クイックスタートチュートリアル | クックブックリファレンス
> - Node.js: クイックスタートチュートリアル | クックブックリファレンス
> - Python: クイックスタートチュートリアル | クックブックリファレンス

トピック

- 構文とセマンティクス
- バックティック表記
- パスナビゲーション

- エイリアス定義
- PartiQL の仕様

構文とセマンティクス

QLDB コンソールまたは QLDB シェルを使用して Ion データをクエリする場合の、PartiQL の基本的なクエリ構文とセマンティクスは以下のとおりです。

大文字と小文字の区別

フィールド名、テーブル名、台帳名を含むすべての QLDB システムオブジェクト名では、大文字と小文字が区別されます。

文字列値

Ion では、二重引用符 ("...") は文字列を表します。

PartiQL では、一重引用符 ('...') は文字列を表します。

記号と識別子

Ion では、一重引用符 ('...') は記号を表します。Ion のシンボルのサブセットは識別子と呼ばれ、引用符で囲まれていないテキストで表されます。

PartiQL では、二重引用符 ("...") は、引用符で囲まれた PartiQL 識別子 (テーブル名として使用される予約語など) を表します。引用符で囲まれていないテキストは、通常の PartiQL 識別子 (予約語ではないテーブル名など) を表します。

Ion リテラル

すべての Ion リテラル値は、PartiQL ステートメントでバックティック (`...`) で表すことができます。

フィールド名

Ion フィールド名は、大文字と小文字が区別される記号です。PartiQL を使用すると、DML ステートメントでフィールド名を一重引用符で表すことができます。これは、PartiQL の `cast` 関数を使用して記号を定義する場合の省略形です。また、バックティックを使用してリテラルの Ion 記号を示すよりも直感的です。

リテラル

PartiQL クエリ言語のリテラルは、次のように Ion データ型に対応します。

スカラー

「PartiQL と Ion の型マッピング」セクションで説明したように、該当する場合は SQL 構文に従います。例:

- `5`
- `'foo'`
- `null`

構造体

多くの形式や他のデータモデルでは、タプルまたはオブジェクトとも呼ばれます。

コンマで区切られた `struct` 要素を中括弧 (`{...}`) で示します。

- `{ 'id' : 3, 'arr': [1, 2] }`

リスト

アレイとも呼ばれます。

カンマで区切られたリスト要素を持つ角括弧 (`[...]`) で示されます。

- `[ 1, 'foo' ]`

バッグ

PartiQL の順序なしコレクション。

カンマで区切られたバッグ要素を持つ二重山括弧 (`<<...>>`) で表されます。QLDB では、テーブルはバッグと考えることができます。ただし、バッグをテーブルのドキュメント内にネストすることはできません。

- `<< 1, 'foo' >>`

例

以下は、Ion のさまざまな型で使用できる INSERT ステートメント用構文の一例です。

```
INSERT INTO VehicleRegistration VALUE
{
    'VIN' : 'KM8SRDHF6EU074761', --string
    'RegNum' : 1722, --integer
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75, --decimal
```

```
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' }
        ]
    },
    'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate' : `2020-06-25T`
}
```

バックティック表記

PartiQL はすべての Ion データ型を完全に対応しているため、バックティックを使用せずにステートメントを記述できます。ただし、この Ion リテラル構文によって、ステートメントがより明確で簡潔になる場合があります。

たとえば、Ion タイムスタンプと記号の値を含むドキュメントを挿入するには、純粋に PartiQL 構文のみを使用して次のステートメントを記述できます。

```
INSERT INTO myTable VALUE
{
    'myTimestamp': to_timestamp('2019-09-04T'),
    'mySymbol': cast('foo' as symbol)
}
```

これはかなり冗長であるため、代わりにバックティックを使用してステートメントを簡素化できます。

```
INSERT INTO myTable VALUE
{
    'myTimestamp': `2019-09-04T`,
    'mySymbol': `foo`
}
```

構造全体をバックティックで囲むことで、さらにいくつかのキーストロークを保存することもできます。

```
INSERT INTO myTable VALUE
`{
    myTimestamp: 2019-09-04T,
```

```
    mySymbol: foo
}`
```

> **Important**
>
> 文字列と記号は、PartiQL の異なるクラスです。つまり、同じテキストがあっても、それらは等しくありません。たとえば、次の PartiQL 式は異なる Ion 値に評価されます。
>
> ```
> 'foo'
> ```
>
> ```
> `foo`
> ```

パスナビゲーション

データ操作言語 (DML) またはクエリステートメントを記述するときは、パスステップを使用してネストされた構造内のフィールドにアクセスできます。PartiQL は、親構造のフィールド名にアクセスするためのドット表記をサポートしています。以下の例では、親 `Vehicle` の `Model` フィールドにアクセスしています。

```
Vehicle.Model
```

リストの特定の要素にアクセスするには、角かっこ演算子を使用してゼロから始まる序数を指定します。以下の例では、序数 2 を使用して `SecondaryOwners` の要素にアクセスしています。つまり、これはリストの 3 番目の要素です。

```
SecondaryOwners[2]
```

エイリアス定義

QLDB はオープンコンテンツとスキーマをサポートしています。したがって、ステートメント内の特定のフィールドにアクセスするときに、想定どおりの結果を得るための最善の方法は、エイリアスを使用することです。たとえば、明示的なエイリアスを指定しない場合、システムによって `FROM` ソースに対して暗黙的なエイリアスが生成されます。

```
SELECT VIN FROM Vehicle
--is rewritten to
```

```
SELECT Vehicle.VIN FROM Vehicle AS Vehicle
```

しかし、フィールド名が競合する可能性があり、結果は予期できません。VIN という名前の別のフィールドが、ドキュメント内のネストされた構造にある場合、VIN の値がこのクエリによって返されて、予期しない結果になることがあります。ベストプラクティスとして、代わりに以下のステートメントを記述します。このクエリでは、v を、Vehicle テーブルに該当するエイリアスとして宣言しています。AS キーワードはオプションです。

```
SELECT v.VIN FROM Vehicle [ AS ] v
```

エイリアス定義は、ドキュメント内のネストされたコレクションにパス指定するときに特に便利です。たとえば、以下のステートメントは o を、コレクション VehicleRegistration.Owners に該当するエイリアスとして宣言しています。

```
SELECT o.SecondaryOwners
FROM VehicleRegistration AS r, @r.Owners AS o
```

ここで @ 文字は構文的に省略可能です。ただしこの文字は、Owners という名前の別のコレクション (存在する場合) ではなく、VehicleRegistration 内の Owners 構造を必要とすることを明示的に示しています。

PartiQL の仕様

PartiQL クエリ言語の詳細については、「PartiQL 仕様」を参照してください。

Amazon QLDB の PartiQL コマンド

PartiQL は、SQL-92 を拡張することで、Amazon Ion データ形式のドキュメントをサポートしています。Amazon QLDB は、以下の PartiQL コマンドをサポートしています。

特定のテーブルで各 PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

> **Note**
>
> - QLDB はすべての PartiQL コマンドをサポートしているわけではありません。
> - QLDB のすべての PartiQL ステートメントは、Amazon QLDB でのクォータと制限 で定義されているトランザクション制限の対象となります。

- このリファレンスでは、QLDB コンソールまたは QLDB シェルで手動で実行する PartiQL ステートメントの基本的な構文と使用例を示します。サポートされているプログラミング言語を使用して同様のステートメントを実行する方法を示すコード例については、「ドライバーの開始方法」のチュートリアルを参照してください。

DDL ステートメント (データ定義言語)

データ定義言語 (DDL) は、テーブルやインデックスなどのデータベースオブジェクトを管理するために使用する PartiQL ステートメントのセットです。これらのオブジェクトを作成および削除するには、DDL を使用します。

- CREATE INDEX
- CREATE TABLE
- DROP INDEX
- DROP TABLE
- テーブルの削除の取り消し

DML ステートメント (データ操作言語)

データ操作言語 (DML、Data Manipulation Language) は、QLDB テーブル内のデータを管理するために使用する PartiQL ステートメントのセットです。DML ステートメントを使用して、テーブル内のデータを追加、変更、または削除します。

次の DML ステートメントとクエリ言語ステートメントがサポートされています。

- DELETE
- FROM (INSERT、REMOVE、または SET)
- INSERT
- SELECT
- UPDATE

Amazon QLDB のインデックス作成コマンド

Amazon QLDB では、テーブルのドキュメントフィールドにインデックスを作成するには、`CREATE INDEX` コマンドを使用します。

特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

> **Important**
>
> ドキュメントを効率的に検索するには、インデックスが必要です。インデックスがないと、QLDB はドキュメントを読み取るときにテーブルスキャンを実行する必要があります。これにより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
>
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子 (= または IN) を使用する WHERE 述語句でステートメントを実行する必要があります。詳細については、「クエリパフォーマンスの最適化」を参照してください。

インデックスを作成する際には、以下の制約があることに注意してください。

- インデックスは 1 つのトップレベルフィールドでのみ作成できます。複合、ネスト、一意、および関数ベースのインデックスはサポートされていません。
- 任意の Ion データ型 (`list`、`struct` など) でインデックスを作成できます。ただし、Ion のデータ型に関わらず、Ion 値全体の等価によってインデックス付けされたルックアップのみを実行できます。例えば、`list` 型をインデックスとして使用すると、リスト内の 1 つの項目でインデックス付けされたルックアップは実行できません。
- クエリのパフォーマンスは、等価述語 (`WHERE indexedField = 123`、`WHERE indexedField IN (456, 789)` など) を使用する場合にのみ向上します。

  QLDB では、クエリの述語で不等式はサポートされていません。そのため、範囲でフィルタリングされるスキャンは実装されていません。
- インデックス付きフィールドの名前は大文字と小文字の区別があり 128 文字以下で指定します。
- QLDB でのインデックス作成は非同期です。空でないテーブルでのインデックスの作成を完了するのにかかる時間は、テーブルサイズによって異なります。詳細については、「インデックスの管理」を参照してください。

トピック

- 構文
- パラメータ
- 戻り値

- 例
- ドライバーを使用したプログラムでの実行

構文

```
CREATE INDEX ON table_name (field)
```

パラメータ

table_name

インデックスを作成するテーブルの名前。このテーブルは既存であることが必要です。

テーブル名では、大文字と小文字が区別されます。

field

インデックスを作成するドキュメントのフィールド名。このフィールドは最上位の属性であることが必要です。

インデックス付きフィールドの名前は大文字と小文字の区別があり 128 文字以下で指定します。

任意の Amazon Ion データ型 (`list`、`struct` など) でインデックスを作成できます。ただし、Ion のデータ型に関わらず、Ion 値全体の等価によってインデックス付けされたルックアップのみを実行できます。例えば、`list` 型をインデックスとして使用すると、リスト内の 1 つの項目でインデックス付けされたルックアップは実行できません。

戻り値

`tableId` - インデックスを作成したテーブルの一意の ID。

例

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

ドライバーを使用したプログラムでの実行

QLDB ドライバーを使用してこのステートメントをプログラムで実行する方法については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。

- Java: クイックスタートチュートリアル | クックブックリファレンス
- .NET: クイックスタートチュートリアル | クックブックリファレンス
- Go: クイックスタートチュートリアル | クックブックリファレンス
- Node.js: クイックスタートチュートリアル | クックブックリファレンス
- Python: クイックスタートチュートリアル | クックブックリファレンス

Amazon QLDB の CREATE TABLE コマンド

Amazon QLDB では、新しいテーブルを作成するには、`CREATE TABLE` コマンドを使用します。

テーブルには名前空間のない単純な名前を付けます。QLDB ではオープンコンテンツがサポートされており、スキーマは適用されないため、テーブル作成時に属性やデータ型を定義しません。

> **Note**
>
> 台帳でこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

構文

```
CREATE TABLE table_name [ WITH (aws_tags = `{'key': 'value'}`) ]
```

パラメータ

table_name

作成するテーブルの一意の名前。同じ名前のアクティブなテーブルがまだ存在していない必要があります。以下は命名に関する制約です。

- 1～128 個の英数字またはアンダースコア文字のみを使用する必要があります。
- 1 字目は文字またはアンダースコアでなければなりません。
- 残りの文字は英数字とアンダースコアの任意の組み合わせにできます。
- 大文字と小文字を区別します。
- QLDB PartiQL の予約語は使用できません。

'***key***': '***value***'

(オプション) 作成時にテーブルリソースにアタッチするタグ。各タグは、キーと値のペアとして定義されます。この場合、キーと値はそれぞれ一重引用符で示されます。キーと値のペアはそれぞれ、バックティックで示される Amazon Ion 構造内で定義されます。

作成時のテーブルのタグ付けは、現在 STANDARD 許可モードの台帳のみでサポートされています。

戻り値

`tableId` - 作成したテーブルの一意の ID。

作成時のテーブルのタグ付け

> **Note**
>
> 作成時のテーブルのタグ付けは、現在 STANDARD 許可モードの台帳のみでサポートされています。

必要に応じて、CREATE TABLE ステートメントでタグを指定して、テーブルリソースをタグ付けできます。タグの詳細については、「Amazon QLDB リソースのタグ付け」を参照してください。次の例では、`Vehicle` という名前のテーブルをタグ `environment=production` を付けて作成します。

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

作成時にテーブルにタグを付けるには、qldb:PartiQLCreateTable および qldb:TagResource アクションの両方にアクセスする必要があります。QLDB リソースの許可の詳細については、「Amazon QLDB で IAM が機能する仕組み」を参照してください。

作成時にリソースにタグ付けすることで、リソース作成後にカスタムタグ付けスクリプトを実行する必要がなくなります。テーブルにタグを付けると、それらのタグに基づいてテーブルへのアクセスを制御できます。例えば、特定のタグを持つテーブルにのみフルアクセスを付与できます。JSON ポリシーの例については、「テーブルタグに基づくすべてのアクションへのフルアクセス」を参照してください。

例

```
CREATE TABLE VehicleRegistration
```

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'development'}`)
```

```
CREATE TABLE Vehicle WITH (aws_tags = `{'key1': 'value1', 'key2': 'value2'}`)
```

ドライバーを使用したプログラムでの実行

QLDB ドライバーを使用してこのステートメントをプログラムで実行する方法については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。

- Java: クイックスタートチュートリアル | クックブックリファレンス
- .NET: クイックスタートチュートリアル | クックブックリファレンス
- Go: クイックスタートチュートリアル | クックブックリファレンス
- Node.js: クイックスタートチュートリアル | クックブックリファレンス
- Python: クイックスタートチュートリアル | クックブックリファレンス

Amazon QLDB の削除コマンド

Amazon QLDB では、ドキュメントの新しい最終のリビジョンを作成して、アクティブなドキュメントをテーブル内で削除されたものとしてマークするには、DELETE コマンドを使用します。この最

終リビジョンは、ドキュメントが削除されることを示しています。このオペレーションにより、ドキュメントのライフサイクルは終了します。つまり、同じドキュメント ID を持つドキュメントのリビジョンは作成できません。

この操作を元に戻すことはできません。削除したドキュメントのリビジョン履歴は、履歴関数を使用してクエリできます。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

- 構文
- パラメータ
- 戻り値
- 例
- ドライバーを使用したプログラムでの実行

構文

```
DELETE FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
```

パラメータ

table_name

削除されるデータを含むユーザーテーブルの名前。DML ステートメントは、デフォルトのユーザービューでのみサポートされています。各ステートメントは、単一のテーブルでのみ実行できます。

AS ***table_alias***

(オプション) 削除するテーブルに該当するユーザー定義のエイリアス。AS キーワードはオプションです。

BY *id_alias*

(オプション) 結果セット内の各ドキュメントの id メタデータフィールドにバインドされる、ユーザー定義のエイリアス。このエイリアスは、BY キーワードを使用して FROM 句で宣言する必要があります。これは、デフォルトのユーザービューへのクエリ実行中にドキュメント ID をフィルタ処理する場合に便利です。詳細については、「BY 句を使用したドキュメント ID のクエリの実行」を参照してください。

WHERE *condition*

削除されるドキュメントの選択条件。

> **Note**
>
> WHERE 句を省略すると、テーブル内のすべてのドキュメントが削除されます。

戻り値

documentId - 削除した各ドキュメントの一意の ID。

例

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

ドライバーを使用したプログラムでの実行

QLDB ドライバーを使用してこのステートメントをプログラムで実行する方法については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。

- Java: クイックスタートチュートリアル | クックブックリファレンス
- .NET: クイックスタートチュートリアル | クックブックリファレンス
- Go: クイックスタートチュートリアル | クックブックリファレンス
- Node.js: クイックスタートチュートリアル | クックブックリファレンス
- Python: クイックスタートチュートリアル | クックブックリファレンス

Amazon QLDB の DROP INDEX コマンド

Amazon QLDB では、テーブルのインデックスを削除するには、DROP INDEX コマンドを使用します。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

構文

```
DROP INDEX "indexId" ON table_name WITH (purge = true)
```

> **Note**
>
> WITH (purge = true) 句はすべての DROP INDEX ステートメントに必要です。現在 true のみが値としてサポートされています。
> キーワード purge は大文字と小文字が区別され、すべて小文字にする必要があります。

パラメータ

"*indexId*"

ドロップするインデックスの一意の ID。二重引用符で囲みます。

ON *table_name*

ドロップするインデックスがあるテーブルの名前。

戻り値

`tableId` — ドロップしたインデックスがあったテーブルの一意の ID。

例

```
DROP INDEX "4tPW3fUhaVhDinRgKRLhGU" ON VehicleRegistration WITH (purge = true)
```

Amazon QLDB の DROP TABLE コマンド

Amazon QLDB では、既存のテーブルを非アクティブ化するには、`DROP TABLE` コマンドを使用します。テーブルの削除の取り消し 文を使用すると、再度有効にできます。テーブルを非アクティブ化または再アクティブ化しても、そのドキュメントまたはインデックスには影響ありません。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

構文

```
DROP TABLE table_name
```

パラメータ

table_name

非アクティブ化するテーブルの名前。テーブルは既に存在しており、ステータスが `ACTIVE` である必要があります。

戻り値

`tableId` - 非アクティブ化したテーブルの一意の ID。

例

```
DROP TABLE VehicleRegistration
```

Amazon QLDB の FROM (INSERT、REMOVE、または SET) コマンド

Amazon QLDB では、`FROM` で始まるステートメントは、PartiQL 拡張機能です。この拡張機能を使用すると、ドキュメント内の特定の要素を挿入および削除できます。このステートメントを使用して、UPDATE コマンドと同様に、ドキュメント内の既存の要素を更新することもできます。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

構文

FROM-INSERT

既存のドキュメント内で新しい要素を挿入します。新しいトップレベルドキュメントをテーブルに挿入するには、INSERT を使用する必要があります。

```
FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
```

```
INSERT INTO element VALUE data [ AT key_name ]
```

FROM-REMOVE

ドキュメント内の既存の要素を削除するか、トップレベルドキュメント全体を削除します。後者は、意味的には従来の DELETE 構文と同じです。

```
FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
REMOVE element
```

FROM-SET

ドキュメント内の 1 つ以上の要素を更新します。要素が存在しない場合は、挿入されます。これは、意味的には従来の UPDATE 構文と同じです。

```
FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
SET element = data [, element = data, ... ]
```

パラメータ

table_name

変更するデータを含むユーザーテーブルの名前。DML ステートメントは、デフォルトのユーザービューでのみサポートされています。各ステートメントは、単一のテーブルでのみ実行できます。

この句には、指定したテーブルにネストされている 1 つ以上のコレクションも含めることができます。詳細については、「ネストされているコレクション」を参照してください。

AS ***table_alias***

(オプション) 変更するテーブルに該当するユーザー定義のエイリアス。SET 句、REMOVE 句、INSERT INTO 句、または WHERE 句で使用されるソースエイリアスは、いずれも FROM 句で宣言する必要があります。AS キーワードはオプションです。

BY ***id_alias***

(オプション) 結果セット内の各ドキュメントの id メタデータフィールドにバインドされる、ユーザー定義のエイリアス。このエイリアスは、BY キーワードを使用して FROM 句で宣言する

必要があります。これは、デフォルトのユーザービューへのクエリ実行中にドキュメント ID をフィルタ処理する場合に便利です。詳細については、「BY 句を使用したドキュメント ID のクエリの実行」を参照してください。

WHERE ***condition***

修正されるドキュメントの選択条件。

> **Note**
>
> WHERE 句を省略すると、テーブル内のすべてのドキュメントが修正されます。

element

作成または変更するドキュメントの要素。

###

要素の新しい値。

AT ***key_name***

修正されるドキュメント内に追加されるキー名。対応する VALUE をキー名とともに指定する必要があります。これは、新しい値をドキュメント内の特定の位置 AT に挿入する場合に必要です。

ネストされているコレクション

DML ステートメントは 1 つのテーブルに対してのみ実行できますが、そのテーブル内のドキュメントにネストされているコレクションを追加のソースとして指定できます。ネストされたコレクションに対して宣言した各エイリアスは、WHERE 句で使用できます。さらに、SET、INSERT INTO、REMOVE のいずれかの句で使用できます。

たとえば、次のステートメントの FROM ソースには、VehicleRegistration テーブルおよびネストされている Owners.SecondaryOwners 構造の両方が含まれます。

```
FROM VehicleRegistration r, @r.Owners.SecondaryOwners o
WHERE r.VIN = '1N4AL11D75C109151' AND o.PersonId = 'abc123'
SET o.PersonId = 'def456'
```

次の例では、VIN が '1N4AL11D75C109151' である VehicleRegistration ドキュメント内で、SecondaryOwners リスト内の PersonId が 'abc123' である特定の要素を更新します。この式を使用すると、リストの要素を、インデックスではなく値で指定できます。

戻り値

documentId - 更新または削除した各ドキュメントの ID。

例

ドキュメント内の要素を変更します。要素が存在しない場合は、挿入されます。

```
FROM Vehicle AS v
WHERE v.VIN = '1N4AL11D75C109151' AND v.Color = 'Silver'
SET v.Color = 'Shiny Gray'
```

要素を変更または挿入し、システムによって割り当てられたドキュメント id メタデータフィールドでフィルタ処理します。

```
FROM Vehicle AS v BY v_id
WHERE v_id = 'documentId'
SET v.Color = 'Shiny Gray'
```

ドキュメント内の Owners.SecondaryOwners リストで先頭要素の PersonId フィールドを修正します。

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
```

ドキュメント内の既存の要素を削除します。

```
FROM Person AS p
WHERE p.GovId = '111-22-3333'
REMOVE p.Address
```

ドキュメント全体をテーブルから削除します。

```
FROM Person AS p
```

```
WHERE p.GovId = '111-22-3333'
REMOVE p
```

VehicleRegistration テーブルでドキュメント内の Owners.SecondaryOwners リストの先頭要素を削除します。

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
REMOVE r.Owners.SecondaryOwners[0]
```

{'Mileage':26500} を最上位の名前と値のペアとして Vehicle テーブルのドキュメント内に挿入します。

```
FROM Vehicle AS v
WHERE v.VIN = '1N4AL11D75C109151'
INSERT INTO v VALUE 26500 AT 'Mileage'
```

VehicleRegistration テーブルのドキュメントの Owners.SecondaryOwners フィールドに、名前と値のペアとして {'PersonId':'abc123'} を追加します。このステートメントが有効になるには、Owners.SecondaryOwners が既存であり、リストデータ型であることが必要です。そうでない場合には、INSERT INTO 句内にキーワード AT が必要です。

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' }
```

ドキュメント内の既存の Owners.SecondaryOwners リストで先頭要素として {'PersonId':'abc123'} を挿入します。

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE {'PersonId' : 'abc123'} AT 0
```

ドキュメント内の既存の Owners.SecondaryOwners リストに名前と値のペアを複数追加します。

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners << {'PersonId' : 'abc123'}, {'PersonId' :
 'def456'} >>
```

ドライバーを使用したプログラムでの実行

QLDB ドライバーを使用してこのステートメントをプログラムで実行する方法については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。

- Java: クイックスタートチュートリアル | クックブックリファレンス
- .NET: クイックスタートチュートリアル | クックブックリファレンス
- Go: クイックスタートチュートリアル | クックブックリファレンス
- Node.js: クイックスタートチュートリアル | クックブックリファレンス
- Python: クイックスタートチュートリアル | クックブックリファレンス

Amazon QLDB のINSERT コマンド

Amazon QLDB では、テーブルに 1 つまたは複数の Amazon Ion ドキュメントを追加するには、`INSERT` コマンドを使用します。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

構文

単一のドキュメントを挿入します。

```
INSERT INTO table_name VALUE document
```

複数のドキュメントを挿入します。

```
INSERT INTO table_name << document, document, ... >>
```

パラメータ

table_name

データを挿入するユーザーテーブルの名前。このテーブルは既存であることが必要です。DML ステートメントは、デフォルトのユーザービューでのみサポートされています。

document

有効な QLDB ドキュメント。1 つ以上のドキュメントを指定する必要があります。複数のドキュメントはカンマで区切る必要があります。

ドキュメントは中括弧 (`{...}`) で表す必要があります。

ドキュメント内の各フィールド名は、大文字小文字を区別する Ion シンボルで、PartiQL では一重引用符 (`'...'`) で表すことができます。

文字列値も、PartiQL では一重引用符 (`'...'`) で表されます。

Ion リテラルはバックティック (`` `...` ``) で示すことができます。

> **Note**
>
> 二重山かっこ (`<<...>>`) は、順序付けられていないコレクション (PartiQL ではバッグと呼ばれます) を示し、複数のドキュメントを挿入する場合にのみ必要です。

戻り値

`documentId` - 挿入した各ドキュメントの ID。

例

単一のドキュメントを挿入します。

```
INSERT INTO VehicleRegistration VALUE
{
```

```
    'VIN' : 'KM8SRDHF6EU074761', --string
    'RegNum' : 1722, --integer
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75, --decimal
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' }
        ]
    },
    'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate' : `2020-06-25T`
}
```

このステートメントは、挿入したドキュメントの一意の ID を次のように返します。

```
{
    documentId: "2kKuOPNB07D2iTPBrUTWGl"
}
```

複数のドキュメントを挿入します。

```
INSERT INTO Person <<
{
    'FirstName' : 'Raul',
    'LastName' : 'Lewis',
    'DOB' : `1963-08-19T`,
    'GovId' : 'LEWISR261LL',
    'GovIdType' : 'Driver License',
    'Address' : '1719 University Street, Seattle, WA, 98109'
},
{
    'FirstName' : 'Brent',
    'LastName' : 'Logan',
    'DOB' : `1967-07-03T`,
    'GovId' : 'LOGANB486CG',
    'GovIdType' : 'Driver License',
    'Address' : '43 Stockert Hollow Road, Everett, WA, 98203'
},
{
    'FirstName' : 'Alexis',
```

```
    'LastName' : 'Pena',
    'DOB' : `1974-02-10T`,
    'GovId' : '744 849 301',
    'GovIdType' : 'SSN',
    'Address' : '4058 Melrose Street, Spokane Valley, WA, 99206'
}
>>
```

このステートメントは、挿入した各ドキュメントの一意の ID を次のように返します。

```
{
    documentId: "6WXzLscsJ3bDWW97Dy8nyp"
},
{
    documentId: "35e0ToZyTGJ7LGvcwrkX65"
},
{
    documentId: "BVHPcH612o7JROQ4yP8jiH"
}
```

ドライバーを使用したプログラムでの実行

QLDB ドライバーを使用してこのステートメントをプログラムで実行する方法については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。

- Java: クイックスタートチュートリアル | クックブックリファレンス
- .NET: クイックスタートチュートリアル | クックブックリファレンス
- Go: クイックスタートチュートリアル | クックブックリファレンス
- Node.js: クイックスタートチュートリアル | クックブックリファレンス
- Python: クイックスタートチュートリアル | クックブックリファレンス

Amazon QLDB の SELECT コマンド

Amazon QLDB では、1 つまたは複数のテーブルからデータを取得するには、SELECT コマンドを使用します。QLDB のすべての SELECT クエリはトランザクションで処理され、トランザクションタイムアウト制限の対象になります。

結果の順序は一定ではなく、SELECT クエリごとに異なる場合があります。QLDB のクエリの結果の順序は変わる可能性があることに注意してください。

特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

> **Warning**
>
> インデックス付きルックアップなしで QLDB でクエリを実行すると、完全なテーブルスキャンが呼び出されます。PartiQL は SQL 互換であるため、このようなクエリをサポートしています。ただし、QLDB の本番環境のユースケースではテーブルスキャンを実行しないでください。テーブルスキャンより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子を使用する WHERE 述語句でステートメントを実行する必要があります (例: `WHERE indexedField = 123` または `WHERE indexedField IN (456, 789)`)。詳細については、「クエリパフォーマンスの最適化」を参照してください。

トピック

構文

```
SELECT [ VALUE ] expression [ AS field_alias ] [, expression, ... ]
FROM source [ AS source_alias ] [ AT idx_alias ] [ BY id_alias ] [, source, ... ]
[ WHERE condition ]
```

パラメータ

VALUE

タプル構造内にラップされている値ではなく raw データ型の値がクエリから返されるようにする式修飾子。

expression

* ワイルドカードからの射影、または結果セットからの 1 つ以上のドキュメントフィールドの射影リスト。式には、PartiQL 関数 への呼び出し、または PartiQL 演算子 によって変更されたフィールドで構成できます。

AS ***field_alias***

(オプション) 最終結果セットで使用されるフィールドの一時的なユーザー定義エイリアス。AS キーワードはオプションです。

シンプルなフィールド名ではない式に対してエイリアスを指定しない場合、結果セットはそのフィールドに対してデフォルト名を適用します。

FROM ***source***

クエリされるソース。現在サポートされているソースは、テーブル名、テーブル間の 内部結合、ネストされた SELECT クエリ (「ネストされたクエリに関する制限事項」の対象)、およびテーブルの履歴関数呼び出しだけです。

1 つ以上のソースを指定する必要があります。複数のソースはカンマで区切る必要があります。

AS ***source_alias***

(オプション) クエリが実行されるソースに該当するユーザー定義のエイリアス名。SELECT 句または WHERE 句で使用されるソースエイリアスは、いずれも FROM 句で宣言する必要があります。AS キーワードはオプションです。

AT ***idx_alias***

(オプション) ソースからリスト内の各要素のインデックス番号 (序数) にバインドされる、ユーザー定義のエイリアス。このエイリアスは、AT キーワードを使用して FROM 句で宣言する必要があります。

BY ***id_alias***

(オプション) 結果セット内の各ドキュメントの id メタデータフィールドにバインドされる、ユーザー定義のエイリアス。このエイリアスは、BY キーワードを使用して FROM 句で宣言する必要があります。これは、デフォルトのユーザービューへのクエリ実行中にドキュメント ID を射影またはフィルタ処理する場合に便利です。詳細については、「BY 句を使用したドキュメント ID のクエリの実行」を参照してください。

WHERE ***condition***

クエリの選択条件および結合条件 (該当する場合)。

> **Note**
>
> WHERE 句を省略すると、テーブル内のすべてのドキュメントが取得されます。

Joins

現在は内部結合のみサポートされています。以下のように、明示的な INNER JOIN 句を使用して内部結合クエリを記述することができます。この構文では、JOIN と ON を組み合わせる必要があります。INNER キーワードはオプションです。

```
SELECT expression
FROM table1 AS t1 [ INNER ] JOIN table2 AS t2
ON t1.element = t2.element
```

また以下のように、黙示的な構文を使用して内部結合を記述することもできます。

```
SELECT expression
FROM table1 AS t1, table2 AS t2
WHERE t1.element = t2.element
```

ネストされたクエリに関する制限事項

SELECT 式や FROM ソースの中にネストされたクエリ (サブクエリ) を記述できます。このようなクエリには、主な制限として、一番外側のクエリのみがグローバルデータベース環境にアクセスできるという制限があります。ここでは、テーブル VehicleRegistration と Person を持つ台帳があると仮定します。以下のネストされたクエリは、内部の SELECT が Person にアクセスしようとしているため、有効ではありません。

```
SELECT r.VIN,
    (SELECT p.PersonId FROM Person AS p WHERE p.PersonId =
 r.Owners.PrimaryOwner.PersonId) AS PrimaryOwner
FROM VehicleRegistration AS r
```

一方、以下のネストされたクエリは有効です。

```
SELECT r.VIN, (SELECT o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM VehicleRegistration AS r
```

例

次のクエリは、すべてを対象とするワイルドカードと IN 演算子を使用する標準 WHERE 述語句を指定した基本的な SELECT を示しています。

```
SELECT * FROM Vehicle
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

以下は、文字列フィルタを使用した SELECT の射影を示しています。

```
SELECT FirstName, LastName, Address
FROM Person
WHERE Address LIKE '%Seattle%'
AND GovId = 'LEWISR261LL'
```

以下は、ネストされたデータを平坦化する相関サブクエリを示しています。ここで @ 文字は構文的に省略可能です。ただし、Owners という名前の別のコレクション (存在する場合) ではなく、VehicleRegistration 内のネストされた Owners 構造を必要とすることを明示的に示しています。コンテキストの詳細については、「データと履歴の使用」章の「ネストされたデータ」を参照してください。

```
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

以下は、ネストされたデータを射影する SELECT リスト内のサブクエリと、暗黙的な内部結合を示しています。

```
SELECT
    v.Make,
    v.Model,
    (SELECT VALUE o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM
    VehicleRegistration AS r, Vehicle AS v
WHERE
    r.VIN = v.VIN AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

以下は、明示的な内部結合を示しています。

```
SELECT
    v.Make,
    v.Model,
    r.Owners
FROM
    VehicleRegistration AS r JOIN Vehicle AS v
ON
    r.VIN = v.VIN
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

以下は、BY 句を使用したドキュメント id メタデータフィールドの射影を示しています。

```
SELECT
    r_id,
    r.VIN
FROM
    VehicleRegistration AS r BY r_id
WHERE
    r_id = 'documentId'
```

以下では、BY 句を使用して、それぞれ PersonId および ドキュメント id フィールドで DriversLicense テーブルと Person テーブルを結合します。

```
SELECT * FROM DriversLicense AS d INNER JOIN Person AS p BY pid
ON d.PersonId = pid
WHERE pid = 'documentId'
```

以下では、コミット済みビュー を使用して、それぞれ PersonId および ドキュメント id フィールドで DriversLicense テーブルと Person テーブルを結合します。

```
SELECT * FROM DriversLicense AS d INNER JOIN _ql_committed_Person AS cp
ON d.PersonId = cp.metadata.id
WHERE cp.metadata.id = 'documentId'
```

以下では、テーブル VehicleRegistration 内のドキュメントについて、Owners.SecondaryOwners リスト内の各個人の PersonId とインデックス番号 (序数) を返します。

```
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = 'KM8SRDHF6EU074761'
```

ドライバーを使用したプログラムでの実行

QLDB ドライバーを使用してこのステートメントをプログラムで実行する方法については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。

- Java: クイックスタートチュートリアル | クックブックリファレンス
- .NET: クイックスタートチュートリアル | クックブックリファレンス
- Go: クイックスタートチュートリアル | クックブックリファレンス
- Node.js: クイックスタートチュートリアル | クックブックリファレンス
- Python: クイックスタートチュートリアル | クックブックリファレンス

Amazon QLDB の UPDATE コマンド

Amazon QLDB では、ドキュメント内にある 1 つまたは複数の要素の値を変更するには、UPDATE コマンドを使用します。要素が存在しない場合は、挿入されます。

このコマンドを使用して、FROM (INSERT、REMOVE、または SET) ステートメントと同様に、ドキュメント内の特定の要素を明示的に挿入および削除することもできます。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

構文

UPDATE-SET

ドキュメント内の 1 つ以上の要素を更新します。要素が存在しない場合は、挿入されます。これはセマンティック的には FROM-SET ステートメントと同じです。

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]
SET element = data [, element = data, ... ]
[ WHERE condition ]
```

UPDATE-INSERT

既存のドキュメント内で新しい要素を挿入します。新しいトップレベルドキュメントをテーブルに挿入するには、INSERT を使用する必要があります。

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]
INSERT INTO element VALUE data [ AT key_name ]
[ WHERE condition ]
```

UPDATE-REMOVE

ドキュメント内の既存の要素を削除するか、トップレベルドキュメント全体を削除します。後者は、意味的には従来の DELETE 構文と同じです。

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]
REMOVE element
[ WHERE condition ]
```

パラメータ

table_name

変更するデータを含むユーザーテーブルの名前。DML ステートメントは、デフォルトのユーザービューでのみサポートされています。各ステートメントは、単一のテーブルでのみ実行できます。

AS ***table_alias***

(オプション) 更新するテーブルに該当するユーザー定義のエイリアス。AS キーワードはオプションです。

BY ***id_alias***

(オプション) 結果セット内の各ドキュメントの `id` メタデータフィールドにバインドされる、ユーザー定義のエイリアス。このエイリアスは、BY キーワードを使用して UPDATE 句で宣言する必要があります。これは、デフォルトのユーザービューへのクエリ実行中にドキュメント ID をフィルタ処理する場合に便利です。詳細については、「BY 句を使用したドキュメント ID のクエリの実行」を参照してください。

element

作成または変更するドキュメントの要素。

###

要素の新しい値。

AT ***key_name***

修正されるドキュメント内に追加されるキー名。対応する VALUE をキー名とともに指定する必要があります。これは、新しい値をドキュメント内の特定の位置 AT に挿入する場合に必要です。

WHERE ***condition***

修正されるドキュメントの選択条件。

> **Note**
>
> WHERE 句を省略すると、テーブル内のすべてのドキュメントが修正されます。

戻り値

`documentId` - 更新した各ドキュメントの一意の ID。

例

ドキュメント内のフィールドを更新します。フィールドが存在しない場合は、挿入されます。

```
UPDATE Person AS p
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE p.GovId = '111-22-3333'
```

システム割当てのドキュメント id メタデータフィールドでフィルタ処理します。

```
UPDATE Person AS p BY pid
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE pid = 'documentId'
```

ドキュメント全体を上書きします。

```
UPDATE Person AS p
SET p = {
    'FirstName' : 'Rosemarie',
    'LastName' : 'Holloway',
    'DOB' : `1977-06-18T`,
    'GovId' : '111-22-3333',
    'GovIdType' : 'Driver License',
    'Address' : '4637 Melrose Street, Ellensburg, WA, 98926'
}
WHERE p.GovId = '111-22-3333'
```

ドキュメント内の Owners.SecondaryOwners リストで先頭要素の PersonId フィールドを修正します。

```
UPDATE VehicleRegistration AS r
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
WHERE r.VIN = '1N4AL11D75C109151'
```

{'Mileage':26500} を最上位の名前と値のペアとして Vehicle テーブルのドキュメント内に挿入します。

```
UPDATE Vehicle AS v
INSERT INTO v VALUE 26500 AT 'Mileage'
WHERE v.VIN = '1N4AL11D75C109151'
```

VehicleRegistration テーブルのドキュメントの Owners.SecondaryOwners フィールドに、名前と値のペアとして {'PersonId':'abc123'} を追加します。このステートメントが有効になるには、Owners.SecondaryOwners が既存であり、リストデータ型であることが必要です。そうでない場合には、INSERT INTO 句内にキーワード AT が必要です。

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' }
```

```
WHERE r.VIN = '1N4AL11D75C109151'
```

ドキュメント内の既存の Owners.SecondaryOwners リストで先頭要素として {'PersonId':'abc123'} を挿入します。

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners VALUE {'PersonId' : 'abc123'} AT 0
WHERE r.VIN = '1N4AL11D75C109151'
```

ドキュメント内の既存の Owners.SecondaryOwners リストに名前と値のペアを複数追加します。

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners << {'PersonId' : 'abc123'}, {'PersonId' :
 'def456'} >>
WHERE r.VIN = '1N4AL11D75C109151'
```

ドキュメント内の既存の要素を削除します。

```
UPDATE Person AS p
REMOVE p.Address
WHERE p.GovId = '111-22-3333'
```

ドキュメント全体をテーブルから削除します。

```
UPDATE Person AS p
REMOVE p
WHERE p.GovId = '111-22-3333'
```

VehicleRegistration テーブルでドキュメント内の Owners.SecondaryOwners リストの先頭要素を削除します。

```
UPDATE VehicleRegistration AS r
REMOVE r.Owners.SecondaryOwners[0]
WHERE r.VIN = '1N4AL11D75C109151'
```

ドライバーを使用したプログラムでの実行

QLDB ドライバーを使用してこのステートメントをプログラムで実行する方法については、「ドライバーの開始方法」の以下のチュートリアルを参照してください。

- Java: クイックスタートチュートリアル | クックブックリファレンス
- .NET: クイックスタートチュートリアル | クックブックリファレンス
- Go: クイックスタートチュートリアル | クックブックリファレンス
- Node.js: クイックスタートチュートリアル | クックブックリファレンス
- Python: クイックスタートチュートリアル | クックブックリファレンス

Amazon QLDB の UNDROP TABLE コマンド

Amazon QLDB では、台帳で以前ドロップした (つまり非アクティブ化した) テーブルを再有効化するには、`UNDROP TABLE` コマンドを使用します。テーブルを非アクティブ化または再アクティブ化しても、そのドキュメントまたはインデックスには影響ありません。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

- 構文
- パラメータ
- 戻り値
- 例

構文

```
UNDROP TABLE "tableId"
```

パラメータ

"*tableId*"

二重引用符で囲まれた、再有効化するテーブルの一意の ID。

テーブルは以前に削除されている (テーブルがシステムカタログテーブルの `information_schema.user_tables` に存在し、ステータスが `INACTIVE` になっている) 必要があります。また、同じ名前の有効なテーブルが存在していないことも条件となります。

戻り値

`tableId` - 再有効化したテーブルの一意の ID。

例

```
UNDROP TABLE "5PLf9SXwndd63lPaSIa0O6"
```

Amazon QLDB の PartiQL 関数

Amazon QLDB の PartiQL は、次の SQL 標準関数の組み込みバージョンをサポートしています。

> **Note**
>
> このリストに含まれていない SQL 関数は、QLDB で現在サポートされていません。
> この関数リファレンスは、PartiQL ドキュメントの「組み込み関数」に基づいています。

不明なタイプ (null および欠落) 伝播

特に明記されていない限り、これらの関数はすべて null と欠落した引数値を伝播します。NULL または MISSING の伝達とは、関数の引数が NULL または MISSING の場合に NULL を返すことであると定義されます。以下は、伝播の例です。

```
CHAR_LENGTH(null)    -- null
CHAR_LENGTH(missing) -- null (also returns null)
```

集計関数

- AVG
- COUNT
- MAX
- MIN

- SIZE
- SUM

条件関数

- COALESCE
- EXISTS
- NULLIF

日付および時刻関数

- DATE_ADD
- DATE_DIFF
- EXTRACT
- UTCNOW

スカラー関数

- TXID

文字列関数

- CHAR_LENGTH
- CHARACTER_LENGTH
- LOWER
- SUBSTRING
- TRIM
- UPPER

データ型フォーマット関数

- CAST

- TO_STRING
- TO_TIMESTAMP

Amazon QLDB の AVG 機能

Amazon QLDB では、入力式の値の平均 (算術平均) を返すには、AVG 関数を使用します。この関数は、数値を処理し、Null 値または欠落した値を無視します。

構文

```
AVG ( expression )
```

引数

expression

関数が動作するフィールド名または数値データ型の式。

データ型

サポートされている引数の型:

- `int`
- `decimal`
- `float`

戻り型: `decimal`

例

```
SELECT AVG(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 147.19
SELECT AVG(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 2.
```

関連関数

- COUNT
- MAX

- MIN
- SIZE
- SUM

Amazon QLDB の CAST 関数

Amazon QLDB では、指定された式を値として評価し、その値を指定されたターゲットデータ型に変換するには、CAST 関数を使用します。変換できない場合、関数はエラーを返します。

構文

```
CAST ( expression AS type )
```

引数

expression

関数が値として評価し変換するフィールド名または式。null 値を変換すると、null が返されます。このパラメータには、サポートされている任意の データ型 を使用できます。

###

変換するデータ型の名前。このパラメータは、サポートされている データ型 のいずれかになります。

戻り型

type 引数で指定されたデータ型。

例

次の例は、不明なタイプ (NULL または MISSING) の伝播を示しています。

```
CAST(null    AS null)    -- null
CAST(missing AS null)    -- null
CAST(missing AS missing) -- missing
CAST(null    AS missing) -- missing
CAST(null    AS boolean) -- null (null AS any data type name results in null)
CAST(missing AS boolean) -- missing (missing AS any data type name results in missing)
```

不明な型でない値は、NULL または MISSING にキャストできません。

```
CAST(true AS null)    -- error
CAST(true AS missing) -- error
CAST(1    AS null)    -- error
CAST(1    AS missing) -- error
```

次の例は、AS boolean のキャストを示しています。

```
CAST(true      AS boolean) -- true no-op
CAST(0         AS boolean) -- false
CAST(1         AS boolean) -- true
CAST(`1e0`     AS boolean) -- true (float)
CAST(`1d0`     AS boolean) -- true (decimal)
CAST('a'       AS boolean) -- false
CAST('true'    AS boolean) -- true (SqlName string 'true')
CAST(``'true'`` AS boolean) -- true (Ion symbol `'true'`)
CAST(``'false'`` AS boolean) -- false (Ion symbol `'false'`)
```

次の例は、AS integer のキャストを示しています。

```
CAST(true   AS integer) -- 1
CAST(false  AS integer) -- 0
CAST(1      AS integer) -- 1
CAST(`1d0`  AS integer) -- 1
CAST(`1d3`  AS integer) -- 1000
CAST(1.00   AS integer) -- 1
CAST(1.45   AS integer) -- 1
CAST(1.75   AS integer) -- 1
CAST('12'   AS integer) -- 12
CAST('aa'   AS integer) -- error
CAST(``'22'`` AS integer) -- 22
CAST(``'x'``  AS integer) -- error
```

次の例は、AS float のキャストを示しています。

```
CAST(true   AS float) -- 1e0
CAST(false  AS float) -- 0e0
CAST(1      AS float) -- 1e0
CAST(`1d0`  AS float) -- 1e0
CAST(`1d3`  AS float) -- 1000e0
CAST(1.00   AS float) -- 1e0
```

```
CAST('12'   AS float) -- 12e0
CAST('aa'   AS float) -- error
CAST(``'22'`` AS float) -- 22e0
CAST(``'x'``  AS float) -- error
```

次の例は、AS decimal のキャストを示しています。

```
CAST(true   AS decimal) -- 1.
CAST(false  AS decimal) -- 0.
CAST(1      AS decimal) -- 1.
CAST(`1d0`  AS decimal) -- 1. (REPL printer serialized to 1.)
CAST(`1d3`  AS decimal) -- 1d3
CAST(1.00   AS decimal) -- 1.00
CAST('12'   AS decimal) -- 12.
CAST('aa'   AS decimal) -- error
CAST(``'22'`` AS decimal) -- 22.
CAST(``'x'``  AS decimal) -- error
```

次の例は、AS timestamp のキャストを示しています。

```
CAST(`2001T`                      AS timestamp) -- 2001T
CAST('2001-01-01T'                AS timestamp) -- 2001-01-01T
CAST(``'2010-01-01T00:00:00.000Z'`` AS timestamp) -- 2010-01-01T00:00:00.000Z
CAST(true                         AS timestamp) -- error
CAST(2001                         AS timestamp) -- error
```

次の例は、AS symbol のキャストを示しています。

```
CAST(``'xx'``                   AS symbol) -- xx (``'xx'`` is an Ion symbol)
CAST('xx'                       AS symbol) -- xx ('xx' is a string)
CAST(42                         AS symbol) -- '42'
CAST(`1e0`                      AS symbol) -- '1.0'
CAST(`1d0`                      AS symbol) -- '1'
CAST(true                       AS symbol) -- 'true'
CAST(false                      AS symbol) -- 'false'
CAST(`2001T`                    AS symbol) -- '2001T'
CAST(`2001-01-01T00:00:00.000Z` AS symbol) -- '2001-01-01T00:00:00.000Z'
```

次の例は、AS string のキャストを示しています。

```
CAST(``'xx'``                   AS string) -- "xx" (``'xx'`` is an Ion symbol)
CAST('xx'                       AS string) -- "xx" ('xx' is a string)
```

```
CAST(42                          AS string) -- "42"
CAST(`1e0`                       AS string) -- "1.0"
CAST(`1d0`                       AS string) -- "1"
CAST(true                        AS string) -- "true"
CAST(false                       AS string) -- "false"
CAST(`2001T`                     AS string) -- "2001T"
CAST(`2001-01-01T00:00:00.000Z` AS string) -- "2001-01-01T00:00:00.000Z"
```

次の例は、AS struct のキャストを示しています。

```
CAST(`{ a: 1 }` AS struct) -- {a:1}
CAST(true       AS struct) -- err
```

次の例は、AS list のキャストを示しています。

```
CAST(`[1, 2, 3]`        AS list) -- [1,2,3]
CAST(<<'a', { 'b':2 }>> AS list) -- ["a",{'b':2}]
CAST({ 'b':2 }          AS list) -- error
```

以下の例は、前述の例の一部を含む実行可能なステートメントです。

```
SELECT CAST(true AS integer) FROM << 0 >>            -- 1
SELECT CAST('2001-01-01T' AS timestamp) FROM << 0 >> -- 2001-01-01T
SELECT CAST('xx' AS symbol) FROM << 0 >>             -- xx
SELECT CAST(42 AS string) FROM << 0 >>               -- "42"
```

関連関数

- TO_STRING
- TO_TIMESTAMP

Amazon QLDB の CHAR_LENGTH 関数

Amazon QLDB では、指定された文字列の文字数を返します。文字は単一の Unicode コードポイントとして定義するには、CHAR_LENGTH を使用します。

構文

```
CHAR_LENGTH ( string )
```

CHAR_LENGTH は Amazon QLDB の CHARACTER_LENGTH 関数 のシノニムです。

引数

###

関数が評価するフィールド名または string データ型の式。

戻り型

int

例

```
SELECT CHAR_LENGTH('') FROM << 0 >>        -- 0
SELECT CHAR_LENGTH('abcdefg') FROM << 0 >> -- 7
SELECT CHAR_LENGTH('e#') FROM << 0 >>      -- 2 (because 'e#' is two code points: the
 letter 'e' and combining character U+032B)
```

関連関数

- LOWER
- SUBSTRING
- TRIM
- UPPER

Amazon QLDB の CHARACTER_LENGTH 関数

CHAR_LENGTH 関数のシノニム。

「Amazon QLDB の CHAR_LENGTH 関数」を参照してください。

Amazon QLDB の COALESCE 関数

Amazon QLDB では、1 つ以上の引数のリストが与えられた場合、COALESCE 関数を使用して引数を左から右に評価し、不明な型 (NULL または MISSING) でない最初の値を返します。すべての引数の型が不明な場合、結果は NULL になります。

COALESCE 関数は、NULL および MISSING を伝播しません。

構文

```
COALESCE ( expression [, expression, ... ] )
```

引数

expression

関数が評価する 1 つ以上のフィールド名または式のリスト。各引数には、サポートされている任意の データ型 を指定できます。

戻り型

サポートされている任意のデータ型。戻り値の型は、NULL または、最初に null 以外で欠落していない値として評価される式の型と同じです。

例

```
SELECT COALESCE(1, null) FROM << 0 >>          -- 1
SELECT COALESCE(null, null, 1) FROM << 0 >>    -- 1
SELECT COALESCE(null, 'string') FROM << 0 >>   -- "string"
```

関連関数

- EXISTS
- NULLIF

Amazon QLDB の COUNT 関数

Amazon QLDB では、指定された式で定義されているドキュメントの数を返すには、COUNT 関数を使用します。この関数には 2 つのバリエーションがあります。

- COUNT(*)– NULL 値や欠損値が含まれているかどうかにかかわらず、ターゲットテーブル内のすべてのドキュメントをカウントします。
- COUNT(expression) – 特定の既存のフィールドまたは式に NULL 以外の値が含まれるドキュメントの数を計算します。

> **Warning**
>
> COUNT 関数は最適化されていないため、インデックスルックアップなしでの使用はお勧めしません。インデックス付きルックアップなしで QLDB でクエリを実行すると、完全なテーブルスキャンが呼び出されます。これにより、同時実行の競合やトランザクションのタイムアウトなど、大きなテーブルでパフォーマンスの問題が発生する可能性があります。
> テーブルスキャンを回避するには、インデックス付きフィールドまたはドキュメント ID で等価演算子 (= または IN) を使用する WHERE 述語句でステートメントを実行する必要があります。詳細については、「クエリパフォーマンスの最適化」を参照してください。

構文

```
COUNT ( * | expression )
```

引数

expression

関数が動作するフィールド名または式。このパラメータには、サポートされている任意の データ型 を使用できます。

戻り型

`int`

例

```
SELECT COUNT(*) FROM VehicleRegistration r WHERE r.LicensePlateNumber = 'CA762X' -- 1
SELECT COUNT(r.VIN) FROM Vehicle r WHERE r.VIN = '1N4AL11D75C109151'              -- 1
SELECT COUNT(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>                   -- 3
```

関連関数

- AVG
- MAX
- MIN

- SIZE
- SUM

Amazon QLDB の DATE_ADD 関数

Amazon QLDB では、特定のタイムスタンプ値を指定された間隔でインクリメントするには、DATE_ADD 関数を使用します。

構文

```
DATE_ADD( datetimepart, interval, timestamp )
```

引数

##/####

関数が動作する日付または時刻の部分。このパラメーターは、次のいずれかになります。

- year
- month
- day
- hour
- minute
- second

interval

指定された*#######*に追加する間隔を指定する整数。負の整数は間隔を減算します。

timestamp

関数によってインクリメントされるフィールド名または timestamp データ型の式。

Ion タイムスタンプリテラル値はバックティック (`...`) で示すことができます。フォーマットの詳細とタイムスタンプ値の例については、Amazon Ion 仕様書の「タイムスタンプ」を参照してください。

戻り型

timestamp

例

```
DATE_ADD(year, 5, `2010-01-01T`)                -- 2015-01-01T
DATE_ADD(month, 1, `2010T`)                     -- 2010-02T (result adds precision as
 necessary)
DATE_ADD(month, 13, `2010T`)                    -- 2011-02T (2010T is equivalent to
 2010-01-01T00:00:00.000Z)
DATE_ADD(day, -1, `2017-01-10T`)                -- 2017-01-09T
DATE_ADD(hour, 1, `2017T`)                      -- 2017-01-01T01:00Z
DATE_ADD(hour, 1, `2017-01-02T03:04Z`)          -- 2017-01-02T04:04Z
DATE_ADD(minute, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:05:05.006Z
DATE_ADD(second, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:04:06.006Z

-- Runnable statements
SELECT DATE_ADD(year, 5, `2010-01-01T`) FROM << 0 >> -- 2015-01-01T
SELECT DATE_ADD(day, -1, `2017-01-10T`) FROM << 0 >> -- 2017-01-09T
```

関連関数

- DATE_DIFF
- EXTRACT
- TO_STRING
- TO_TIMESTAMP
- UTCNOW

Amazon QLDB の DATE_DIFF 関数

Amazon QLDB では、指定された 2 つのタイムスタンプの指定された日付部分の差を返すには、DATE_DIFF 関数を使用します。

構文

```
DATE_DIFF( datetimepart, timestamp1, timestamp2 )
```

引数

##/####

関数が動作する日付または時刻の部分。このパラメーターは、次のいずれかになります。

- year
- month
- day
- hour
- minute
- second

timestamp1, *timestamp2*

関数が比較する 2 つのフィールド名または timestamp データ型の式。*timestamp2* が *timestamp1* より後の場合、結果は正です。*timestamp2* が *timestamp1* より前の場合、結果は負になります。

Ion タイムスタンプリテラル値はバックティック (`...`) で示すことができます。フォーマットの詳細とタイムスタンプ値の例については、Amazon Ion 仕様書の「タイムスタンプ」を参照してください。

戻り型

int

例

```
DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`)            -- 1
DATE_DIFF(year, `2010-12T`, `2011-01T`)                  -- 0 (must be at least 12
 months apart to evaluate as a 1 year difference)
DATE_DIFF(month, `2010T`, `2010-05T`)                    -- 4 (2010T is equivalent to
 2010-01-01T00:00:00.000Z)
DATE_DIFF(month, `2010T`, `2011T`)                       -- 12
DATE_DIFF(month, `2011T`, `2010T`)                       -- -12
DATE_DIFF(month, `2010-12-31T`, `2011-01-01T`)           -- 0 (must be at least a full
 month apart to evaluate as a 1 month difference)
DATE_DIFF(day, `2010-01-01T23:00Z`, `2010-01-02T01:00Z`) -- 0 (must be at least 24
 hours apart to evaluate as a 1 day difference)

-- Runnable statements
SELECT DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`) FROM << 0 >> -- 1
SELECT DATE_DIFF(month, `2010T`, `2010-05T`) FROM << 0 >>         -- 4
```

関連関数

- DATE_ADD
- EXTRACT
- TO_STRING
- TO_TIMESTAMP
- UTCNOW

Amazon QLDB の EXISTS 関数

Amazon QLDB では、PartiQL 値が指定された場合、値が空ではないコレクションの場合は EXISTS 関数を使用して TRUE を返します。それ以外の場合、この関数は FALSE を返します。EXISTS への入力がコンテナでない場合、結果は FALSE になります。

EXISTS 関数は、NULL および MISSING を伝播しません。

構文

```
EXISTS ( value )
```

引数

#

関数が評価するフィールド名または式。このパラメータには、サポートされている任意の データ型 を使用できます。

戻り型

bool

例

```
EXISTS(`[]`)        -- false (empty list)
EXISTS(`[1, 2, 3]`) -- true (non-empty list)
EXISTS(`[missing]`) -- true (non-empty list)
```

```
EXISTS(`{}`)        -- false (empty struct)
EXISTS(`{ a: 1 }`)  -- true (non-empty struct)
EXISTS(`()`)        -- false (empty s-expression)
EXISTS(`(+ 1 2)`)   -- true (non-empty s-expression)
EXISTS(1)           -- false
EXISTS(`2017T`)     -- false
EXISTS(null)        -- false
EXISTS(missing)     -- error

-- Runnable statements
SELECT EXISTS(`[]`) FROM << 0 >>        -- false
SELECT EXISTS(`[1, 2, 3]`) FROM << 0 >> -- true
```

関連関数

- COALESCE
- NULLIF

Amazon QLDB の EXTRACT 関数

Amazon QLDB では、特定のタイムスタンプから指定された日付または時刻部分の整数値を返すには、EXTRACT 関数を使用します。

構文

```
EXTRACT ( datetimepart FROM timestamp )
```

引数

##/####

関数が抽出する日付または時刻の部分。このパラメーターは、次のいずれかになります。

- year
- month
- day
- hour
- minute

- second
- timezone_hour
- timezone_minute

timestamp

関数が抽出するフィールド名または timestamp データ型の式。このパラメータが不明な型 (NULL または MISSING) の場合、関数は NULL を返します。

Ion タイムスタンプリテラル値はバックティック (`...`) で示すことができます。フォーマットの詳細とタイムスタンプ値の例については、Amazon Ion 仕様書の「タイムスタンプ」を参照してください。

戻り型

int

例

```
EXTRACT(YEAR FROM `2010-01-01T`)                              -- 2010
EXTRACT(MONTH FROM `2010T`)                                   -- 1 (equivalent to
 2010-01-01T00:00:00.000Z)
EXTRACT(MONTH FROM `2010-10T`)                                -- 10
EXTRACT(HOUR FROM `2017-01-02T03:04:05+07:08`)                -- 3
EXTRACT(MINUTE FROM `2017-01-02T03:04:05+07:08`)              -- 4
EXTRACT(TIMEZONE_HOUR FROM `2017-01-02T03:04:05+07:08`)       -- 7
EXTRACT(TIMEZONE_MINUTE FROM `2017-01-02T03:04:05+07:08`)     -- 8

-- Runnable statements
SELECT EXTRACT(YEAR FROM `2010-01-01T`) FROM << 0 >> -- 2010
SELECT EXTRACT(MONTH FROM `2010T`) FROM << 0 >>      -- 1
```

関連関数

- DATE_ADD
- DATE_DIFF
- TO_STRING
- TO_TIMESTAMP

- UTCNOW

Amazon QLDB の POWER 機能

Amazon QLDB では、指定した文字列のすべての大文字を小文字に変換するには、LOWER 関数を使用します。

構文

```
LOWER ( string )
```

引数

###

関数が変換するフィールド名または string データ型の式。

戻り型

`string`

例

```
SELECT LOWER('AbCdEfG!@#$') FROM << 0 >> -- 'abcdefg!@#$'
```

関連関数

- CHAR_LENGTH
- SUBSTRING
- TRIM
- UPPER

Amazon QLDB の MAX 関数

Amazon QLDB では、数値のセットの最大値を返すには、MAX 関数を使用します。

構文

```
MAX ( expression )
```

引数

expression

関数が動作するフィールド名または数値データ型の式。

データ型

サポートされている引数の型:

- int
- decimal
- float

サポートされている戻り値の型:

- int
- decimal
- float

例

```
SELECT MAX(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 442.30
SELECT MAX(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 3
```

関連関数

- AVG
- COUNT
- MIN
- SIZE
- SUM

Amazon QLDB の MIN 関数

Amazon QLDB では、一連の数値の最小値を返すには、MIN 関数を使用します。

構文

```
MIN ( expression )
```

引数

expression

関数が動作するフィールド名または数値データ型の式。

データ型

サポートされている引数の型:

- int
- decimal
- float

サポートされている戻り値の型:

- int
- decimal
- float

例

```
SELECT MIN(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 30.45
SELECT MIN(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 1
```

関連関数

- AVG

- COUNT
- MAX
- SIZE
- SUM

Amazon QLDB の NULLIF 関数

Amazon QLDB では、2 つの式が与えられた場合、NULLIF 関数を使用して、2 つの式が同じ値に評価される場合に NULL を返します。それ以外の場合、この関数は、最初の式の評価結果を返します。

NULLIF 関数は、NULL および MISSING を伝播しません。

構文

```
NULLIF ( expression1, expression2 )
```

引数

expression1, *expression2*

関数が比較する 2 つのフィールド名または式。これらのパラメータには、サポートされている任意の データ型 を使用できます。

戻り型

サポートされている任意のデータ型。戻り値の型は、NULL または最初の式の型と同じです。

例

```
NULLIF(1, 1)            -- null
NULLIF(1, 2)            -- 1
NULLIF(1.0, 1)          -- null
NULLIF(1, '1')          -- 1
NULLIF([1], [1])        -- null
NULLIF(1, NULL)         -- 1
NULLIF(NULL, 1)         -- null
NULLIF(null, null)      -- null
NULLIF(missing, null)   -- null
```

```
NULLIF(missing, missing) -- null

-- Runnable statements
SELECT NULLIF(1, 1) FROM << 0 >>   -- null
SELECT NULLIF(1, '1') FROM << 0 >> -- 1
```

関連関数

- COALESCE
- EXISTS

Amazon QLDB の SIZE 関数

Amazon QLDB では、指定されたコンテナデータ型 (リスト、構造、またはバッグ) の要素の数を返すには、SIZE 関数を使用します。

構文

```
SIZE ( container )
```

引数

####

関数が動作するコンテナフィールド名または式。

データ型

サポートされている引数の型:

- list
- 構造
- バッグ

戻り型: `int`

SIZE への入力がコンテナでない場合、関数はエラーをスローします。

例

```
SIZE(`[]`)                  -- 0
SIZE(`[null]`)              -- 1
SIZE(`[1,2,3]`)             -- 3
SIZE(<<'foo', 'bar'>>)      -- 2
SIZE(`{foo: bar}`)          -- 1 (number of key-value pairs)
SIZE(`[{foo: 1}, {foo: 2}]`) -- 2
SIZE(12)                    -- error

-- Runnable statements
SELECT SIZE(`[]`) FROM << 0 >>      -- 0
SELECT SIZE(`[1,2,3]`) FROM << 0 >> -- 3
```

関連関数

- AVG
- COUNT
- MAX
- MIN
- SUM

Amazon QLDB のサブストリング関数

Amazon QLDB では、特定の文字列から部分文字列を返すには、SUBSTRING 関数を使用します。部分文字列は、指定された開始インデックスから始まり、文字列の最後の文字または指定された長さで終了します。

構文

```
SUBSTRING ( string, start-index [, length ] )
```

引数

###

部分文字列を抽出するフィールド名または `string` データ型の式。

start-index

抽出を開始する *string* 内の開始位置。負の数を指定することもできます。

string の最初の文字のインデックスは 1 です。

length

(オプション) *string* から抽出する文字数 (コードポイント)。*start-index* から始まり、(*start-index* + *length*) - 1 で終わります。つまり、部分文字列の長さです。負の数を指定することはできません。

このパラメータを指定しない場合、関数は *string* の最後まで続行されます。

戻り型

string

例

```
SUBSTRING('123456789', 0)      -- '123456789'
SUBSTRING('123456789', 1)      -- '123456789'
SUBSTRING('123456789', 2)      -- '23456789'
SUBSTRING('123456789', -4)     -- '123456789'
SUBSTRING('123456789', 0, 999) -- '123456789'
SUBSTRING('123456789', 0, 2)   -- '1'
SUBSTRING('123456789', 1, 999) -- '123456789'
SUBSTRING('123456789', 1, 2)   -- '12'
SUBSTRING('1', 1, 0)           -- ''
SUBSTRING('1', 1, 0)           -- ''
SUBSTRING('1', -4, 0)          -- ''
SUBSTRING('1234', 10, 10)      -- ''

-- Runnable statements
SELECT SUBSTRING('123456789', 1) FROM << 0 >>    -- "123456789"
SELECT SUBSTRING('123456789', 1, 2) FROM << 0 >> -- "12"
```

関連関数

- CHAR_LENGTH
- LOWER

- TRIM
- UPPER

Amazon QLDB の SUM 関数

Amazon QLDB では、入力フィールドまたは式の値の合計を返すには、SUM 関数を使用します。この関数は、数値を処理し、Null 値または欠落した値を無視します。

構文

```
SUM ( expression )
```

引数

expression

関数が動作するフィールド名または数値データ型の式。

データ型

サポートされている引数の型:

- `int`
- `decimal`
- `float`

サポートされている戻り値の型:

- `int` - 整数引数の場合
- `decimal` - 小数点または浮動小数点引数の場合

例

```
SELECT SUM(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 735.95
SELECT SUM(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 6
```

関連関数

- AVG
- COUNT
- MAX
- MIN
- SIZE

Amazon QLDB の TO_STRING 関数

Amazon QLDB では、指定されたフォーマットパターンで、指定されたタイムスタンプの文字列表現を返すには、TO_STRING 関数を使用します。

構文

```
TO_STRING ( timestamp, 'format' )
```

引数

timestamp

関数が文字列に変換するフィールド名または timestamp データ型の式。

Ion タイムスタンプリテラル値はバックティック (`...`) で示すことができます。フォーマットの詳細とタイムスタンプ値の例については、Amazon Ion 仕様書の「タイムスタンプ」を参照してください。

format

結果の日付部分の書式パターンを指定する文字列リテラル。有効な形式については、「タイムスタンプのフォーマット文字列」を参照してください。

戻り型

string

例

```
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y')                    -- "July 20, 1969"
```

```
TO_STRING(`1969-07-20T20:18Z`, 'MMM d, yyyy')                    -- "Jul 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'M-d-yy')                         -- "7-20-69"
TO_STRING(`1969-07-20T20:18Z`, 'MM-d-y')                         -- "07-20-1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y h:m a')                -- "July 20, 1969 8:18
 PM"
TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd''T''H:m:ssX')            --
 "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00Z`, 'y-MM-dd''T''H:m:ssX')      --
 "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd''T''H:m:ssXXXX')    --
 "1969-07-20T20:18:00+0800"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd''T''H:m:ssXXXXX')   --
 "1969-07-20T20:18:00+08:00"

-- Runnable statements
SELECT TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y') FROM << 0 >>           -- "July 20,
 1969"
SELECT TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd''T''H:m:ssX') FROM << 0 >> --
 "1969-07-20T20:18:00Z"
```

関連関数

- CAST
- DATE_ADD
- DATE_DIFF
- EXTRACT
- TO_TIMESTAMP
- UTCNOW

Amazon QLDB の TO_TIMESTAMP 関数

Amazon QLDB では、タイムスタンプを表す文字列が指定された場合、TO_TIMESTAMP 関数を使用して文字列をデータ型に変換します。これは TO_STRING の逆オペレーションです。

構文

```
TO_TIMESTAMP ( string [, 'format' ] )
```

引数

###

関数がタイムスタンプに変換するフィールド名または string データ型の式。

format

(オプション) 入力の *string* の日付部分の書式パターンを定義する文字列リテラル。有効な形式については、「タイムスタンプのフォーマット文字列」を参照してください。

この引数を省略すると、関数は *string* が 標準 Ion タイムスタンプ形式であると見なします。これは、この関数を使用して Ion タイムスタンプを解析するために推奨される方法です。

ゼロパディングは、単一文字の書式記号 (y、M、d、H、h、m、s など) を使用する場合はオプションですが、ゼロパディングのバリエーション (yyyy、MM、dd、HH、hh、mm、ss など) には必須です。

2 桁の年 (書式記号 yy) は特別に扱われます。70 以上の値には 1900 が加算され、70 未満の値には 2000 が加算されます。

月名と AM または PM インジケータでは、大文字と小文字が区別されません。

戻り型

timestamp

例

```
TO_TIMESTAMP('2007T')                          -- `2007T`
TO_TIMESTAMP('2007-02-23T12:14:33.079-08:00') -- `2007-02-23T12:14:33.079-08:00`
TO_TIMESTAMP('2016', 'y')                      -- `2016T`
TO_TIMESTAMP('2016', 'yyyy')                   -- `2016T`
TO_TIMESTAMP('02-2016', 'MM-yyyy')             -- `2016-02T`
TO_TIMESTAMP('Feb 2016', 'MMM yyyy')           -- `2016-02T`
TO_TIMESTAMP('February 2016', 'MMMM yyyy')     -- `2016-02T`

-- Runnable statements
SELECT TO_TIMESTAMP('2007T') FROM << 0 >>              -- 2007T
SELECT TO_TIMESTAMP('02-2016', 'MM-yyyy') FROM << 0 >> -- 2016-02T
```

関連関数

- CAST
- DATE_ADD
- DATE_DIFF
- EXTRACT
- TO_STRING
- UTCNOW

Amazon QLDB の TRIM 機能

Amazon QLDB では、先頭と末尾の空白または指定した文字セットを削除して、指定された文字列をトリミングするには、TRIM 関数を使用します。

構文

```
TRIM ( [ LEADING | TRAILING | BOTH [ characters ] FROM ] string)
```

引数

LEADING

(オプション) 空白または指定した文字を *string* の先頭から削除することを示します。指定しない場合、デフォルトの動作は BOTH です。

TRAILING

(オプション) 空白または指定した文字を *string* の末尾から削除することを示します。指定しない場合、デフォルトの動作は BOTH です。

BOTH

(オプション) 文字列の先頭または末尾の空白または指定した文字を *string* の先頭および末尾から削除することを示します。

##

(オプション) 削除する文字のセットを string として指定します。

このパラメータを指定しない場合、空白が削除されます。

###

関数がトリムするフィールド名または `string` データ型の式。

戻り型

`string`

例

```
TRIM('       foobar         ')               -- 'foobar'
TRIM('      \tfoobar\t         ')            -- '\tfoobar\t'
TRIM(LEADING FROM '       foobar         ')  -- 'foobar         '
TRIM(TRAILING FROM '       foobar         ') -- '       foobar'
TRIM(BOTH FROM '       foobar         ')     -- 'foobar'
TRIM(BOTH '1' FROM '11foobar11')             -- 'foobar'
TRIM(BOTH '12' FROM '1112211foobar22211122') -- 'foobar'

-- Runnable statements
SELECT TRIM('       foobar         ') FROM << 0 >>               -- "foobar"
SELECT TRIM(LEADING FROM '       foobar         ') FROM << 0 >> -- "foobar         "
```

関連関数

- CHAR_LENGTH
- LOWER
- SUBSTRING
- UPPER

Amazon QLDB の TXID 機能

Amazon QLDB では、実行している現在のステートメントの一意のトランザクション ID を返すには、`TXID` 関数を使用します。これは、現在のトランザクションがジャーナルにコミットされたときにドキュメントの `txId` メタデータフィールドに割り当てられる値です。

構文

```
TXID()
```

引数

なし

戻り型

`string`

例

```
SELECT TXID() FROM << 0 >>  -- "L7S9iJqcn9W2M4qOEn27ay"
SELECT TXID() FROM Person WHERE GovId = 'LEWISR261LL'  -- "BKeMb48PNyvHWJGZHkaodG"
```

Amazon QLDB の UPPER 関数

Amazon QLDB では、指定した文字列のすべての小文字を大文字に変換するには、UPPER 関数を使用します。

構文

```
UPPER ( string )
```

引数

###

関数が変換するフィールド名または `string` データ型の式。

戻り型

`string`

例

```
SELECT UPPER('AbCdEfG!@#$') FROM << 0 >> -- 'ABCDEFG!@#$'
```

関連関数

- CHAR_LENGTH

- LOWER
- SUBSTRING
- TRIM

Amazon QLDB の UTCNOW 関数

Amazon QLDB では、協定世界時 (UTC) の現在時刻を `timestamp` として返すには、UTCNOW 関数を使用します。

構文

```
UTCNOW()
```

この関数では、SELECT クエリに FROM 句を指定する必要があります。

引数

なし

戻り型

`timestamp`

例

```
SELECT UTCNOW() FROM << 0 >>  -- 2019-12-27T20:12:16.999Z
SELECT UTCNOW() FROM Person WHERE GovId = 'LEWISR261LL'  -- 2019-12-27T20:12:26.999Z

INSERT INTO Person VALUE { 'firstName': 'Jane', 'createdAt': UTCNOW() }
UPDATE Person p SET p.updatedAt = UTCNOW() WHERE p.firstName = 'John'
```

関連関数

- DATE_ADD
- DATE_DIFF
- EXTRACT
- TO_STRING
- TO_TIMESTAMP

タイムスタンプのフォーマット文字列

このセクションでは、タイムスタンプのフォーマット文字列のリファレンス情報を提供します。

タイムスタンプのフォーマット文字列は、TO_STRING 関数および TO_TIMESTAMP 関数に適用されます。これらの文字列には、日付部分の区切り文字 (「-」、「/」、「:」など') と次の書式記号を含めることができます。

形式	例	説明
yy	70	2 桁の年
y	1970	4 桁の年
yyyy	1970	ゼロ詰めの 4 桁の年
M	1	月の整数
MM	01	ゼロ詰めの月整数
MMM	Jan	省略形の月の名前
MMMM	1 月	完全な月の名前
d	2	日 (1 ~ 31)
dd	02	ゼロ詰めの日 (01 ~ 31)
a	AM または PM	午前午後の指標 (12 時間制用)
h	3	時 (12 時間制、1 ~ 12)
hh	03	ゼロ詰めの時 (12 時間制、01 ~ 12)
H	3	時 (24 時間制、0 ~ 23)
HH	03	ゼロ詰めの時 (24 時間時計、00 ~ 23)
m	4	分 (0 ~ 59)

形式	例	説明
mm	04	ゼロ詰めの分 (00 ~ 59)
s	5	秒 (0 ~ 59)
ss	05	ゼロ詰めの秒 (00 ~ 59)
S	0	1 秒の端数 (精度: 0.1、範囲: 0.0 ~ 0.9)
SS	06	1 秒の端数 (精度: 0.01、範囲: 0.0 ~ 0.99)
SSS	060	1 秒の端数 (精度: 0.001、範囲: 0.0 ~ 0.999)
X	+07 または Z	時間単位の UTC からのオフセット。オフセットが 0 の場合は「Z」
XX	+0700 または Z	時間および分単位の UTC からのオフセット。オフセットが 0 の場合は「Z」
XXX	+07:00 または Z	時間および分単位の UTC からのオフセット。オフセットが 0 の場合は「Z」
x	+07	時間単位の UTC からのオフセット
xx	+0700	時間および分単位の UTC からのオフセット
xxx	+07:00	時間および分単位の UTC からのオフセット

Amazon QLDB の PartiQL ストアドプロシージャ

Amazon QLDB では、EXEC コマンドを使用して次の構文で PartiQL ストアドプロシージャを実行できます。

```
EXEC stored_procedure_name argument [, ... ]
```

QLDB は次のシステムストアドプロシージャをサポートしています。

トピック

- Amazon QLDB の REDACT_REVISION ストアドプロシージャ

Amazon QLDB の REDACT_REVISION ストアドプロシージャ

> **Note**
>
> 2021 年 7 月 22 日より前に作成された台帳は、現時点では秘匿化の対象にはなりません。台帳の作成時間は Amazon QLDB コンソールで確認できます。

Amazon QLDB では、インデックス付きストレージとジャーナルストレージの両方にある個々の非アクティブなドキュメントリビジョンを完全に削除するには、REDACT_REVISION ストアドプロシージャを使用します。このストアドプロシージャは、指定されたリビジョンのユーザーデータをすべて削除します。ただし、ジャーナルシーケンスと、ドキュメント ID やハッシュなどのドキュメントメタデータは変更されません。この操作を元に戻すことはできません。

指定されたドキュメントリビジョンは、履歴上使用頻度の低いリビジョンでなければなりません。ドキュメントの最新の有効なリビジョンは、秘匿化の対象にはなりません。

このストアドプロシージャを実行して秘匿化リクエストを送信すると、QLDB はデータの編集を非同期的に処理します。秘匿化が完了すると、指定されたリビジョン内 (data 構造で表される) のユーザーデータが新しい dataHash フィールドに置き換えられます。このフィールドの値は、削除された data 構造の Amazon Ion ハッシュです。その結果、台帳は全体的なデータ整合性を維持し、既存の検証 API オペレーションを通じて暗号的に検証できる状態を維持します。

サンプルデータを使った秘匿化オペレーションの例については、「ドキュメントのリビジョンを秘匿化する」の「秘匿化の例」を参照してください。

> **Note**
>
> 特定のテーブルでこの PartiQL コマンドを実行するためのアクセスを制御する方法については、「Amazon QLDB の標準アクセス許可モードの開始方法」を参照してください。

トピック

秘匿化に関する考慮事項と制約事項

Amazon QLDB でデータの秘匿化を開始する前に、以下の注意事項と制限事項を確認してください。

- REDACT_REVISION ストアドプロシージャは、使用頻度の低い個別のドキュメントリビジョン内のユーザーデータを対象とします。複数のリビジョンを秘匿化するには、リビジョンごとに 1 回ストアドプロシージャを実行する必要があります。トランザクションごとに 1 つのリビジョンを秘匿化できます。
- ドキュメントリビジョン内の特定のフィールドを秘匿化するには、まず別のデータ操作言語 (DML) ステートメントを使用してリビジョンを変更する必要があります。詳細については、「リビジョン内の特定のフィールドを秘匿化する」を参照してください。
- QLDB が秘匿化リクエストを受け取った後は、そのリクエストをキャンセルしたり、変更したりすることはできません。秘匿化が完了したかどうかを確認するには、リビジョンの data 構造が dataHash フィールドに置き換えられているかどうかを確認できます。詳細については、「秘匿化が完了したかどうかの確認」を参照してください。
- 秘匿化は、QLDB サービスの外部に複製される QLDB データには影響しません。これには、Amazon S3 へのエクスポートと、Amazon Kinesis Data Streams へのストリーミングが含まれます。QLDB の外部に保存されているデータを管理するには、他のデータ保持方法を使用する必要があります。
- 秘匿化は、ジャーナルに記録されている PartiQL ステートメントのリテラル値には影響しません。ベストプラクティスとして、パラメータ化されたステートメントは、リテラル値の代わりに変数プ

レースホルダーを使用してプログラムで実行するようにしてください。プレースホルダーは、秘匿化が必要な機密情報ではなく、疑問符 (?) としてジャーナルに書き込まれます。

QLDB ドライバーを使用して PartiQL ステートメントをプログラムで実行する方法については、サポートされている各プログラミング言語のチュートリアルを「ドライバーの開始方法」で参照してください。

構文

```
EXEC REDACT_REVISION `block-address`, 'table-id', 'document-id'
```

引数

`` `block-address` ``

秘匿化されたドキュメントリビジョンのジャーナルブロックの場所。アドレスは、`strandId` と `sequenceNo` という 2 つのフィールドを含む Amazon Ion 構造です。

これはバックスティックで示される Ion リテラル値です。例:

```
`{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`
```

ブロックアドレスの検索方法については、「ドキュメントのメタデータのクエリの実行」を参照してください。

'table-id'

一重引用符で囲まれた、秘匿化するドキュメントのリビジョンがあったテーブルの一意の ID。

テーブル ID の検索方法については、「システムカタログのクエリの実行」を参照してください。

'document-id'

一重引用符で囲まれた、秘匿化対象のリビジョンの一意のドキュメント ID。

ドキュメント ID の検索方法については、「ドキュメントのメタデータのクエリの実行」を参照してください。

戻り値

秘匿化対象のドキュメントのリビジョンを表す Amazon Ion 構造を以下の形式で表します。

```
{
  blockAddress: {
    strandId: String,
    sequenceNo: Int
  },
  tableId: String,
  documentId: String,
  version: Int
}
```

戻り値構造フィールド

- `blockAddress` – 秘匿化するリビジョンのジャーナルブロックの場所。アドレスには以下の 2 つのフィールドがあります。
  - `strandId` - ブロックを含むジャーナルストランドの一意の ID。
  - `sequenceNo`: ストランド内でブロックの場所を指定するインデックス番号。
- `tableId` – 秘匿化しているリビジョンのテーブルの一意の ID。
- `documentId` – 秘匿化対象のリビジョンの一意のドキュメント ID。
- `version` – 秘匿化対象のドキュメントリビジョンのバージョン番号。

以下に、サンプルデータを含む戻り値構造の例を示します。

```
{
  blockAddress: {
    strandId: "CsRnx0RDoNK6ANEEePa1ov",
    sequenceNo: 134
  },
  tableId: "6GZumdHggklLdMGyQq9DNX",
  documentId: "IXlQPSbfyKMIIsygePeKrZ",
  version: 0
}
```

例

```
EXEC REDACT_REVISION `{strandId:"7z2P0AyQKWD8oFYmGNhi8D", sequenceNo:7}`,
 '8F0TPCmdNQ6JTRpiLj2TmW', '05K8zpGYWynDlEOK5afDRc'
```

Amazon QLDB の PartiQL 演算子

Amazon QLDB の PartiQL は、以下の SQL 標準演算子をサポートしています。

> **Note**
>
> このリストに含まれていない SQL 演算子は、QLDB では現在サポートされていません。

算術演算子

演算子	説明
+	Add
-	- (減算)
*	[Multiply] (乗算)
/	/ (除算)
%	モジュロ

比較演算子

演算子	説明
=	Equal to
>	超
<	未満

演算子	説明
>=	以上
<=	以下
<>	等しくない

論理演算子

演算子	説明
AND	TRUE AND で区切られたすべての条件が TRUE の場合に
BETWEEN	TRUE オペランドが比較の範囲内にある場合に
IN	オペランドが式のリストの 1 つに等しい場合に TRUE
IS	オペランドが指定されたデータ型の場合に TRUE (NULL または MISSING を含む)
LIKE	オペランドがパターンに一致する場合に TRUE
NOT	指定されたブール式の値を反転する
OR	OR で区切られた条件のいずれかが TRUEの場合に TRUE

文字列演算子

演算子	説明
||	|| 記号の両側の 2 つの文字列を連結し、連結された文字列を返します。一方または両方の文

演算子	説明
	字列が NULL の場合、連結の結果は null になります。

Amazon QLDB の予約キーワード

以下に示しているのは、Amazon QLDB での PartiQL の予約キーワードのリストです。予約キーワードは、二重引用符で囲まれた引用された識別子として使用します ("user" など)。QLDB での PartiQL の引用規則については、「PartiQL での Ion のクエリ」を参照してください。

> **Important**
>
> PartiQL は SQL-92 と下位互換性があるため、このリストのキーワードはすべて予約済みと見なされます。ただし、QLDB では、これらの予約語のサブセットのみがサポートされます。QLDB が現在サポートしている SQL キーワードのリストについては、以下のトピックを参照してください。
>
> - PartiQL 関数
> - PartiQL 演算子
> - PartiQL コマンド

```
ABSOLUTE
ACTION
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BAG
```

```
BEGIN
BETWEEN
BIT
BIT_LENGTH
BLOB
BOOL
BOOLEAN
BOTH
BY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHARACTER_LENGTH
CHAR_LENGTH
CHECK
CLOB
CLOSE
COALESCE
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DATE
DATE_ADD
```

```
DATE_DIFF
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DISCONNECT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
END
END-EXEC
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
```

```
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEADING
LEFT
LEVEL
LIKE
LIMIT
LIST
LOCAL
LOWER
MATCH
MAX
MIN
MINUTE
MISSING
MODULE
MONTH
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
```

```
NO
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
ON
ONLY
OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PAD
PARTIAL
PIVOT
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
RELATIVE
REMOVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SESSION
```

```
SESSION_USER
SET
SEXP
SIZE
SMALLINT
SOME
SPACE
SQL
SQLCODE
SQLERROR
SQLSTATE
STRING
STRUCT
SUBSTRING
SUM
SYMBOL
SYSTEM_USER
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TO_STRING
TO_TIMESTAMP
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
TRIM
TRUE
TUPLE
TXID
UNDROP
UNION
UNIQUE
UNKNOWN
UNPIVOT
UPDATE
UPPER
USAGE
USER
```

```
USING
UTCNOW
VALUE
VALUES
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE
```

Amazon QLDB の Amazon Ion データ形式リファレンス

Amazon QLDB は、Amazon Ion を PartiQL の型のサブセットと統合するデータ表記モデルを使用します。このセクションでは、Ion ドキュメントデータ形式のリファレンス概要を、PartiQL との統合とは別に紹介します。

Amazon QLDB での PartiQL による Ion のクエリ

QLDB で PartiQL を使用して Ion データをクエリするための構文とセマンティクスについては、「Amazon QLDB の PartiQL リファレンス」の「PartiQL での Ion のクエリ」を参照してください。

QLDB 台帳の Ion データをクエリおよび処理するコード例については、「Amazon Ion コード例」と「Amazon Ion の操作」を参照してください。

トピック

- Amazon Ion の概要
- Ion 仕様
- JSON との互換性
- JSON からの拡張機能
- Ion テキストの例
- API リファレンス
- QLDB の Amazon Ion コード例

Amazon Ion の概要

Ion は、Amazon 内部で独自に開発された、オープンソースでリッチタイプの自己記述型階層データシリアル化フォーマットです。Ion は、構造化データと非構造化データの両方を一緒に保存する抽象データモデルに基づいています。Ion は JSON のスーパーセットであるため、有効な JSON ドキュメントは有効な Ion ドキュメントでもあります。このガイドの内容は、JSON を使って基本的な作業ができる程度の知識があることを前提としています。JSON についてまだよく理解できていない方は、「JSON の紹介」を参照してください。

Ion ドキュメントは、人間が読めるテキスト形式またはバイナリエンコード形式で互換的に表記できます。JSON と同様に、テキスト形式は読み書きが容易で、ラピッドプロトタイピングをサポートしています。バイナリエンコード形式はよりコンパクトで、効率よく保持、送信および解析できます。Ion プロセッサは、データ喪失なく同じデータ構造セットを正確に再現できるように、両形式間でコードを変換できます。この機能によって、アプリケーションはさまざまなユースケースに応じてデータを処理方法を最適化できます。

> **Note**
>
> Ion データモデルは厳格な値ベースであり、リファレンスはサポートしていません。このため Ion データモデルでは、任意の深さまでネストできるデータ階層を表すことはできますが、有向グラフを表すことはできません。

Ion 仕様

Ion の主要なデータ型、そのすべての説明、値の形式の詳細のリストについては、Amazon の GitHub サイトにある Ion の仕様書を参照してください。

アプリケーション開発を効率化するために、Amazon Ion は Ion データを処理するクライアントライブラリを提供します。Ion データを処理するための一般的なユースケースのコード例については、GitHub の「Amazon Ion クックブック」を参照してください。

JSON との互換性

JSON と同様に、Amazon Ion ドキュメントは、一連のプリミティブデータ型と一連の再帰的に定義されるコンテナ型を使って作成できます。Ion のデータ型には、以下のような従来の JSON データ型が含まれています。

- `null`: 汎用型なし null (空) 値。加えて Ion は、以下のセクションにも記載されているとおり、プリミティブ型ごとに個別の null 型をサポートしています。
- `bool`: ブール値。
- `string`: Unicode 文字リテラル。
- `list`: 異種値の順序付きコレクション。
- `struct`: 名前と値のペアの順序なしコレクション。JSON と同様に、`struct` では、名前あたりの値を複数にできますが、これは概してお勧めできません。

JSON からの拡張機能

数値型

曖昧な JSON `number` 型に代え、Amazon Ion では、数値を次のいずれかの型として厳格に定義しています。

- `int`: 任意の大きさの符号付き整数。
- `decimal`: 任意精度の 10 進数エンコード実数。
- `float`: バイナリエンコード浮動小数点数 (64 ビット IEEE)。

ドキュメントを解析する場合、Ion プロセッサは次のように数値型を割り当てます。

- `int`: 指数も小数点もない数値 (`100200` など)。
- `decimal`: 小数点はあるが指数はない数値 (`0.00001`、`200.0` など)。
- `float`: 科学的記数法または E 表記法といった指数のある数値 (`2e0`、`3.1e-4` など)。

新しいデータ型

Amazon Ion には次のデータ型が追加されています。

- `timestamp`: 任意精度の日付/時刻/タイムゾーン。
- `symbol`: Unicode シンボリックアトム (識別子など)。
- `blob`: ユーザー定義エンコードのバイナリデータ。
- `clob`: ユーザー定義エンコードのテキストデータ。

- sexp: アプリケーション定義のセマンティクスを持つ値の順序付きコレクション。

null 型

JSON で定義されている汎用 null 型に加え、Amazon Ion では、プリミティブ型ごとに個別の null 型をサポートしています。この型は、厳格なデータ型を維持できる値がないことを示します。

```
null
null.null       // Identical to untyped null
null.bool
null.int
null.float
null.decimal
null.timestamp
null.string
null.symbol
null.blob
null.clob
null.struct
null.list
null.sexp
```

Ion テキストの例

```
// Here is a struct, which is similar to a JSON object.
{
    // Field names don't always have to be quoted.
    name: "fido",

    // This is an integer.
    age: 7,

    // This is a timestamp with day precision.
    birthday: 2012-03-01T,

    // Here is a list, which is like a JSON array.
    toys: [
        // These are symbol values, which are like strings,
        // but get encoded as integers in binary.
        ball,
        rope
```

```
    ],
}
```

API リファレンス

- ion-go
- ion-java
- ion-js
- ion-python

QLDB の Amazon Ion コード例

このセクションでは、Amazon QLDB 台帳のドキュメント値を読み書きして Amazon Ion データを処理するコード例を示します。コード例では、QLDB ドライバーを使用して台帳で PartiQL ステートメントを実行します。これらの例は、「サンプルアプリケーションチュートリアルを使用した Amazon QLDB の使用開始」のサンプルアプリケーションの一部であり、AWS Samples GitHub サイトでオープンソース化されています。

Ion データ処理の一般的ユースケースを示す汎用のコード例については、GitHub の Amazon Ion クックブックを参照してください。

コードの実行

各プログラミング言語のチュートリアルコードでは、以下の手順を実行します。

1. `vehicle-registration` サンプル台帳に接続します。
2. `IonTypes` という名前のテーブルを作成します。
3. 1 つの `Name` フィールドを持つテーブルにドキュメントを挿入します。
4. サポートされている各 Ion データ型について:
   a. ドキュメントの `Name` フィールドを、データ型のリテラル値で更新します。
   b. テーブルをクエリして、ドキュメントの最新リビジョンを取得します。
   c. `Name` の値が、予期される型と一致することを確認して、元のデータ型のプロパティを保持していることを検証します。
5. `IonTypes` テーブルをドロップします。

> **Note**
>
> このチュートリアルコードを実行する前に、vehicle-registration という名前の台帳を作成する必要があります。

Java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonBlob;
import com.amazon.ion.IonBool;
import com.amazon.ion.IonClob;
import com.amazon.ion.IonDecimal;
import com.amazon.ion.IonFloat;
import com.amazon.ion.IonInt;
import com.amazon.ion.IonList;
import com.amazon.ion.IonNull;
import com.amazon.ion.IonSexp;
```

```
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSymbol;
import com.amazon.ion.IonTimestamp;
import com.amazon.ion.IonValue;
import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.Collections;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;

/**
 * Insert all the supported Ion types into a ledger and verify that they are stored
 and can be retrieved properly, retaining
 * their original properties.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public class InsertIonTypes {
    public static final Logger log = LoggerFactory.getLogger(InsertIonTypes.class);
    public static final String TABLE_NAME = "IonTypes";

    private InsertIonTypes() {}

    /**
     * Update a document's Name value in the database. Then, query the value of the
 Name key and verify the expected Ion type was
     * saved.
     *
     * @param txn
     *              The {@link TransactionExecutor} for statement execution.
     * @param ionValue
     *              The {@link IonValue} to set the document's Name value to.
     *
     * @throws AssertionError when no value is returned for the Name key or if the
 value does not match the expected type.
     */
    public static void updateRecordAndVerifyType(final TransactionExecutor txn,
 final IonValue ionValue) {
```

```
        final String updateStatement = String.format("UPDATE %s SET Name = ?",
TABLE_NAME);
        final List<IonValue> parameters = Collections.singletonList(ionValue);
        txn.execute(updateStatement, parameters);
        log.info("Updated document.");

        final String searchQuery = String.format("SELECT VALUE Name FROM %s",
TABLE_NAME);
        final Result result = txn.execute(searchQuery);

        if (result.isEmpty()) {
            throw new AssertionError("Did not find any values for the Name key.");
        }
        for (IonValue value : result) {
            if (!ionValue.getClass().isInstance(value)) {
                throw new AssertionError(String.format("The queried value, %s, is
not an instance of %s.",
                        value.getClass().toString(),
ionValue.getClass().toString()));
            }
            if (!value.getType().equals(ionValue.getType())) {
                throw new AssertionError(String.format("The queried value type, %s,
does not match %s.",
                        value.getType().toString(), ionValue.getType().toString()));
            }
        }

        log.info("Successfully verified value is instance of {} with type {}.",
ionValue.getClass().toString(),
                ionValue.getType().toString());
    }

    /**
     * Delete a table.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              The name of the table to delete.
     */
    public static void deleteTable(final TransactionExecutor txn, final String
tableName) {
        log.info("Deleting {} table...", tableName);
        final String statement = String.format("DROP TABLE %s", tableName);
```

```
        txn.execute(statement);
        log.info("{} table successfully deleted.", tableName);
    }

    public static void main(final String... args) {
        final IonBlob ionBlob = Constants.SYSTEM.newBlob("hello".getBytes());
        final IonBool ionBool = Constants.SYSTEM.newBool(true);
        final IonClob ionClob = Constants.SYSTEM.newClob("{{'This is a CLOB of
text.'}}".getBytes());
        final IonDecimal ionDecimal = Constants.SYSTEM.newDecimal(0.1);
        final IonFloat ionFloat = Constants.SYSTEM.newFloat(0.2);
        final IonInt ionInt = Constants.SYSTEM.newInt(1);
        final IonList ionList = Constants.SYSTEM.newList(new int[]{1, 2});
        final IonNull ionNull = Constants.SYSTEM.newNull();
        final IonSexp ionSexp = Constants.SYSTEM.newSexp(new int[]{2, 3});
        final IonString ionString = Constants.SYSTEM.newString("string");
        final IonStruct ionStruct = Constants.SYSTEM.newEmptyStruct();
        ionStruct.put("brand", Constants.SYSTEM.newString("ford"));
        final IonSymbol ionSymbol = Constants.SYSTEM.newSymbol("abc");
        final IonTimestamp ionTimestamp =
Constants.SYSTEM.newTimestamp(Timestamp.now());

        final IonBlob ionNullBlob = Constants.SYSTEM.newNullBlob();
        final IonBool ionNullBool = Constants.SYSTEM.newNullBool();
        final IonClob ionNullClob = Constants.SYSTEM.newNullClob();
        final IonDecimal ionNullDecimal = Constants.SYSTEM.newNullDecimal();
        final IonFloat ionNullFloat = Constants.SYSTEM.newNullFloat();
        final IonInt ionNullInt = Constants.SYSTEM.newNullInt();
        final IonList ionNullList = Constants.SYSTEM.newNullList();
        final IonSexp ionNullSexp = Constants.SYSTEM.newNullSexp();
        final IonString ionNullString = Constants.SYSTEM.newNullString();
        final IonStruct ionNullStruct = Constants.SYSTEM.newNullStruct();
        final IonSymbol ionNullSymbol = Constants.SYSTEM.newNullSymbol();
        final IonTimestamp ionNullTimestamp = Constants.SYSTEM.newNullTimestamp();


        ConnectToLedger.getDriver().execute(txn -> {
            CreateTable.createTable(txn, TABLE_NAME);
            final Document document = new
Document(Constants.SYSTEM.newString("val"));
            InsertDocument.insertDocuments(txn, TABLE_NAME,
Collections.singletonList(document));

            updateRecordAndVerifyType(txn, ionBlob);
```

```
            updateRecordAndVerifyType(txn, ionBool);
            updateRecordAndVerifyType(txn, ionClob);
            updateRecordAndVerifyType(txn, ionDecimal);
            updateRecordAndVerifyType(txn, ionFloat);
            updateRecordAndVerifyType(txn, ionInt);
            updateRecordAndVerifyType(txn, ionList);
            updateRecordAndVerifyType(txn, ionNull);
            updateRecordAndVerifyType(txn, ionSexp);
            updateRecordAndVerifyType(txn, ionString);
            updateRecordAndVerifyType(txn, ionStruct);
            updateRecordAndVerifyType(txn, ionSymbol);
            updateRecordAndVerifyType(txn, ionTimestamp);

            updateRecordAndVerifyType(txn, ionNullBlob);
            updateRecordAndVerifyType(txn, ionNullBool);
            updateRecordAndVerifyType(txn, ionNullClob);
            updateRecordAndVerifyType(txn, ionNullDecimal);
            updateRecordAndVerifyType(txn, ionNullFloat);
            updateRecordAndVerifyType(txn, ionNullInt);
            updateRecordAndVerifyType(txn, ionNullList);
            updateRecordAndVerifyType(txn, ionNullSexp);
            updateRecordAndVerifyType(txn, ionNullString);
            updateRecordAndVerifyType(txn, ionNullStruct);
            updateRecordAndVerifyType(txn, ionNullSymbol);
            updateRecordAndVerifyType(txn, ionNullTimestamp);

            deleteTable(txn, TABLE_NAME);
        });
    }

    /**
     * This class represents a simple document with a single key, Name, to use for
the IonTypes table.
     */
    private static class Document {
        private final IonValue name;

        @JsonCreator
        private Document(@JsonProperty("Name") final IonValue name) {
            this.name = name;
        }

        @JsonProperty("Name")
        private IonValue getName() {
```

```
            return name;
        }
    }
}
```

Node.js

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { AssertionError } from "assert";
import { dom, IonType, IonTypes } from "ion-js";

import { insertDocument } from "./InsertDocument";
import { getQldbDriver } from "./ConnectToLedger";
import { createTable } from "./CreateTable";
import { error, log } from "./qldb/LogUtil";

const TABLE_NAME: string = "IonTypes";

/**
```

```
 * Delete a table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to delete.
 * @returns Promise which fulfills with void.
 */
export async function deleteTable(txn: TransactionExecutor, tableName: string):
 Promise<void> {
    log(`Deleting ${tableName} table...`);
    const statement: string = `DROP TABLE ${tableName}`;
    await txn.execute(statement);
    log(`${tableName} table successfully deleted.`);
}

/**
 * Update a document's Name value in QLDB. Then, query the value of the Name key and
 verify the expected Ion type was
 * saved.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param parameter The IonValue to set the document's Name value to.
 * @param ionType The Ion type that the Name value should be.
 * @returns Promise which fulfills with void.
 */
async function updateRecordAndVerifyType(
    txn: TransactionExecutor,
    parameter: any,
    ionType: IonType
): Promise<void> {
    const updateStatement: string = `UPDATE ${TABLE_NAME} SET Name = ?`;
    await txn.execute(updateStatement, parameter);
    log("Updated record.");

    const searchStatement: string = `SELECT VALUE Name FROM ${TABLE_NAME}`;
    const result: Result = await txn.execute(searchStatement);

    const results: dom.Value[] = result.getResultList();

    if (0 === results.length) {
        throw new AssertionError({
            message: "Did not find any values for the Name key."
        });
    }

    results.forEach((value: dom.Value) => {
        if (value.getType().binaryTypeId !== ionType.binaryTypeId) {
```

```
            throw new AssertionError({
                message: `The queried value type, ${value.getType().name}, does not
 match expected type, ${ionType.name}.`
            });
        }
    });

    log(`Successfully verified value is of type ${ionType.name}.`);
}

/**
 * Insert all the supported Ion types into a table and verify that they are stored
 and can be retrieved properly,
 * retaining their original properties.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await createTable(txn, TABLE_NAME);
            await insertDocument(txn, TABLE_NAME, [{ "Name": "val" }]);
            await updateRecordAndVerifyType(txn, dom.load("null"), IonTypes.NULL);
            await updateRecordAndVerifyType(txn, true, IonTypes.BOOL);
            await updateRecordAndVerifyType(txn, 1, IonTypes.INT);
            await updateRecordAndVerifyType(txn, 3.2, IonTypes.FLOAT);
            await updateRecordAndVerifyType(txn, dom.load("5.5"), IonTypes.DECIMAL);
            await updateRecordAndVerifyType(txn, dom.load("2020-02-02"),
 IonTypes.TIMESTAMP);
            await updateRecordAndVerifyType(txn, dom.load("abc123"),
 IonTypes.SYMBOL);
            await updateRecordAndVerifyType(txn, dom.load("\"string\""),
 IonTypes.STRING);
            await updateRecordAndVerifyType(txn, dom.load("{{ \"clob\" }}"),
 IonTypes.CLOB);
            await updateRecordAndVerifyType(txn, dom.load("{{ blob }}"),
 IonTypes.BLOB);
            await updateRecordAndVerifyType(txn, dom.load("(1 2 3)"),
 IonTypes.SEXP);
            await updateRecordAndVerifyType(txn, dom.load("[1, 2, 3]"),
 IonTypes.LIST);
            await updateRecordAndVerifyType(txn, dom.load("{brand: ford}"),
 IonTypes.STRUCT);
            await deleteTable(txn, TABLE_NAME);
```

```
        });
    } catch (e) {
        error(`Error updating and validating Ion types: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

Python

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy, modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime
from decimal import Decimal
from logging import basicConfig, getLogger, INFO

from amazon.ion.simple_types import IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict,
 IonPyFloat, IonPyInt, IonPyList, \
    IonPyNull, IonPySymbol, IonPyText, IonPyTimestamp
from amazon.ion.simpleion import loads
from amazon.ion.symbols import SymbolToken
from amazon.ion.core import IonType
```

```
from pyqldbsamples.create_table import create_table
from pyqldbsamples.constants import Constants
from pyqldbsamples.insert_document import insert_documents
from pyqldbsamples.model.sample_data import convert_object_to_ion
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

TABLE_NAME = 'IonTypes'


def update_record_and_verify_type(driver, parameter, ion_object, ion_type):
    """
    Update a record in the database table. Then query the value of the record and
 verify correct ion type saved.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to Ion
 for filling in parameters of the
                      statement.

    :type
 ion_object: :py:obj:`IonPyBool`/:py:obj:`IonPyBytes`/:py:obj:`IonPyDecimal`/:py:obj:`IonPyD

 /:py:obj:`IonPyFloat`/:py:obj:`IonPyInt`/:py:obj:`IonPyList`/:py:obj:`IonPyNull`

 /:py:obj:`IonPySymbol`/:py:obj:`IonPyText`/:py:obj:`IonPyTimestamp`
    :param ion_object: The Ion object to verify against.

    :type ion_type: :py:class:`amazon.ion.core.IonType`
    :param ion_type: The Ion type to verify against.

    :raises TypeError: When queried value is not an instance of Ion type.
    """
    update_query = 'UPDATE {} SET Name = ?'.format(TABLE_NAME)
    driver.execute_lambda(lambda executor: executor.execute_statement(update_query,
 parameter))
    logger.info('Updated record.')
```

```
    search_query = 'SELECT VALUE Name FROM {}'.format(TABLE_NAME)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(search_query))

    for c in cursor:
        if not isinstance(c, ion_object):
            raise TypeError('The queried value is not an instance of
 {}'.format(ion_object.__name__))

        if c.ion_type is not ion_type:
            raise TypeError('The queried value type does not match
 {}'.format(ion_type))

    logger.info("Successfully verified value is instance of '{}' with type
 '{}'.".format(ion_object.__name__, ion_type))
    return cursor


def delete_table(driver, table_name):
    """
    Delete a table.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to delete.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Deleting '{}' table...".format(table_name))
    cursor = driver.execute_lambda(lambda executor: executor.execute_statement('DROP
 TABLE {}'.format(table_name)))
    logger.info("'{}' table successfully deleted.".format(table_name))
    return len(list(cursor))


def insert_and_verify_ion_types(driver):
    """
    Insert all the supported Ion types and Python values that are convertible to Ion
 into a ledger and verify that they
    are stored and can be retrieved properly, retaining their original properties.
```

```
    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: A QLDB Driver object.
    """
    python_bytes = str.encode('hello')
    python_bool = True
    python_float = float('0.2')
    python_decimal = Decimal('0.1')
    python_string = "string"
    python_int = 1
    python_null = None
    python_datetime = datetime(2016, 12, 20, 5, 23, 43)
    python_list = [1, 2]
    python_dict = {"brand": "Ford"}

    ion_clob = convert_object_to_ion(loads('{{"This is a CLOB of text."}}'))
    ion_blob = convert_object_to_ion(python_bytes)
    ion_bool = convert_object_to_ion(python_bool)
    ion_decimal = convert_object_to_ion(python_decimal)
    ion_float = convert_object_to_ion(python_float)
    ion_int = convert_object_to_ion(python_int)
    ion_list = convert_object_to_ion(python_list)
    ion_null = convert_object_to_ion(python_null)
    ion_sexp = convert_object_to_ion(loads('(cons 1 2)'))
    ion_string = convert_object_to_ion(python_string)
    ion_struct = convert_object_to_ion(python_dict)
    ion_symbol = convert_object_to_ion(SymbolToken(text='abc', sid=123))
    ion_timestamp = convert_object_to_ion(python_datetime)

    ion_null_clob = convert_object_to_ion(loads('null.clob'))
    ion_null_blob = convert_object_to_ion(loads('null.blob'))
    ion_null_bool = convert_object_to_ion(loads('null.bool'))
    ion_null_decimal = convert_object_to_ion(loads('null.decimal'))
    ion_null_float = convert_object_to_ion(loads('null.float'))
    ion_null_int = convert_object_to_ion(loads('null.int'))
    ion_null_list = convert_object_to_ion(loads('null.list'))
    ion_null_sexp = convert_object_to_ion(loads('null.sexp'))
    ion_null_string = convert_object_to_ion(loads('null.string'))
    ion_null_struct = convert_object_to_ion(loads('null.struct'))
    ion_null_symbol = convert_object_to_ion(loads('null.symbol'))
    ion_null_timestamp = convert_object_to_ion(loads('null.timestamp'))

    create_table(driver, TABLE_NAME)
    insert_documents(driver, TABLE_NAME, [{'Name': 'val'}])
    update_record_and_verify_type(driver, python_bytes, IonPyBytes, IonType.BLOB)
```

```
    update_record_and_verify_type(driver, python_bool, IonPyBool, IonType.BOOL)
    update_record_and_verify_type(driver, python_float, IonPyFloat, IonType.FLOAT)
    update_record_and_verify_type(driver, python_decimal, IonPyDecimal,
IonType.DECIMAL)
    update_record_and_verify_type(driver, python_string, IonPyText, IonType.STRING)
    update_record_and_verify_type(driver, python_int, IonPyInt, IonType.INT)
    update_record_and_verify_type(driver, python_null, IonPyNull, IonType.NULL)
    update_record_and_verify_type(driver, python_datetime, IonPyTimestamp,
IonType.TIMESTAMP)
    update_record_and_verify_type(driver, python_list, IonPyList, IonType.LIST)
    update_record_and_verify_type(driver, python_dict, IonPyDict, IonType.STRUCT)
    update_record_and_verify_type(driver, ion_clob, IonPyBytes, IonType.CLOB)
    update_record_and_verify_type(driver, ion_blob, IonPyBytes, IonType.BLOB)
    update_record_and_verify_type(driver, ion_bool, IonPyBool, IonType.BOOL)
    update_record_and_verify_type(driver, ion_decimal, IonPyDecimal,
IonType.DECIMAL)
    update_record_and_verify_type(driver, ion_float, IonPyFloat, IonType.FLOAT)
    update_record_and_verify_type(driver, ion_int, IonPyInt, IonType.INT)
    update_record_and_verify_type(driver, ion_list, IonPyList, IonType.LIST)
    update_record_and_verify_type(driver, ion_null, IonPyNull, IonType.NULL)
    update_record_and_verify_type(driver, ion_sexp, IonPyList, IonType.SEXP)
    update_record_and_verify_type(driver, ion_string, IonPyText, IonType.STRING)
    update_record_and_verify_type(driver, ion_struct, IonPyDict, IonType.STRUCT)
    update_record_and_verify_type(driver, ion_symbol, IonPySymbol, IonType.SYMBOL)
    update_record_and_verify_type(driver, ion_timestamp, IonPyTimestamp,
IonType.TIMESTAMP)
    update_record_and_verify_type(driver, ion_null_clob, IonPyNull, IonType.CLOB)
    update_record_and_verify_type(driver, ion_null_blob, IonPyNull, IonType.BLOB)
    update_record_and_verify_type(driver, ion_null_bool, IonPyNull, IonType.BOOL)
    update_record_and_verify_type(driver, ion_null_decimal, IonPyNull,
IonType.DECIMAL)
    update_record_and_verify_type(driver, ion_null_float, IonPyNull, IonType.FLOAT)
    update_record_and_verify_type(driver, ion_null_int, IonPyNull, IonType.INT)
    update_record_and_verify_type(driver, ion_null_list, IonPyNull, IonType.LIST)
    update_record_and_verify_type(driver, ion_null_sexp, IonPyNull, IonType.SEXP)
    update_record_and_verify_type(driver, ion_null_string, IonPyNull,
IonType.STRING)
    update_record_and_verify_type(driver, ion_null_struct, IonPyNull,
IonType.STRUCT)
    update_record_and_verify_type(driver, ion_null_symbol, IonPyNull,
IonType.SYMBOL)
    update_record_and_verify_type(driver, ion_null_timestamp, IonPyNull,
IonType.TIMESTAMP)
    delete_table(driver, TABLE_NAME)
```

```
def main(ledger_name=Constants.LEDGER_NAME):
    """
    Insert all the supported Ion types and Python values that are convertible to Ion
 into a ledger and verify that they
    are stored and can be retrieved properly, retaining their original properties.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            insert_and_verify_ion_types(driver)
    except Exception as e:
        logger.exception('Error updating and validating Ion types.')
        raise e


if __name__ == '__main__':
    main()
```

Amazon QLDB API リファレンス

この章では、HTTP、AWS Command Line Interface (AWS CLI)、または AWS SDK を使用してアクセスできる、Amazon QLDB の低レベル API オペレーションについて説明します。

- Amazon QLDB – QLDB リソース管理 API (コントロールプレーンとも呼ばれます)。この API は、台帳リソースの管理および非トランザクションデータオペレーションにのみ使用されます。これらのオペレーションを使用して、台帳を作成、削除、説明表示、一覧表示、更新できます。また、ジャーナルデータを暗号的に検証し、ジャーナルブロックをエクスポートまたはストリーミングすることもできます。
- Amazon QLDB セッション - QLDB トランザクションデータプレーン。この API を PartiQL ステートメントで使用して、台帳のデータトランザクションを実行できます。

> **Important**
>
> QLDB セッション API と直接やり取りする代わりに、QLDB ドライバーまたは QLDB シェルを使用して、台帳のデータトランザクションを実行することをお勧めします。
>
> - AWS SDK を使用している場合は、QLDB ドライバーを使用します。このドライバーは、QLDB セッションデータ API 上に高レベルの抽象化レイヤーを提供し、`SendCommand` オペレーションを管理します。サポートされているプログラミング言語の情報とリストについては、「ドライバーの開始方法」を参照してください。
> - AWS CLI で作業している場合は、QLDB シェルを使用します。シェルは、QLDB ドライバーを使用して台帳と対話するコマンドラインインターフェイスです。詳細については、Amazon QLDB シェルの使用 (データ API のみ)を参照してください。

トピック

アクション

次のアクションが Amazon QLDB でサポートされています。

- CancelJournalKinesisStream
- CreateLedger
- DeleteLedger
- DescribeJournalKinesisStream
- DescribeJournalS3Export
- DescribeLedger
- ExportJournalToS3
- GetBlock
- GetDigest
- GetRevision
- ListJournalKinesisStreamsForLedger
- ListJournalS3Exports
- ListJournalS3ExportsForLedger
- ListLedgers
- ListTagsForResource
- StreamJournalToKinesis
- TagResource
- UntagResource
- UpdateLedger
- UpdateLedgerPermissionsMode

次のアクションが Amazon QLDB セッションでサポートされています。

- SendCommand

Amazon QLDB

次のアクションが Amazon QLDB でサポートされています。

- CancelJournalKinesisStream
- CreateLedger
- DeleteLedger

- DescribeJournalKinesisStream
- DescribeJournalS3Export
- DescribeLedger
- ExportJournalToS3
- GetBlock
- GetDigest
- GetRevision
- ListJournalKinesisStreamsForLedger
- ListJournalS3Exports
- ListJournalS3ExportsForLedger
- ListLedgers
- ListTagsForResource
- StreamJournalToKinesis
- TagResource
- UntagResource
- UpdateLedger
- UpdateLedgerPermissionsMode

CancelJournalKinesisStream

サービス: Amazon QLDB

指定された Amazon QLDB ジャーナルストリームを終了します。ストリームをキャンセルするには、その現在のステータスが ACTIVE である必要があります。

キャンセルしたストリームを再開することはできません。キャンセルされた QLDB ストリームリソースには 7 日間の保持期間が適用されます。したがって、この制限の期限が切れたら自動的に削除されます。

リクエストの構文

```
DELETE /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

streamId

キャンセルされる QLDB ジャーナルストリームの UUID (Base62 でエンコードされたテキストで表されます)。

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "StreamId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

StreamId

キャンセルされた QLDB ジャーナルストリームの UUID (Base62 でエンコードされたテキスト)。

タイプ: 文字列

長さの制限: 22 の固定長

パターン：^[A-Za-z-0-9]+$

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateLedger

サービス: Amazon QLDB

現在のリージョンの AWS アカウント に新しい台帳を作成します。

リクエストの構文

```
POST /ledgers HTTP/1.1
Content-type: application/json

{
   "DeletionProtection": boolean,
   "KmsKey": "string",
   "Name": "string",
   "PermissionsMode": "string",
   "Tags": {
      "string" : "string"
   }
}
```

URI リクエストパラメータ

リクエストでは URI パラメータを使用しません。

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

DeletionProtection

台帳がユーザーによって削除されないように保護されるかどうかを指定します。台帳の作成時に指定しない場合、この機能はデフォルトで有効 `true` になります。

削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。無効にするには、`UpdateLedger` オペレーションを呼び出してパラメータを `false` に設定します。

型: ブール値

必須: いいえ

KmsKey

台帳に保管中のデータの暗号化に使用する AWS Key Management Service（ AWS KMS) のキー。詳細については、Amazon QLDB デベロッパーガイドの保管時の暗号化を参照してください。

次のオプションのいずれかを使用して、このパラメータを指定します。

- `AWS_OWNED_KMS_KEY`: ユーザーに代わって が AWS 所有および管理する AWS KMS キーを使用します。
- 未定義： デフォルトでは、 AWS 所有の KMS キーを使用します。
- 有効な対称カスタマーマネージド KMS キー: 作成、所有、管理しているアカウントで指定した対称暗号 KMS キーを使用します。

  Amazon QLDB は非対称キーをサポートしていません。詳細については、「 AWS Key Management Service デベロッパーガイド」の「対称キーと非対称キーの使用」を参照してください。

カスタマーマネージド KMS キーを指定するには、その KMS キー ID、Amazon リソースネーム (ARN)、エイリアス名、またはエイリアス ARN を使用します。エイリアス名を使用する場合は、`"alias/"` をプレフィックスします。別の でキーを指定するには AWS アカウント、キー ARN またはエイリアス ARN を使用する必要があります。

例:

- キー ID: `1234abcd-12ab-34cd-56ef-1234567890ab`
- キー ARN: `arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`
- エイリアス名: `alias/ExampleAlias`
- エイリアス ARN: `arn:aws:kms:us-east-2:111122223333:alias/ExampleAlias`

詳細については、「 AWS Key Management Service デベロッパーガイド」の「キー識別子 (KeyId) 」を参照してください。

型: 文字列

長さの制限: 最大長は 1600

必須: いいえ

Name

作成する台帳の名前。名前は、現在のリージョンの 内のすべての台帳 AWS アカウント 間で一意である必要があります。

台帳名の命名に関する制約は、「Amazon QLDB デベロッパーガイド」の「Amazon QLDB のクォータ」で定義されています。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

PermissionsMode

作成する台帳に割り当てるアクセス許可モード。このパラメータには以下の値のいずれかがあります。

- `ALLOW_ALL`: 台帳のAPIレベル詳細度でアクセス制御を可能にする、レガシーアクセス許可モード

  このモードは、この台帳の SendCommand API アクセス許可を持つユーザーが、指定された台帳の任意のテーブルで、すべての PartiQL コマンド (すなわち `ALLOW_ALL`) を実行することを許可します。このモードでは、台帳用に作成したテーブルレベルまたはコマンドレベルの IAM アクセス許可ポリシーはすべて無視されます。

- `STANDARD`: (推奨) 台帳、テーブル、PartiQL コマンドの、より詳細なレベルのアクセス許可を可能にするアクセス許可モード。

  デフォルトでは、このモードは、この台帳内の任意のテーブルで、任意の PartiQL コマンドを実行するすべてのユーザーリクエストを拒否します。PartiQL コマンドの実行を許可するには、台帳の SendCommand API アクセス許可に加えて、特定のテーブルリソースと PartiQL アクション用の IAM アクセス許可ポリシーを作成する必要があります。詳細については、Amazon QLDB デベロッパーガイドの Getting started with the standard permissions mode を参照してください。

> **Note**
>
> 台帳データのセキュリティを最大化する、`STANDARD` アクセス許可モードの使用を強く推奨します。

型: 文字列

有効な値：ALLOW_ALL | STANDARD

必須: はい

Tags

作成する台帳にタグとして追加するキーと値のペア。タグキーでは、大文字と小文字が区別されます。タグ値は大文字と小文字が区別され、null にすることができます。

型: 文字列間のマッピング

マップエントリ: 最小数は 0 項目です。最大数は 200 項目です。

キーの長さ制限: 最小長さは 1 です。最大長は 128 です。

値の長さの制限: 最小長は 0。最大長は 256 です。

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "KmsKeyArn": "string",
   "Name": "string",
   "PermissionsMode": "string",
   "State": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Arn

台帳の Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

CreationDateTime

台帳が作成された日時 (エポック時刻形式)。(エポック時間形式は 1970 年 1 月 1 日 12:00:00 AM UTC からの経過秒数です。)

型: タイムスタンプ

DeletionProtection

台帳がユーザーによって削除されないように保護されるかどうかを指定します。台帳の作成時に指定しない場合、この機能はデフォルトで有効 `true` になります。

削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。無効にするには、`UpdateLedger` オペレーションを呼び出してパラメータを `false` に設定します。

型: ブール値

KmsKeyArn

台帳が保管時の暗号化に使用する、カスタマーマネージド KMS キーの ARN。このパラメータが未定義の場合、台帳は暗号化に AWS 所有の KMS キーを使用します。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

Name

台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

パターン: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

作成した台帳のアクセス許可モード。

型: 文字列

有効な値：ALLOW_ALL | STANDARD

State

台帳の現在のステータス。

型: 文字列

有効な値：CREATING | ACTIVE | DELETING | DELETED

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

LimitExceededException

許可されているリソースの最大数の制限に達しました。

HTTP ステータスコード：400

ResourceAlreadyExistsException

指定されたリソースは既に存在します。

HTTP ステータスコード: 409

ResourceInUseException

指定されたリソースは、現時点では変更できません。

HTTP ステータスコード: 409

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteLedger

サービス: Amazon QLDB

台帳とそのすべてのコンテンツを削除します。このアクションを元に戻すことはできません。

削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。無効にするには、UpdateLedger オペレーションを呼び出してパラメータを false に設定します。

リクエストの構文

```
DELETE /ledgers/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

削除する台帳の名前です。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード : 400

ResourceInUseException

指定されたリソースは、現時点では変更できません。

HTTP ステータスコード: 409

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeJournalKinesisStream

サービス: Amazon QLDB

特定の Amazon QLDB ジャーナルストリームに関する詳細情報を返します。出力には、Amazon リソースネーム (ARN)、ストリーム名、現在のステータス、作成時刻、および元のストリーム作成リクエストのパラメータが含まれます。

このアクションでは、有効期限切れのジャーナルストリームは返されません。詳細については、「Amazon QLDB デベロッパーガイド」の「端末ストリームの有効期限」を参照してください。

リクエストの構文

```
GET /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

streamId

記述する QLDB ジャーナルストリームの UUID (Base62 でエンコードされたテキストで表されます)。

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Stream": {
      "Arn": "string",
      "CreationTime": number,
      "ErrorCause": "string",
      "ExclusiveEndTime": number,
      "InclusiveStartTime": number,
      "KinesisConfiguration": {
         "AggregationEnabled": boolean,
         "StreamArn": "string"
      },
      "LedgerName": "string",
      "RoleArn": "string",
      "Status": "string",
      "StreamId": "string",
      "StreamName": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Stream

DescribeJournalS3Export リクエストによって返される QLDB ジャーナルストリームに関する情報。

型: JournalKinesisStreamDescription オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード : 400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeJournalS3Export

サービス: Amazon QLDB

台帳名、エクスポート ID、作成時刻、現在のステータス、および元のエクスポート作成リクエストのパラメータを含む、ジャーナルエクスポートジョブに関する情報を返します。

このアクションでは、有効期限切れのエクスポートジョブは返されません。詳細については、「Amazon QLDB デベロッパーガイド」の「エクスポートジョブの有効期限」を参照してください。

指定された `ExportId` を持つエクスポートジョブが存在しない場合は、`ResourceNotFoundException` をスローします。

指定された `Name` を持つ台帳が存在しない場合は、`ResourceNotFoundException` をスローします。

リクエストの構文

```
GET /ledgers/name/journal-s3-exports/exportId HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

exportId

記述するジャーナルエクスポートジョブの UUID (Base62 でエンコードされたテキストで表されます)。

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須：はい

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "ExportDescription": {
      "ExclusiveEndTime": number,
      "ExportCreationTime": number,
      "ExportId": "string",
      "InclusiveStartTime": number,
      "LedgerName": "string",
      "OutputFormat": "string",
      "RoleArn": "string",
      "S3ExportConfiguration": {
         "Bucket": "string",
         "EncryptionConfiguration": {
            "KmsKeyArn": "string",
            "ObjectEncryptionType": "string"
         },
         "Prefix": "string"
      },
      "Status": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

ExportDescription

DescribeJournalS3Export リクエストによって返されるジャーナルエクスポートジョブに関する情報。

型: JournalS3ExportDescription オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeLedger

サービス: Amazon QLDB

状態、アクセス許可モード、保管時の暗号化の設定、作成日時など、台帳に関する情報を返します。

リクエストの構文

```
GET /ledgers/name HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

記述する台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "EncryptionDescription": {
      "EncryptionStatus": "string",
      "InaccessibleKmsKeyDateTime": number,
      "KmsKeyArn": "string"
   },
   "Name": "string",
   "PermissionsMode": "string",
```

```
   "State": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Arn

台帳の Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

CreationDateTime

台帳が作成された日時 (エポック時刻形式)。(エポック時間形式は 1970 年 1 月 1 日 12:00:00 AM UTC からの経過秒数です。)

型: タイムスタンプ

DeletionProtection

台帳がユーザーによって削除されないように保護されるかどうかを指定します。台帳の作成時に指定しない場合、この機能はデフォルトで有効 `true` になります。

削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。無効にするには、`UpdateLedger` オペレーションを呼び出してパラメータを `false` に設定します。

型: ブール値

EncryptionDescription

台帳の保管中のデータの暗号化に関する情報。これには、現在のステータス、 AWS KMS キー、キーにアクセスできなくなった日時 (エラーの場合) が含まれます。このパラメータが未定義の場合、台帳は暗号化に AWS 所有の KMS キーを使用します。

タイプ : LedgerEncryptionDescription オブジェクト

Name

台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

パターン: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

PermissionsMode

台帳のアクセス許可モード。

型: 文字列

有効な値：ALLOW_ALL | STANDARD

State

台帳の現在のステータス。

型: 文字列

有効な値：CREATING | ACTIVE | DELETING | DELETED

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ExportJournalToS3

サービス: Amazon QLDB

日時範囲内のジャーナルコンテンツを、台帳から指定先の Amazon Simple Storage Service (Amazon S3) バケットにエクスポートします。ジャーナルエクスポートジョブは、Amazon Ion 形式のテキストまたはバイナリ表現、または JSON Lines テキスト形式でデータオブジェクトを書き込むことができます。

指定された `Name` を持つ台帳が存在しない場合は、`ResourceNotFoundException` をスローします。

指定された `Name` を持つ台帳が `CREATING` ステータスの場合は、`ResourcePreconditionNotMetException` をスローします。

各台帳に対して最大 2 つの同時ジャーナルエクスポートリクエストを開始できます。この制限を超えると、ジャーナルエクスポートリクエストで `LimitExceededException` がスローされます。

リクエストの構文

```
POST /ledgers/name/journal-s3-exports HTTP/1.1
Content-type: application/json

{
   "ExclusiveEndTime": number,
   "InclusiveStartTime": number,
   "OutputFormat": "string",
   "RoleArn": "string",
   "S3ExportConfiguration": {
      "Bucket": "string",
      "EncryptionConfiguration": {
         "KmsKeyArn": "string",
         "ObjectEncryptionType": "string"
      },
      "Prefix": "string"
   }
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

ExclusiveEndTime

エクスポートするジャーナルコンテンツの範囲の終了日時 (この日時は含まれません)。

ExclusiveEndTime は、ISO 8601 の日付と時刻の形式および協定世界時 (UTC) である必要があります。例えば、2019-06-13T21:36:34Z です。

ExclusiveEndTime は、協定世界時 (UTC) における現在の日時よりも前の時刻にしてください。

型: タイムスタンプ

必須: はい

InclusiveStartTime

エクスポートするジャーナルコンテンツの範囲の開始日時 (この日時は含まれます)。

InclusiveStartTime は、ISO 8601 の日付と時刻の形式および協定世界時 (UTC) である必要があります。例えば、2019-06-13T21:36:34Z です。

InclusiveStartTime は ExclusiveEndTime より前の日時にする必要があります。

台帳の CreationDateTime より前の InclusiveStartTime を指定した場合、Amazon QLDB のデフォルトは台帳の CreationDateTime になります。

型: タイムスタンプ

必須: はい

OutputFormat

エクスポートするジャーナルデータの出力形式。ジャーナルエクスポートジョブは、Amazon Ion 形式のテキストまたはバイナリ表現、または JSON Lines テキスト形式でデータオブジェクトを書き込むことができます。

デフォルト: `ION_TEXT`

JSON Lines 形式では、エクスポートされたデータオブジェクト内の各ジャーナルブロックは、改行で区切られた有効な JSON オブジェクトです。これらのサービスは改行区切りのJSON を自動的に解析できるため、この形式を使用して、JSON エクスポートを Amazon Athena や AWS Glue などの分析ツールと直接統合できます。

型: 文字列

有効な値：`ION_BINARY | ION_TEXT | JSON`

必須：いいえ

RoleArn

次のことを実行するジャーナルエクスポートジョブに対する QLDB アクセス許可を付与する IAM ロールの Amazon リソースネーム (ARN)。

- Amazon S3 バケットにオブジェクトを書き込みます。
- （オプション) AWS Key Management Service（AWS KMS) のカスタマーマネージドキーを使用して、エクスポートしたデータのサーバー側の暗号化を行います。

ジャーナルエクスポートをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで `iam:PassRole` アクションを実行するためのアクセス許可が必要です。これはすべてのジャーナルエクスポートリクエストに必要です。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

S3ExportConfiguration

エクスポートリクエストの Simple Storage Service (Amazon S3) バケット送信先の構成設定。

型: S3ExportConfiguration オブジェクト

必須: はい

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "ExportId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

ExportId

QLDB が各ジャーナルエクスポートジョブに割り当てる UUID (Base62 でエンコードされたテキストで表されます)。

エクスポートリクエストを記述してジョブのステータスを確認するには、`ExportId` を使用して `DescribeJournalS3Export` を呼び出します。

型: 文字列

長さの制限: 22 の固定長

パターン：`^[A-Za-z-0-9]+$`

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetBlock

サービス: Amazon QLDB

ジャーナルの指定されたアドレスにあるブロックオブジェクトを返します。また、DigestTipAddress が指定されている場合は、指定されたブロックの証明を検証のために返します。

ブロック内のデータコンテンツの詳細については、「Amazon QLDB デベロッパーガイド」の「ジャーナルコンテンツ」を参照してください。

指定された台帳が存在しないか、DELETING ステータスの場合は、ResourceNotFoundException をスローします。

指定された台帳が CREATING ステータスの場合は、ResourcePreconditionNotMetException をスローします。

指定されたアドレスを持つブロックが存在しない場合は、InvalidParameterException をスローします。

リクエストの構文

```
POST /ledgers/name/block HTTP/1.1
Content-type: application/json

{
   "BlockAddress": {
      "IonText": "string"
   },
   "DigestTipAddress": {
      "IonText": "string"
   }
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

BlockAddress

リクエストするブロックの場所。アドレスは、`strandId` と `sequenceNo` という 2 つのフィールドを含む Amazon Ion 構造です。

例: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`。

型: ValueHolder オブジェクト

必須: はい

DigestTipAddress

証明をリクエストするダイジェストの対象となっている最新のブロックの場所。アドレスは、`strandId` と `sequenceNo` という 2 つのフィールドを含む Amazon Ion 構造です。

例: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:49}`。

タイプ : ValueHolder オブジェクト

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Block": {
      "IonText": "string"
   },
   "Proof": {
      "IonText": "string"
   }
```

```
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Block

Amazon Ion 形式のブロックデータオブジェクト。

タイプ：ValueHolder オブジェクト

Proof

GetBlock リクエストによって返される、Amazon Ion 形式の証明オブジェクト。証明には、指定されたブロックから始まるマークルツリーを使用して指定されたダイジェストを再計算するために必要なハッシュ値のリストが含まれます。

型: ValueHolder オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetDigest

サービス: Amazon QLDB

ジャーナル内の最後にコミットされたブロックにある台帳のダイジェストを返します。応答には、256 ビットのハッシュ値とブロックアドレスが含まれます。

リクエストの構文

```
POST /ledgers/name/digest HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Digest": blob,
   "DigestTipAddress": {
      "IonText": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Digest

`GetDigest` リクエストによって返されるダイジェストを表す 256 ビットのハッシュ値。

型: Base64 でエンコードされたバイナリデータオブジェクト

長さの制限: 32 の固定長

DigestTipAddress

リクエストしたダイジェストの対象となっている最新のブロックの場所。アドレスは、`strandId` と `sequenceNo` という 2 つのフィールドを含む Amazon Ion 構造です。

型: ValueHolder オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード : 400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetRevision

サービス: Amazon QLDB

指定されたドキュメント ID とブロックアドレスのリビジョンデータオブジェクトを返します。また、DigestTipAddress が指定されている場合は、指定されたリビジョンの証明を検証のために返します。

リクエストの構文

```
POST /ledgers/name/revision HTTP/1.1
Content-type: application/json

{
   "BlockAddress": {
      "IonText": "string"
   },
   "DigestTipAddress": {
      "IonText": "string"
   },
   "DocumentId": "string"
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

BlockAddress

検証するドキュメントリビジョンのブロックの場所。アドレスは、strandId と sequenceNo という 2 つのフィールドを含む Amazon Ion 構造です。

例: {strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}。

型: ValueHolder オブジェクト

必須: はい

DigestTipAddress

証明をリクエストするダイジェストの対象となっている最新のブロックの場所。アドレスは、strandId と sequenceNo という 2 つのフィールドを含む Amazon Ion 構造です。

例: {strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:49}。

タイプ : ValueHolder オブジェクト

必須: いいえ

DocumentId

検証するドキュメントの UUID (Base62 でエンコードされたテキストで表されます)。

型: 文字列

長さの制限: 22 の固定長

Pattern: ^[A-Za-z-0-9]+$

必須 : はい

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Proof": {
      "IonText": "string"
   },
   "Revision": {
```

```
      "IonText": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Proof

GetRevision リクエストによって返される、Amazon Ion 形式の証明オブジェクト。証明には、指定されたドキュメントリビジョンから始まるマークルツリーを使用して指定されたダイジェストを再計算するために必要なハッシュ値のリストが含まれます。

タイプ：ValueHolder オブジェクト

Revision

Amazon Ion 形式のドキュメントリビジョンのデータオブジェクト。

型: ValueHolder オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListJournalKinesisStreamsForLedger

サービス: Amazon QLDB

指定された台帳のすべての Amazon QLDB ジャーナルストリームストリームを返します。

このアクションでは、有効期限切れのジャーナルストリームは返されません。詳細については、「Amazon QLDB デベロッパーガイド」の「端末ストリームの有効期限」を参照してください。

このアクションは最大 `MaxResults` 個の項目を返します。これは、`ListJournalKinesisStreamsForLedger` を複数回呼び出してすべての項目を取得できるようにページ分割されます。

リクエストの構文

```
GET /ledgers/name/journal-kinesis-streams?max_results=MaxResults&next_token=NextToken
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

MaxResults

1 回の `ListJournalKinesisStreamsForLedger` リクエストで返す結果の最大数。(実際に返される結果の数は少なくなる可能性があります。)

有効範囲: 最小値は 1 です。最大値は 100 です。

NextToken

次の結果ページを取得することを示すページ分割トークン。前の `ListJournalKinesisStreamsForLedger` 呼び出しからのレスポンスで `NextToken` の値を受け取った場合は、入力値として、ここでその値を使用する必要があります。

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン: ^[A-Za-z-0-9+/=]+$

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "NextToken": "string",
   "Streams": [
      {
         "Arn": "string",
         "CreationTime": number,
         "ErrorCause": "string",
         "ExclusiveEndTime": number,
         "InclusiveStartTime": number,
         "KinesisConfiguration": {
            "AggregationEnabled": boolean,
            "StreamArn": "string"
         },
         "LedgerName": "string",
         "RoleArn": "string",
         "Status": "string",
         "StreamId": "string",
         "StreamName": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

NextToken

- NextToken が空の場合は、結果の最後のページが処理されており、それ以上取得する結果はありません。

- NextToken が空でない場合は、さらに結果があります。結果の次のページを取得するには、後続の ListJournalKinesisStreamsForLedger 呼び出しで NextToken の値を使用します。

  型: 文字列

  長さの制限: 最小長は 4 です。最大長は 1,024 です。

  パターン: ^[A-Za-z-0-9+/=]+$

Streams

  指定された台帳に現在関連付けられている QLDB ジャーナルストリーム。

  型: JournalKinesisStreamDescription オブジェクトの配列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

  リクエスト内の 1 つ以上のパラメータが有効ではありません。

  HTTP ステータスコード : 400

ResourceNotFoundException

  指定されたリソースは存在しません。

  HTTP ステータスコード: 404

ResourcePreconditionNotMetException

  事前に条件が満たされていなかったため、オペレーションが失敗しました。

  HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListJournalS3Exports

サービス: Amazon QLDB

現在の AWS アカウント およびリージョンに関連付けられているすべての台帳について、すべてのジャーナルエクスポートジョブを返します。

このアクションは最大 MaxResults 個の項目を返します。これは、ListJournalS3Exports を複数回呼び出してすべての項目を取得できるようにページ分割されます。

このアクションでは、有効期限切れのエクスポートジョブは返されません。詳細については、「Amazon QLDB デベロッパーガイド」の「エクスポートジョブの有効期限」を参照してください。

リクエストの構文

```
GET /journal-s3-exports?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

MaxResults

1 回の ListJournalS3Exports リクエストで返す結果の最大数。(実際に返される結果の数は少なくなる可能性があります。)

有効範囲: 最小値は 1 です。最大値は 100 です。

NextToken

次の結果ページを取得することを示すページ分割トークン。前の ListJournalS3Exports 呼び出しからのレスポンスで NextToken の値を受け取った場合は、入力値として、ここでその値を使用する必要があります。

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン: ^[A-Za-z-0-9+/=]+$

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "JournalS3Exports": [
      {
         "ExclusiveEndTime": number,
         "ExportCreationTime": number,
         "ExportId": "string",
         "InclusiveStartTime": number,
         "LedgerName": "string",
         "OutputFormat": "string",
         "RoleArn": "string",
         "S3ExportConfiguration": {
            "Bucket": "string",
            "EncryptionConfiguration": {
               "KmsKeyArn": "string",
               "ObjectEncryptionType": "string"
            },
            "Prefix": "string"
         },
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

JournalS3Exports

現在の AWS アカウント およびリージョンに関連付けられているすべての台帳についての、すべてのジャーナルエクスポートジョブ。

型: JournalS3ExportDescription オブジェクトの配列

NextToken

- NextToken が空の場合は、結果の最後のページが処理されており、それ以上取得する結果はありません。
- NextToken が空でない場合は、さらに結果があります。結果の次のページを取得するには、後続の ListJournalS3Exports 呼び出しで NextToken の値を使用します。

型: 文字列

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン：^[A-Za-z-0-9+/=]+$

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListJournalS3ExportsForLedger

サービス: Amazon QLDB

指定された台帳について、ジャーナルエクスポートジョブをすべて返します。

このアクションは最大 MaxResults 個の項目を返します。これは、ListJournalS3ExportsForLedger を複数回呼び出してすべての項目を取得できるようにページ分割されます。

このアクションでは、有効期限切れのエクスポートジョブは返されません。詳細については、「Amazon QLDB デベロッパーガイド」の「エクスポートジョブの有効期限」を参照してください。

リクエストの構文

```
GET /ledgers/name/journal-s3-exports?max_results=MaxResults&next_token=NextToken
 HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

MaxResults

1 回の ListJournalS3ExportsForLedger リクエストで返す結果の最大数。(実際に返される結果の数は少なくなる可能性があります。)

有効範囲: 最小値は 1 です。最大値は 100 です。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

必須：はい

NextToken

次の結果ページを取得することを示すページ分割トークン。前の ListJournalS3ExportsForLedger 呼び出しからのレスポンスで NextToken の値を受け取った場合は、入力値として、ここでその値を使用する必要があります。

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン: ^[A-Za-z-0-9+/=]+$

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "JournalS3Exports": [
      {
         "ExclusiveEndTime": number,
         "ExportCreationTime": number,
         "ExportId": "string",
         "InclusiveStartTime": number,
         "LedgerName": "string",
         "OutputFormat": "string",
         "RoleArn": "string",
         "S3ExportConfiguration": {
            "Bucket": "string",
            "EncryptionConfiguration": {
               "KmsKeyArn": "string",
               "ObjectEncryptionType": "string"
            },
            "Prefix": "string"
         },
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

JournalS3Exports

指定された台帳に関連付けられているジャーナルエクスポートジョブ。

型: JournalS3ExportDescription オブジェクトの配列

NextToken

- NextToken が空の場合は、結果の最後のページが処理されており、それ以上取得する結果はありません。
- NextToken が空でない場合は、さらに結果があります。結果の次のページを取得するには、後続の ListJournalS3ExportsForLedger 呼び出しで NextToken の値を使用します。

型: 文字列

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン：^[A-Za-z-0-9+/=]+$

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListLedgers

サービス: Amazon QLDB

現在の AWS アカウント とリージョンに関連付けられているすべての台帳を返します。

このアクションは最大 `MaxResults` 個の項目を返します。これは、`ListLedgers` を複数回呼び出してすべての項目を取得できるようにページ分割されます。

リクエストの構文

```
GET /ledgers?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

MaxResults

1 回の `ListLedgers` リクエストで返す結果の最大数。(実際に返される結果の数は少なくなる可能性があります。)

有効範囲: 最小値は 1 です。最大値は 100 です。

NextToken

次の結果ページを取得することを示すページ分割トークン。前の `ListLedgers` 呼び出しからのレスポンスで `NextToken` の値を受け取った場合は、入力値として、ここでその値を使用する必要があります。

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン: `^[A-Za-z-0-9+/=]+$`

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json
```

```
{
   "Ledgers": [
      {
         "CreationDateTime": number,
         "Name": "string",
         "State": "string"
      }
   ],
   "NextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Ledgers

現在の AWS アカウント およびリージョンに関連付けられている台帳。

型: LedgerSummary オブジェクトの配列

NextToken

さらに結果があるかどうかを示すページ分割トークン。

- NextToken が空の場合は、結果の最後のページが処理されており、それ以上取得する結果はありません。
- NextToken が空でない場合は、さらに結果があります。結果の次のページを取得するには、後続の ListLedgers 呼び出しで NextToken の値を使用します。

型: 文字列

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン：^[A-Za-z-0-9+/=]+$

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTagsForResource

サービス: Amazon QLDB

指定された Amazon QLDB リソースのすべてのタグを返します。

リクエストの構文

```
GET /tags/resourceArn HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

resourceArn

タグを一覧表示する Amazon リソースネーム (ARN)。例:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Tags": {
      "string" : "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Tags

指定された Amazon QLDB リソースに現在関連付けられているタグ。

型: 文字列間のマッピング

マップエントリ: 最小数は 0 項目です。最大数は 200 項目です。

キーの長さ制限: 最小長さは 1 です。最大長は 128 です。

値の長さの制限: 最小長は 0。最大長は 256 です。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード : 400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StreamJournalToKinesis

サービス: Amazon QLDB

特定の Amazon QLDB 台帳のジャーナルストリームを作成します。このストリームは、台帳のジャーナルにコミットされるすべてのドキュメントリビジョンをキャプチャし、指定した Amazon Kinesis Data Streams リソースにそのデータを配信します。

リクエストの構文

```
POST /ledgers/name/journal-kinesis-streams HTTP/1.1
Content-type: application/json

{
   "ExclusiveEndTime": number,
   "InclusiveStartTime": number,
   "KinesisConfiguration": {
      "AggregationEnabled": boolean,
      "StreamArn": "string"
   },
   "RoleArn": "string",
   "StreamName": "string",
   "Tags": {
      "string" : "string"
   }
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

ExclusiveEndTime

ストリームの終了を指定する日時。この日時は範囲内に含まれません。このパラメータを定義しない場合、ストリームはキャンセルするまで無期限に実行されます。

`ExclusiveEndTime` は、`ISO 8601` の日付と時刻の形式および協定世界時 (UTC) である必要があります。例えば、`2019-06-13T21:36:34Z` です。

型: タイムスタンプ

必須: いいえ

InclusiveStartTime

ジャーナルデータのストリーミングを開始する日時。この日時は範囲内に含まれます。このパラメータは、`ISO 8601` の日付と時刻の形式および協定世界時 (UTC) である必要があります。例えば、`2019-06-13T21:36:34Z` です。

`InclusiveStartTime` を将来の日時にすることはできず、`ExclusiveEndTime` より前にする必要があります。

台帳の `CreationDateTime` より前の `InclusiveStartTime` を指定した場合、QLDB のデフォルトは事実上台帳の `CreationDateTime` になります。

型: タイムスタンプ

必須: はい

KinesisConfiguration

ストリームリクエストの Kinesis Data Streams 送信先の構成設定です。

型: KinesisConfiguration オブジェクト

必須: はい

RoleArn

Kinesis Data Streams リソースにデータレコードを書き込むためのジャーナルストリームに対する QLDB アクセス許可を付与する IAM ロールの Amazon リソースネーム (ARN)。

ジャーナルストリームをリクエストするときに QLDB にロールを渡すには、IAM ロールリソースで `iam:PassRole` アクションを実行するためのアクセス許可が必要です。これは、すべてのジャーナルストリームリクエストに必要です。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

StreamName

QLDB ジャーナルストリームに割り当てる名前。ユーザー定義の名前は、ストリームの目的を識別して示すのに役立ちます。

ストリーム名は、特定の台帳の他のアクティブなストリーム間で一意である必要があります。ストリーム名は台帳名と同じ命名の制約があります。この制約は、「Amazon QLDB デベロッパーガイド」の「Amazon QLDB のクォータ」で定義されています。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

Tags

作成するストリームにタグとして追加するキーと値のペア。タグキーでは、大文字と小文字が区別されます。タグ値は大文字と小文字が区別され、null にすることができます。

型: 文字列間のマッピング

マップエントリ: 最小数は 0 項目です。最大数は 200 項目です。

キーの長さ制限: 最小長さは 1 です。最大長は 128 です。

値の長さの制限: 最小長は 0。最大長は 256 です。

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json
```

```
{
   "StreamId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

StreamId

QLDB が各 QLDB ジャーナルストリームに割り当てる UUID (Base62 でエンコードされたテキストで表されます)。

型: 文字列

長さの制限: 22 の固定長

パターン：^[A-Za-z-0-9]+$

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

ResourcePreconditionNotMetException

事前に条件が満たされていなかったため、オペレーションが失敗しました。

HTTP ステータスコード: 412

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

TagResource

サービス: Amazon QLDB

指定された Amazon QLDB リソースに 1 つまたは複数のタグを追加します。

リソースには、最大 50 個のタグを含めることができます。リソースに 50 個を超えるタグを作成しようとすると、リクエストは失敗し、エラーが返されます。

リクエストの構文

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
   "Tags": {
      "string" : "string"
   }
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

resourceArn

タグを追加する Amazon リソースネーム (ARN)。例:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

Tags

指定された QLDB リソースにタグとして追加するキーと値のペア。タグキーでは、大文字と小文字が区別されます。リソースに既に存在するキーを指定すると、リクエストは失敗し、エラーが返されます。タグ値は大文字と小文字が区別され、null にすることができます。

型: 文字列間のマッピング

マップエントリ: 最小数は 0 項目です。最大数は 200 項目です。

キーの長さ制限: 最小長さは 1 です。最大長は 128 です。

値の長さの制限: 最小長は 0。最大長は 256 です。

必須: はい

レスポンスの構文

```
HTTP/1.1 200
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード : 400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET

- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UntagResource

サービス: Amazon QLDB

指定された Amazon QLDB リソースから 1 つまたは複数のタグを削除します。削除するタグキーを 50 個まで指定できます。

リクエストの構文

```
DELETE /tags/resourceArn?tagKeys=TagKeys HTTP/1.1
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

resourceArn

タグを削除する Amazon リソースネーム (ARN)。例:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

TagKeys

削除するタグキーのリスト。

配列メンバー：最小数は 0 項目です。最大数は 200 項目です。

長さの制限：最小長は 1 です。最大長は 128 です。

必須: はい

リクエストボディ

リクエストにリクエスト本文がありません。

レスポンスの構文

```
HTTP/1.1 200
```

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateLedger

サービス: Amazon QLDB

台帳のプロパティを更新します。

リクエストの構文

```
PATCH /ledgers/name HTTP/1.1
Content-type: application/json

{
   "DeletionProtection": boolean,
   "KmsKey": "string"
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

DeletionProtection

台帳がユーザーによって削除されないように保護されるかどうかを指定します。台帳の作成時に指定しない場合、この機能はデフォルトで有効 `true` になります。

削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。無効にするには、`UpdateLedger` オペレーションを呼び出してパラメータを `false` に設定します。

型: ブール値

必須: いいえ

KmsKey

台帳に保管中のデータの暗号化に使用する AWS Key Management Service（ AWS KMS) のキー。詳細については、Amazon QLDB デベロッパーガイドの保管時の暗号化を参照してください。

次のオプションのいずれかを使用して、このパラメータを指定します。

- AWS_OWNED_KMS_KEY: ユーザーに代わって が AWS 所有および管理する AWS KMS キーを使用します。
- 未定義: 台帳の KMS キーは変更されません。
- 有効な対称カスタマーマネージド KMS キー: 作成、所有、管理しているアカウントで指定した対称暗号 KMS キーを使用します。

  Amazon QLDB は非対称キーをサポートしていません。詳細については、「 AWS Key Management Service デベロッパーガイド」の「対称キーと非対称キーの使用」を参照してください。

カスタマーマネージド KMS キーを指定するには、その KMS キー ID、Amazon リソースネーム (ARN)、エイリアス名、またはエイリアス ARN を使用します。エイリアス名を使用する場合は、"alias/" をプレフィックスします。別の でキーを指定するには AWS アカウント、キー ARN またはエイリアス ARN を使用する必要があります。

例:

- キー ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- キー ARN: arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- エイリアス名: alias/ExampleAlias
- エイリアス ARN: arn:aws:kms:us-east-2:111122223333:alias/ExampleAlias

詳細については、「 AWS Key Management Service デベロッパーガイド」の「キー識別子 (KeyId) 」を参照してください。

型: 文字列

長さの制限: 最大長は 1600

必須: いいえ

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "EncryptionDescription": {
      "EncryptionStatus": "string",
      "InaccessibleKmsKeyDateTime": number,
      "KmsKeyArn": "string"
   },
   "Name": "string",
   "State": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Arn

台帳の Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

CreationDateTime

台帳が作成された日時 (エポック時刻形式)。(エポック時間形式は 1970 年 1 月 1 日 12:00:00 AM UTC からの経過秒数です。)

型: タイムスタンプ

DeletionProtection

台帳がユーザーによって削除されないように保護されるかどうかを指定します。台帳の作成時に指定しない場合、この機能はデフォルトで有効 `true` になります。

削除保護が有効になっている場合は、台帳を削除する前に、まず無効にする必要があります。無効にするには、UpdateLedger オペレーションを呼び出してパラメータを false に設定します。

型: ブール値

EncryptionDescription

台帳の保管中のデータの暗号化に関する情報。これには、現在のステータス、 AWS KMS キー、キーにアクセスできなくなった日時 (エラーの場合) が含まれます。

タイプ：LedgerEncryptionDescription オブジェクト

Name

台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

パターン: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

State

台帳の現在のステータス。

型: 文字列

有効な値：CREATING | ACTIVE | DELETING | DELETED

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateLedgerPermissionsMode

サービス: Amazon QLDB

台帳のアクセス許可モードを更新します。

> **Important**
>
> STANDARD アクセス許可モードに切り替える前に、ユーザーの混乱を避けるため、まず必要な IAM ポリシーとテーブルタグをすべて作成する必要があります。詳細については、「Amazon QLDB デベロッパーガイド」の「標準アクセス許可モードへの移行」を参照してください。

リクエストの構文

```
PATCH /ledgers/name/permissions-mode HTTP/1.1
Content-type: application/json

{
   "PermissionsMode": "string"
}
```

URI リクエストパラメータ

リクエストでは、次の URI パラメータを使用します。

name

台帳の名前。

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: はい

リクエストボディ

リクエストは以下の JSON 形式のデータを受け入れます。

PermissionsMode

台帳に割り当てるアクセス許可モード。このパラメータには以下の値のいずれかがあります。

- ALLOW_ALL: 台帳のAPIレベル詳細度でアクセス制御を可能にする、レガシーアクセス許可モード

  このモードは、この台帳の SendCommand API アクセス許可を持つユーザーが、指定された台帳の任意のテーブルで、すべての PartiQL コマンド (すなわち ALLOW_ALL) を実行することを許可します。このモードでは、台帳用に作成したテーブルレベルまたはコマンドレベルの IAM アクセス許可ポリシーはすべて無視されます。

- STANDARD: (推奨) 台帳、テーブル、PartiQL コマンドの、より詳細なレベルのアクセス許可を可能にするアクセス許可モード。

  デフォルトでは、このモードは、この台帳内の任意のテーブルで、任意の PartiQL コマンドを実行するすべてのユーザーリクエストを拒否します。PartiQL コマンドの実行を許可するには、台帳の SendCommand API アクセス許可に加えて、特定のテーブルリソースと PartiQL アクション用の IAM アクセス許可ポリシーを作成する必要があります。詳細については、Amazon QLDB デベロッパーガイドの Getting started with the standard permissions mode を参照してください。

> **Note**
>
> 台帳データのセキュリティを最大化する、STANDARD アクセス許可モードの使用を強く推奨します。

型: 文字列

有効な値 : ALLOW_ALL | STANDARD

必須: はい

レスポンスの構文

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
```

```
    "Name": "string",
    "PermissionsMode": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Arn

台帳の Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

Name

台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

パターン: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

台帳の現在のアクセス許可モード。

型: 文字列

有効な値：`ALLOW_ALL | STANDARD`

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidParameterException

リクエスト内の 1 つ以上のパラメータが有効ではありません。

HTTP ステータスコード：400

ResourceNotFoundException

指定されたリソースは存在しません。

HTTP ステータスコード: 404

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Amazon QLDB セッション

次のアクションが Amazon QLDB セッションでサポートされています。

- SendCommand

SendCommand

サービス: Amazon QLDB Session

Amazon QLDB 台帳にコマンドを送信します。

> **Note**
>
> この API と直接やり取りする代わりに、QLDB ドライバーまたは QLDB シェルを使用して、台帳のデータトランザクションを実行することをお勧めします。
>
> - AWS SDK を使用している場合は、QLDB ドライバーを使用します。このドライバーは、この QLDB セッションデータ API 上に高レベルの抽象化レイヤーを提供し、SendCommand オペレーションを管理します。サポートされているプログラミング言語の詳細およびリストについては、「Amazon QLDB デベロッパーガイド」の「ドライバーの開始方法」を参照してください。
> - AWS Command Line Interface（ AWS CLI) を使用している場合は、QLDB シェルを使用します。シェルは、QLDB ドライバーを使用して台帳と対話するコマンドラインインターフェイスです。詳細については、「QLDB シェルを使用した Amazon QLDB へのアクセス」を参照してください。

リクエストの構文

```
{
   "AbortTransaction": {
   },
   "CommitTransaction": {
      "CommitDigest": blob,
      "TransactionId": "string"
   },
   "EndSession": {
   },
   "ExecuteStatement": {
      "Parameters": [
         {
            "IonBinary": blob,
            "IonText": "string"
         }
      ],
      "Statement": "string",
```

```
      "TransactionId": "string"
   },
   "FetchPage": {
      "NextPageToken": "string",
      "TransactionId": "string"
   },
   "SessionToken": "string",
   "StartSession": {
      "LedgerName": "string"
   },
   "StartTransaction": {
   }
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは以下の JSON 形式のデータを受け入れます。

AbortTransaction

現在のトランザクションを中断するコマンド。

タイプ：AbortTransactionRequest オブジェクト

必須: いいえ

CommitTransaction

指定されたトランザクションをコミットするコマンド。

タイプ：CommitTransactionRequest オブジェクト

必須: いいえ

EndSession

現在のセッションを終了するコマンド。

タイプ：EndSessionRequest オブジェクト

必須: いいえ

ExecuteStatement

指定されたトランザクションでステートメントを実行するコマンド。

タイプ：ExecuteStatementRequest オブジェクト

必須: いいえ

FetchPage

ページを取得するコマンド。

タイプ：FetchPageRequest オブジェクト

必須: いいえ

SessionToken

現在のコマンドのセッショントークンを指定します。セッショントークンは、セッションの有効期間を通して不変です。

セッショントークンを取得するには、StartSession コマンドを実行します。この SessionToken は、現在のセッション中に発行される後続のすべてのコマンドに必要です。

型: 文字列

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン: ^[A-Za-z-0-9+/=]+$

必須: いいえ

StartSession

新しいセッションを開始するコマンド。セッショントークンは、レスポンスの一部として取得されます。

タイプ：StartSessionRequest オブジェクト

必須: いいえ

StartTransaction

新しいトランザクションを開始するコマンド。

タイプ : StartTransactionRequest オブジェクト

必須: いいえ

レスポンスの構文

```
{
   "AbortTransaction": {
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      }
   },
   "CommitTransaction": {
      "CommitDigest": blob,
      "ConsumedIOs": {
         "ReadIOs": number,
         "WriteIOs": number
      },
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      },
      "TransactionId": "string"
   },
   "EndSession": {
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      }
   },
   "ExecuteStatement": {
      "ConsumedIOs": {
         "ReadIOs": number,
         "WriteIOs": number
      },
      "FirstPage": {
         "NextPageToken": "string",
         "Values": [
            {
               "IonBinary": blob,
               "IonText": "string"
            }
         ]
      },
      "TimingInformation": {
```

```
            "ProcessingTimeMilliseconds": number
         }
      },
      "FetchPage": {
         "ConsumedIOs": {
            "ReadIOs": number,
            "WriteIOs": number
         },
         "Page": {
            "NextPageToken": "string",
            "Values": [
               {
                  "IonBinary": blob,
                  "IonText": "string"
               }
            ]
         },
         "TimingInformation": {
            "ProcessingTimeMilliseconds": number
         }
      },
      "StartSession": {
         "SessionToken": "string",
         "TimingInformation": {
            "ProcessingTimeMilliseconds": number
         }
      },
      "StartTransaction": {
         "TimingInformation": {
            "ProcessingTimeMilliseconds": number
         },
         "TransactionId": "string"
      }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

AbortTransaction

中断されたトランザクションの詳細が含まれています。

タイプ：AbortTransactionResult オブジェクト

CommitTransaction

コミットされたトランザクションの詳細が含まれています。

タイプ：CommitTransactionResult オブジェクト

EndSession

終了したセッションの詳細が含まれています。

タイプ：EndSessionResult オブジェクト

ExecuteStatement

実行されたステートメントの詳細が含まれています。

タイプ：ExecuteStatementResult オブジェクト

FetchPage

取得されたページの詳細が含まれています。

タイプ：FetchPageResult オブジェクト

StartSession

セッショントークンを含む、開始されたセッションの詳細が含まれています。この `SessionToken` は、現在のセッション中に発行される後続のすべてのコマンドに必要です。

タイプ：StartSessionResult オブジェクト

StartTransaction

開始されたトランザクションの詳細が含まれています。

型: StartTransactionResult オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

BadRequestException

リクエストの形式が正しくない場合や、無効なパラメータ値や必須パラメータの欠落などのエラーが含まれている場合に返されます。

HTTP ステータスコード：400

CapacityExceededException

リクエストが台帳の処理能力を超えたときに返されます。

HTTP ステータスコード：400

InvalidSessionException

タイムアウトしたか有効期限が切れたために、セッションがこれ以上存在しなくなった場合に返されます。

HTTP ステータスコード：400

LimitExceededException

アクティブセッション数などのリソース制限を超えた場合に返されます。

HTTP ステータスコード：400

OccConflictException

オプティミスティック同時実行制御 (OCC) の検証フェーズで生じた障害によって、トランザクションをジャーナルに書き込めない場合に返されます。

HTTP ステータスコード：400

RateExceededException

リクエストの割合が、許可されているスループットを超えた場合に返されます。

HTTP ステータスコード：400

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2

- AWS SDK for JavaScript V3
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

データ型

Amazon QLDB では以下のデータ型がサポートされています。

- JournalKinesisStreamDescription
- JournalS3ExportDescription
- KinesisConfiguration
- LedgerEncryptionDescription
- LedgerSummary
- S3EncryptionConfiguration
- S3ExportConfiguration
- ValueHolder

Amazon QLDB セッションでは以下のデータ型がサポートされています。

- AbortTransactionRequest
- AbortTransactionResult
- CommitTransactionRequest
- CommitTransactionResult
- EndSessionRequest
- EndSessionResult
- ExecuteStatementRequest
- ExecuteStatementResult
- FetchPageRequest
- FetchPageResult
- IOUsage
- Page

- StartSessionRequest
- StartSessionResult
- StartTransactionRequest
- StartTransactionResult
- TimingInformation
- ValueHolder

Amazon QLDB

Amazon QLDB では以下のデータ型がサポートされています。

- JournalKinesisStreamDescription
- JournalS3ExportDescription
- KinesisConfiguration
- LedgerEncryptionDescription
- LedgerSummary
- S3EncryptionConfiguration
- S3ExportConfiguration
- ValueHolder

JournalKinesisStreamDescription

サービス: Amazon QLDB

Amazon QLDB ジャーナルストリームに関する情報。これには、Amazon リソースネーム (ARN)、ストリーム名、作成時刻、現在のステータス、および元のストリーム作成リクエストのパラメータが含まれます。

内容

KinesisConfiguration

QLDB ジャーナルストリームの Amazon Kinesis Data Streams ターゲットの構成設定。

型: KinesisConfiguration オブジェクト

必須: はい

LedgerName

台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

RoleArn

Kinesis Data Streams リソースにデータレコードを書き込むためのジャーナルストリームに対する QLDB アクセス許可を付与する IAM ロールの Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

Status

QLDB ジャーナルストリームの現在の状態。

型: 文字列

有効な値：ACTIVE | COMPLETED | CANCELED | FAILED | IMPAIRED

必須: はい

StreamId

QLDB ジャーナルストリームの UUID (Base62 でエンコードされたテキストで表されます)。

型: 文字列

長さの制限: 22 の固定長

Pattern: ^[A-Za-z-0-9]+$

必須：はい

StreamName

QLDB ジャーナルストリームのユーザー定義名。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

必須：はい

Arn

QLDB ジャーナルストリームの Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: いいえ

CreationTime

QLDB ジャーナルストリームが作成された日時 (エポック時刻形式)。(エポック時間形式は 1970 年 1 月 1 日 12:00:00 AM UTC からの経過秒数です。)

型: タイムスタンプ

必須: いいえ

ErrorCause

ストリームのステータスが IMPAIRED または FAILED である理由を説明するエラーメッセージ。これは、他のステータス値を持つストリームには適用されません。

型: 文字列

有効な値：KINESIS_STREAM_NOT_FOUND | IAM_PERMISSION_REVOKED

必須：いいえ

ExclusiveEndTime

ストリームの終了を指定する日時。この日時は範囲内に含まれません。このパラメータが未定義の場合、ストリームはキャンセルするまで無期限に実行されます。

型: タイムスタンプ

必須: いいえ

InclusiveStartTime

ジャーナルデータのストリーミングを開始する日時。この日時は範囲内に含まれます。

型: タイムスタンプ

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

JournalS3ExportDescription

サービス: Amazon QLDB

台帳名、エクスポート ID、作成時刻、現在のステータス、および元のエクスポート作成リクエストのパラメータを含む、ジャーナルエクスポートジョブに関する情報。

内容

ExclusiveEndTime

元のエクスポートリクエストで指定された、ジャーナルコンテンツの範囲の終了日時 (この日時は含まれません)。

型: タイムスタンプ

必須: はい

ExportCreationTime

エクスポートジョブが作成された日時 (エポック時刻形式)。(エポック時間形式は 1970 年 1 月 1 日 12:00:00 AM UTC からの経過秒数です。)

型: タイムスタンプ

必須: はい

ExportId

ジャーナルエクスポートジョブの UUID (Base62 でエンコードされたテキストで表されます)。

型: 文字列

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須：はい

InclusiveStartTime

元のエクスポートリクエストで指定された、ジャーナルコンテンツの範囲の開始日時 (この日時は含まれます)。

型: タイムスタンプ

必須: はい

LedgerName

台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

RoleArn

次のことを実行するジャーナルエクスポートジョブに対する QLDB アクセス許可を付与する IAM ロールの Amazon リソースネーム (ARN)。

- Amazon Simple Storage Service (Amazon S3) バケットにオブジェクトを書き込む。
- （オプション) AWS Key Management Service（AWS KMS) のカスタマーマネージドキーを使用して、エクスポートしたデータのサーバー側の暗号化を行います。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

S3ExportConfiguration

ジャーナルエクスポートジョブでジャーナルコンテンツを書き込む先の Amazon Simple Storage Service (Amazon S3) バケットの場所。

型: S3ExportConfiguration オブジェクト

必須: はい

Status

ジャーナルエクスポートジョブの現在の状態。

型: 文字列

有効な値：`IN_PROGRESS | COMPLETED | CANCELLED`

必須: はい

OutputFormat

エクスポートするジャーナルデータの出力形式。

型: 文字列

有効な値：`ION_BINARY | ION_TEXT | JSON`

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisConfiguration

サービス: Amazon QLDB

Amazon QLDB ジャーナルストリームの Amazon Kinesis Data Streams 送信先の構成です。

内容

StreamArn

Kinesis Data Streams リソースの Amazon リソースネーム (ARN)。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

AggregationEnabled

QLDB が 1 つの Kinesis Data Streams レコードで複数のデータレコードを発行できるようにします。これにより、API 呼び出しごとに送信されるレコードの数が増加します。

デフォルト: `True`

> **Important**
>
> レコード集約は、レコードの処理に重要な意味を持ち、ストリーミングコンシューマーで集約を解除する必要があります。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「KPL の主要な概念」および「コンシューマーの集約解除」を参照してください。

型: ブール値

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++

- AWS SDK for Java V2
- AWS SDK for Ruby V3

LedgerEncryptionDescription

サービス: Amazon QLDB

Amazon QLDB 台帳の保管中のデータの暗号化に関する情報。これには、現在のステータス、 AWS Key Management Service （ AWS KMS) のキー、キーにアクセスできなくなった日時 (エラーの場合) が含まれます。

詳細については、Amazon QLDB デベロッパーガイドの保管時の暗号化を参照してください。

内容

EncryptionStatus

台帳の保管時の暗号化の現在の状態。これは、以下の値のいずれかになります。

- ENABLED: 暗号化は、指定されたキーを使用して完全に有効になります。
- UPDATING: 台帳は、指定されたキーの変更をアクティブに処理しています。

  QLDB のキーの変更は非同期です。キーの変更が処理されている間も、パフォーマンスに影響を与えることなく、台帳にフルアクセスできます。キーの更新にかかる時間は、台帳のサイズによって異なります。
- KMS_KEY_INACCESSIBLE: 指定されたカスタマーマネージド KMS キーにアクセスできず、台帳に障害が発生しています。キーが無効化または削除されたか、キーの許可が取り消されました。台帳に障害が発生すると、アクセスできず、読み取りまたは書き込みリクエストも受け付けません。

  障害のある台帳は、キーの許可を復元した後、または無効化されたキーを再度有効にした後に、自動的にアクティブ状態に戻ります。ただし、カスタマーマネージド KMS キーの削除は元に戻せません。キーを削除すると、そのキーで保護されている台帳にアクセスできなくなり、データが完全に回復不可能になります。

型: 文字列

有効な値 : ENABLED | UPDATING | KMS_KEY_INACCESSIBLE

必須: はい

KmsKeyArn

台帳が保管時の暗号化に使用する、カスタマーマネージド KMS キーの Amazon リソースネーム (ARN)。このパラメータが未定義の場合、台帳は暗号化に AWS 所有の KMS キーを使用します。台帳の暗号化設定を AWS 所有の KMS キーに更新AWS_OWNED_KMS_KEYすると表示されます。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須: はい

InaccessibleKmsKeyDateTime

エラーが発生した場合にキーが AWS KMS 最初にアクセス不能になった日時をエポック時間形式で表します。(エポック時間形式は 1970 年 1 月 1 日 12:00:00 AM UTC からの経過秒数です。)

AWS KMS キーがアクセス可能な場合、このパラメータは未定義です。

型: タイムスタンプ

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LedgerSummary

サービス: Amazon QLDB

名前、状態、作成日時など、台帳に関する情報。

内容

CreationDateTime

台帳が作成された日時 (エポック時刻形式)。(エポック時間形式は 1970 年 1 月 1 日 12:00:00 AM UTC からの経過秒数です。)

型: タイムスタンプ

必須: いいえ

Name

台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

パターン: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須: いいえ

State

台帳の現在のステータス。

型: 文字列

有効な値：`CREATING | ACTIVE | DELETING | DELETED`

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++

- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3EncryptionConfiguration

サービス: Amazon QLDB

Amazon Simple Storage Service (Amazon S3) バケットにデータを書き込むためにジャーナルエクスポートジョブで使用される暗号化設定。

内容

ObjectEncryptionType

Simple Storage Service (Amazon S3) オブジェクトの暗号化タイプ

Simple Storage Service (Amazon S3) のサーバー側の暗号化オプションの詳細については、「Amazon S3 デベロッパーガイド」の「サーバー側の暗号化を使用したデータの保護」を参照してください。

型: 文字列

有効な値：`SSE_KMS | SSE_S3 | NO_ENCRYPTION`

必須: はい

KmsKeyArn

AWS Key Management Service () の対称暗号化キーの Amazon リソースネーム (ARN)AWS KMS。Amazon S3 は非対称 KMS キーをサポートしていません。

`ObjectEncryptionType` として `SSE_KMS` を指定する場合は、`KmsKeyArn` を指定する必要があります。

`ObjectEncryptionType` として `SSE_S3` を指定する場合は、`KmsKeyArn` は必要ありません。

型: 文字列

長さの制限: 最小長は 20 です。最大長は 1600 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3ExportConfiguration

サービス: Amazon QLDB

ジャーナルエクスポートジョブでジャーナルコンテンツを書き込む先の Amazon Simple Storage Service (Amazon S3) バケットの場所。

内容

Bucket

ジャーナルエクスポートジョブでジャーナルコンテンツを書き込む先の Simple Storage Service (Amazon S3) バケットの名前。

バケット名は、Simple Storage Service (Amazon S3) バケットの命名規則に準拠している必要があります。詳細については、「Amazon S3 デベロッパーガイド」の「バケットの制約と制限」を参照してください。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 255 です。

パターン: `^[A-Za-z-0-9-_.]+$`

必須：はい

EncryptionConfiguration

Simple Storage Service (Amazon S3) バケットにデータを書き込むためにジャーナルエクスポートジョブで使用される暗号化設定。

型: S3EncryptionConfiguration オブジェクト

必須: はい

Prefix

ジャーナルエクスポートジョブでジャーナルコンテンツを書き込む先の Simple Storage Service (Amazon S3) バケットのプレフィックス。

プレフィックスは、Simple Storage Service (Amazon S3) キーの命名規則と制限に準拠している必要があります。詳細については、「Amazon S3 デベロッパーガイド」の「オブジェクトキーとメタデータ」を参照してください。

次に示すのは、`Prefix` の有効な値の例です。

- `JournalExports-ForMyLedger/Testing/`
- `JournalExports`
- `My:Tests/`

型: 文字列

長さの制限: 最小長は 0 です。最大長は 128 です。

必須：はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ValueHolder

サービス: Amazon QLDB

複数のエンコード形式の値を含めることができる構造。

内容

IonText

`ValueHolder` 構造に含まれる Amazon Ion のプレーンテキスト値。

型: 文字列

長さの制限：最小長は 1 です。最大長は 1048576 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Amazon QLDB セッション

Amazon QLDB セッションでは以下のデータ型がサポートされています。

- AbortTransactionRequest
- AbortTransactionResult
- CommitTransactionRequest
- CommitTransactionResult
- EndSessionRequest
- EndSessionResult
- ExecuteStatementRequest
- ExecuteStatementResult

- FetchPageRequest
- FetchPageResult
- IOUsage
- Page
- StartSessionRequest
- StartSessionResult
- StartTransactionRequest
- StartTransactionResult
- TimingInformation
- ValueHolder

AbortTransactionRequest

サービス: Amazon QLDB Session

中断するトランザクションの詳細が含まれています。

内容

この例外構造のメンバーは、コンテキストに依存します。

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

AbortTransactionResult

サービス: Amazon QLDB Session

中断されたトランザクションの詳細が含まれています。

内容

TimingInformation

コマンドのサーバー側のパフォーマンス情報が含まれています。

タイプ：TimingInformation オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CommitTransactionRequest

サービス: Amazon QLDB Session

コミットするトランザクションの詳細が含まれています。

内容

CommitDigest

コミットするトランザクションのコミットダイジェストを指定します。アクティブなトランザクションごとに、コミットダイジェストを渡す必要があります。QLDB は `CommitDigest` を検証し、クライアントで計算されたダイジェストが QLDB によって計算されたダイジェストと一致しない場合、エラーでコミットを拒否します。

`CommitDigest` パラメータの目的は、クライアントが送信したステートメントの正確なセットを、クライアントが送信したのと同じ順序で、かつ重複しないように、サーバーが処理した場合およびその場合に限って、QLDB がトランザクションをコミットするようにすることです。

型: Base64 でエンコードされたバイナリデータオブジェクト

必須: はい

TransactionId

コミットするトランザクションのトランザクション ID を指定します。

型: 文字列

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須：はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CommitTransactionResult

サービス: Amazon QLDB Session

コミットされたトランザクションの詳細が含まれています。

内容

CommitDigest

コミットされたトランザクションのコミットダイジェスト。

型: Base64 でエンコードされたバイナリデータオブジェクト

必須: いいえ

ConsumedIOs

消費された I/O リクエストの数に関するメトリクスが含まれています。

タイプ : IOUsage オブジェクト

必須: いいえ

TimingInformation

コマンドのサーバー側のパフォーマンス情報が含まれています。

タイプ : TimingInformation オブジェクト

必須: いいえ

TransactionId

コミットされたトランザクションのトランザクション ID。

型: 文字列

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

EndSessionRequest

サービス: Amazon QLDB Session

セッションを終了するリクエストを指定します。

内容

この例外構造のメンバーは、コンテキストに依存します。

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

EndSessionResult

サービス: Amazon QLDB Session

終了したセッションの詳細が含まれています。

内容

TimingInformation

コマンドのサーバー側のパフォーマンス情報が含まれています。

タイプ：TimingInformation オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ExecuteStatementRequest

サービス: Amazon QLDB Session

ステートメントを実行するリクエストを指定します。

内容

Statement

リクエストのステートメントを指定します。

型: 文字列

長さの制限：最小長は 1 です。最大長 100,000。

必須: はい

TransactionId

リクエストのトランザクション ID を指定します。

型: 文字列

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須：はい

Parameters

リクエスト内のパラメータ化されたステートメントのパラメータを指定します。

型: ValueHolder オブジェクトの配列

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

ExecuteStatementResult

サービス: Amazon QLDB Session

実行されたステートメントの詳細が含まれています。

内容

ConsumedIOs

消費された I/O リクエストの数に関するメトリクスが含まれています。

タイプ：IOUsage オブジェクト

必須: いいえ

FirstPage

最初に取得されたページの詳細が含まれています。

タイプ：Page オブジェクト

必須: いいえ

TimingInformation

コマンドのサーバー側のパフォーマンス情報が含まれています。

タイプ：TimingInformation オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

FetchPageRequest

サービス: Amazon QLDB Session

取得するページの詳細を指定します。

内容

NextPageToken

取得するページの次のページのトークンを指定します。

型: 文字列

長さの制限: 最小長は 4 です。最大長は 1,024 です。

Pattern: `^[A-Za-z-0-9+/=]+$`

必須：はい

TransactionId

取得するページのトランザクション ID を指定します。

型: 文字列

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須：はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

FetchPageResult

サービス: Amazon QLDB Session

取得されたページが含まれています。

内容

ConsumedIOs

消費された I/O リクエストの数に関するメトリクスが含まれています。

タイプ：IOUsage オブジェクト

必須: いいえ

Page

取得されたページの詳細が含まれています。

タイプ：Page オブジェクト

必須: いいえ

TimingInformation

コマンドのサーバー側のパフォーマンス情報が含まれています。

タイプ：TimingInformation オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

IOUsage

サービス: Amazon QLDB Session

呼び出されたコマンドの I/O 使用状況メトリクスが含まれています。

内容

ReadIOs

コマンドが実行した読み取り I/O リクエストの数。

型: Long

必須: いいえ

WriteIOs

コマンドが実行した書き込み I/O リクエストの数。

型: Long

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Page

サービス: Amazon QLDB Session

取得されたページの詳細が含まれています。

内容

NextPageToken

次のページのトークン。

型: 文字列

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン: `^[A-Za-z-0-9+/=]+$`

必須: いいえ

Values

複数のエンコード形式の値が含まれる構造。

型: ValueHolder オブジェクトの配列

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

StartSessionRequest

サービス: Amazon QLDB Session

新しいセッションを開始するリクエストを指定します。

内容

LedgerName

新しいセッションを開始する台帳の名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 32 です。

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

必須：はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

StartSessionResult

サービス: Amazon QLDB Session

開始されたセッションの詳細が含まれています。

内容

SessionToken

開始されたセッションのセッショントークン。この SessionToken は、現在のセッション中に発行される後続のすべてのコマンドに必要です。

型: 文字列

長さの制限: 最小長は 4 です。最大長は 1,024 です。

パターン: ^[A-Za-z-0-9+/=]+$

必須: いいえ

TimingInformation

コマンドのサーバー側のパフォーマンス情報が含まれています。

タイプ : TimingInformation オブジェクト

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

StartTransactionRequest

サービス: Amazon QLDB Session

トランザクションを開始するリクエストを指定します。

内容

この例外構造のメンバーは、コンテキストに依存します。

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

StartTransactionResult

サービス: Amazon QLDB Session

開始されたトランザクションの詳細が含まれています。

内容

TimingInformation

コマンドのサーバー側のパフォーマンス情報が含まれています。

タイプ：TimingInformation オブジェクト

必須: いいえ

TransactionId

開始されたトランザクションのトランザクション ID。

型: 文字列

長さの制限: 22 の固定長

Pattern: `^[A-Za-z-0-9]+$`

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

TimingInformation

サービス: Amazon QLDB Session

コマンドのサーバー側のパフォーマンス情報が含まれています。Amazon QLDB は、リクエストを受信してから対応するレスポンスを送信するまでの間のタイミング情報をキャプチャします。

内容

ProcessingTimeMilliseconds

QLDB がコマンドの処理に費やした時間 (ミリ秒単位)。

型: Long

必須: いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ValueHolder

サービス: Amazon QLDB Session

複数のエンコード形式の値を含めることができる構造。

内容

IonBinary

`ValueHolder` 構造に含まれる Amazon Ion のバイナリ値。

型: Base64 でエンコードされたバイナリデータオブジェクト

長さの制限：最小長は 1 です。最大長は 131072 です。

必須: いいえ

IonText

`ValueHolder` 構造に含まれる Amazon Ion のプレーンテキスト値。

型: 文字列

長さの制限：最小長は 1 です。最大長は 1048576 です。

必須：いいえ

その他の参照資料

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

共通エラー

このセクションでは、AWS のすべてのサービスの API アクションに共通のエラーを一覧表示しています。このサービスの API アクションに固有のエラーについては、その API アクションのトピックを参照してください。

AccessDeniedException

このアクションを実行する十分なアクセス権限がありません。

HTTP ステータスコード: 400

IncompleteSignature

リクエストの署名が AWS 基準に適合しません。

HTTP ステータスコード: 400

InternalFailure

リクエストの処理が、不明なエラー、例外、または障害により実行できませんでした。

HTTP ステータスコード: 500

InvalidAction

リクエストされたアクション、またはオペレーションは無効です。アクションが正しく入力されていることを確認します。

HTTP ステータスコード: 400

InvalidClientTokenId

指定された x.509 証明書、または AWS アクセスキー ID が見つかりません。

HTTP ステータスコード: 403

NotAuthorized

このアクションを実行するにはアクセス許可が必要です。

HTTP ステータスコード: 400

OptInRequired

サービスを利用するためには、AWS アクセスキー ID を取得する必要があります。

HTTP ステータスコード: 403

RequestExpired

リクエストの日付スタンプの 15 分以上後またはリクエストの有効期限 (署名付き URL の場合など) の 15 分以上後に、リクエストが到着しました。または、リクエストの日付スタンプが現在より 15 分以上先です。

HTTP ステータスコード: 400

ServiceUnavailable

リクエストは、サーバーの一時的障害のために実行に失敗しました。

HTTP ステータスコード: 503

ThrottlingException

リクエストは、制限が必要なために実行が拒否されました。

HTTP ステータスコード: 400

ValidationError

入力が、AWS サービスで指定された制約を満たしていません。

HTTP ステータスコード: 400

共通パラメータ

次のリストには、すべてのアクションが署名バージョン 4 リクエストにクエリ文字列で署名するために使用するパラメータを示します。アクション固有のパラメータは、アクションのトピックに示されています。Signature Version 4 の詳細については、「IAM ユーザーガイド」の「AWS API リクエストの署名」を参照してください。

Action

実行するアクション。

型: 文字列

必須: はい

Version

リクエストが想定している API バージョンである、YYYY-MM-DD 形式で表示されます。

型: 文字列

必須: はい

X-Amz-Algorithm

リクエストの署名を作成するのに使用したハッシュアルゴリズム。

条件: HTTP 認証ヘッダーではなくクエリ文字列に認証情報を含める場合は、このパラメータを指定します。

型: 文字列

有効な値: `AWS4-HMAC-SHA256`

必須: 条件による

X-Amz-Credential

認証情報スコープの値で、アクセスキー、日付、対象とするリージョン、リクエストしているサービス、および終了文字列 ("aws4_request") を含む文字列です。値は次の形式で表現されます。[access_key/YYYYYYYYMMDD/リージョン/サービス/aws4_request]

詳細については、「IAM ユーザーガイド」の「署名付きAWS API リクエストの作成」を参照してください。

条件: HTTP 認証ヘッダーではなくクエリ文字列に認証情報を含める場合は、このパラメータを指定します。

型: 文字列

必須: 条件による

X-Amz-Date

署名を作成するときに使用する日付です。形式は ISO 8601 基本形式の YYYYMMDD'T'HHMMSS'Z' でなければなりません。例えば、日付 `20120325T120000Z` は、有効な X-Amz-Date の値です。

条件: X-Amz-Date はすべてのリクエストに対してオプションです。署名リクエストで使用する日付よりも優先される日付として使用できます。ISO 8601 ベーシック形式で日付ヘッダーが指定されている場合、X-Amz-Date は必要ありません。X-Amz-Date を使用すると、常に Date ヘッダーの値よりも優先されます。詳細については、「IAM ユーザーガイド」の「AWS API リクエスト署名の要素」を参照してください。

タイプ: 文字列

必須: 条件による

X-Amz-Security-Token

AWS Security Token Service (AWS STS) へのを呼び出しで取得された一時的なセキュリティトークン。AWS STS の一時的なセキュリティ認証情報をサポートするサービスのリストについては、「IAM ユーザーガイド」の「IAM と連携するAWS のサービス」を参照してください。

条件:AWS STS の一時的なセキュリティ認証情報を使用する場合、セキュリティトークンを含める必要があります。

タイプ: 文字列

必須: 条件による

X-Amz-Signature

署名する文字列と派生署名キーから計算された 16 進符号化署名を指定します。

条件: HTTP 認証ヘッダーではなくクエリ文字列に認証情報を含める場合は、このパラメータを指定します。

型: 文字列

必須: 条件による

X-Amz-SignedHeaders

正規リクエストの一部として含まれていたすべての HTTP ヘッダーを指定します。署名付きヘッダーの指定に関する詳細については、「IAM ユーザーガイド」の「署名付き AWS API リクエストの作成」を参照してください。

条件: HTTP 認証ヘッダーではなくクエリ文字列に認証情報を含める場合は、このパラメータを指定します。

型: 文字列

必須: 条件による

Amazon QLDB でのクォータと制限

このセクションでは、Amazon QLDB の現在のクォータ (制限とも呼ばれる) について説明します。

トピック

デフォルトのクォータ

QLDB には、「AWS 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」にも示すように、以下のデフォルトのクォータがあります。これらのクォータは、リージョン別の AWS アカウント ごとに適用されます。リージョンでのアカウントのクォータの引き上げをリクエストするには、Service Quotas コンソールを使用してください。

AWS Management Consoleにサインインし、https://console.aws.amazon.com/servicequotas/ の [Service Quotas] コンソールを開きます。

リソース	デフォルトのクォータ
このアカウントで現在のリージョンに作成できるアクティブな台帳の最大数	5
台帳あたりの Simple Storage Service (Amazon S3) へのアクティブなジャーナルエクスポートの最大数	2
台帳あたりの Kinesis Data Streams へのアクティブなジャーナルストリームの最大数	5

固定クォータ

デフォルトクォータに加えて、QLDB には台帳ごとに次の固定クォータがあります。Service Quotas を使用してこれらのクォータを引き上げることはできません。

リソース	固定クォータ
同時アクティブセッションの数	1500
アクティブなテーブルの数	20
テーブル (アクティブおよび非アクティブ) の合計数 **Note** QLDB では、削除されたテーブルは非アクティブと見なされ、この合計クォータに対してカウントされます。	40
テーブルあたりのインデックス数	5
トランザクション内のドキュメントの数	40
トランザクションで秘匿化するリビジョンの数	1
ドキュメントサイズ (`IonBinary` 形式でエンコード)	128 KB
ステートメントのパラメータサイズ (`IonBinary` 形式)	128 KB
ステートメントのパラメータサイズ (`IonText` 形式)	1 MB
ステートメントの文字列長	100,000 文字
トランザクションサイズ	4 MB

リソース	固定クォータ
トランザクションタイムアウト	30 秒
完了したジャーナルエクスポートジョブの有効期限	7 日間
ターミナルジャーナルストリームの有効期限	7 日間

台帳のクォータ

Service Quotas コンソールを使用して、リージョンでのアカウントの台帳クォータの引き上げをリクエストできます。

https://console.aws.amazon.com/servicequotas/ で Service Quotas コンソールを開きます。

一部の QLDB ユースケースでは、ビジネスの成長に応じて AWS アカウント ごと、リージョンごとに台帳の数を増やす必要があります。例えば、顧客やデータを分離するために専用の台帳を作成する必要があるかもしれません。この場合、マルチアカウントアーキテクチャを活用して QLDB クォータと連携することを検討してください。詳細については、AWS ホワイトペーパー「SaaS のテナント分離戦略」の「アカウントのサイロ分離」を参照してください。

ドキュメントサイズ

`IonBinary` 形式でエンコードされたドキュメントの最大サイズは 128 KB です。`IonText` ではドキュメントのサイズに正確な制限を設定することはできません。テキストからバイナリへの変換は、各ドキュメントの構造によって大きく異なるためです。QLDB はオープンコンテンツを持つドキュメントをサポートしているため、一意のドキュメント構造ごとにサイズの計算が変化します。

トランザクションサイズ

QLDB におけるトランザクションの最大サイズは 4 MB です。トランザクションサイズは、次の要素の合計をもとに計算されます。

デルタ

トランザクション内のすべての命令文によって生成されるドキュメント変更。複数のドキュメントに影響を与えるトランザクションの場合には、影響を受ける個々のドキュメントのデルタの合計が合計のデルタサイズとなります。

メタデータ

影響を受ける個々のドキュメントに関連付けられる、システム生成のトランザクションメタデータ。

インデックス

トランザクションの影響を受けるテーブルでインデックスが定義されている場合、関連するインデックスエントリもデルタを生成します。

履歴

ドキュメントに対するすべてのリビジョンは QLDB 内で永続的に維持されるため、すべてのトランザクションも履歴に追加されます。

挿入 - テーブルに挿入されるすべてのドキュメントには、その履歴テーブルにもコピーが挿入されます。たとえば、100 KB のドキュメントが新たに挿入されるトランザクションにおいては、最低でも 200 KB のデルタが生成されます (これは、メタデータまたはインデックスを含まない概算値です)。

更新 - ドキュメント更新時には、たとえ単一フィールドのみの更新であっても、新たなドキュメント全体のリビジョンが履歴内に生成され、更新のデルタが加減されます。このため、大きなサイズのドキュメントに対し小さな更新を行う場合であっても、大きなトランザクションデルタが生成されます。たとえば、2 KB のデータを既存の 100 KB のドキュメントに追加する場合、新しい 102 KB のリビジョンが履歴内に生成されます。このため、トランザクションの合計デルタは少なくとも 104 KB に達します (繰り返しになりますが、これは、メタデータまたはインデックスを含まない概算値です)。

削除 - 更新と同様に、削除トランザクションでも、新しいドキュメントリビジョンが履歴内に生成されます。ただし新たに作成される DELETE リビジョンは、ユーザーデータが null で、メタデータのみ含むため、元のドキュメントよりサイズが小さくなります。

命名に関する制約

以下の表に、Amazon QLDB での命名に関する制約を示します。

台帳名 ジャーナルストリーム名	• 1～32 個の英数字またはハイフンのみを使用する必要があります。 • 最初と最後の文字は文字または数字であることが必要です。 • すべてを数字にないでください。 • 連続する 2 つのハイフンを含めることはできません。 • 大文字と小文字を区別します。
テーブル名	• 1～128 個の英数字またはアンダースコア文字のみを使用する必要があります。 • 1 字目は文字またはアンダースコアでなければなりません。 • 残りの文字は英数字とアンダースコアの任意の組み合わせにできます。 • 大文字と小文字を区別します。 • QLDB PartiQL の予約語は使用できません。

Amazon QLDB 関連情報

このサービスを利用する際に役立つ関連リソースは次のとおりです。

トピック

- 技術ドキュメント
- GitHub リポジトリ
- AWS ブログ投稿と記事
- メディア
- 一般的な AWS リソース

技術ドキュメント

- Amazon Quantum Ledger Database (QLDB) のよくある質問 – 製品に関してよく寄せられる質問。
- Amazon Quantum Ledger Database (QLDB) 料金 — AWS の料金情報および例。
- AWS re:Post – 質問と回答 (Q&A) のための AWS コミュニティフォーラム。
- Amazon Ion - Amazon Ion データ形式のデベロッパーガイド、ユーザーガイド、およびリファレンス。
- PartiQL - PartiQL クエリ言語の仕様ドキュメントおよび一般的なチュートリアル。
- QLDB Workshops - Amazon QLDB を使用してレコードシステムアプリケーションを構築する実用的で実践的な例を提供するワークショップ。次のラボが含まれます。
  - QLDB の基礎を学ぶ
  - Amazon Ion を使用し、Ion と JSON (Java) との間で相互に変換する
  - AWS Glue と Amazon Athena を使用して、データレイクで QLDB データを有効にする
  - QLDB データを Amazon Aurora MySQL DB インスタンスにストリーミングする
- Amazon QLDB を使用した改ざん防止品質データ - QLDB を使用してデータの変更の正確な履歴を維持することにより、攻撃者による品質データの改ざんを防止する方法を示す AWS ソリューション実装。AWSソリューション実装により、AWS を使用して一般的な問題を解決し、構築を効率化できます。

GitHub リポジトリ

ドライバ

- .NET ドライバー - QLDB ドライバーの.NET 実装。
- Go ドライバー - QLDB ドライバーの Go 実装。
- Java ドライバー - QLDB ドライバーの Java 実装。
- Node.js ドライバー - QLDB ドライバーの Node.js 実装。
- Python ドライバー - QLDB ドライバーの Python 実装。

コマンドラインシェル

- QLDB シェル - QLDB トランザクションデータ API 用のコマンドラインインターフェイスの Python および Rust 実装。

サンプルアプリケーション

- Java DMV アプリケーション - 自動車部門 (DMV) のユースケースに基づくチュートリアルアプリケーション。QLDB および Java 用 QLDB ドライバーを使用するための基本的なオペレーションとベストプラクティスを示します。
- .NET DMV アプリケーション - QLDB および .NET 用 QLDB ドライバーを使用するための基本的なオペレーションとベストプラクティスを示す DMV ベースのチュートリアルアプリケーション。
- Node.js DMV アプリケーション - QLDB および Node.js 用 QLDB ドライバーを使用するための基本的なオペレーションとベストプラクティスを示す DMV ベースのチュートリアルアプリケーション。
- Python DMV アプリケーション - QLDB および Python 用 QLDB ドライバーを使用するための基本的なオペレーションとベストプラクティスを示す DMV ベースのチュートリアルアプリケーション。
- Ledger loader – サポートされている配信チャネル (AWS DMS、Amazon SQS、Amazon SNS、Kinesis データストリーム、Amazon MSK、またはEventBridge) を使用して QLDB 台帳に高速でデータを非同期ロードするための Java フレームワーク。
- Export processor – Amazon S3 で QLDB エクスポートを処理する作業を処理する拡張可能な Java フレームワーク。エクスポートの出力を読み取り、エクスポートされたブロックを順番に反復処理します。

- QLDB streams sample Lambda in Python - Amazon SQS キューをサブスクライブしている Amazon SNS トピックに QLDB データを送信する AWS Lambda 関数を使用して、QLDB ストリームを消費する方法を示すアプリケーション。
- QLDB streams OpenSearch integration sample – ストリームを使用して Amazon OpenSearch Service と QLDB を統合する方法を示す Python アプリケーション。
- Double-entry application - QLDB を使用して複式簿記財務元帳アプリケーションをモデル化する方法を示す Java アプリケーション。
- Node.js 用の QLDB KVS - ドキュメント検証のための追加の関数を備えた QLDB 用のシンプルな key-value ストアインターフェイスライブラリ。

Amazon Ion と PartiQL

- Amazon Ion ライブラリ - Ion チームがサポートするライブラリ、ツール、ドキュメント。
- PartiQL 実装 - PartiQL の実装、仕様、およびチュートリアル。

AWS ブログ投稿と記事

- How Earnin built their ledger service using Amazon QLDB (2023 年 2 月 16 日) – Earnin.com が QLDB を使用して台帳サービスを構築し、モダンで完全な機能を準備したモバイル金融アプリケーションをユーザーに提供した方法について説明しています。
- Export and analyze Amazon QLDB journal data using AWS Glue and Amazon Athena (2022 年 12 月 19 日) – QLDB AWS Glue と Athena のエクスポート機能を使用して、台帳ベースのアーキテクチャにレポートや分析機能を提供する方法について説明します。
- How Shinsegae International enhances customer experience and prevents counterfeiting with Amazon QLDB (2022 年 8 月 3 日) – 新世界インターナショナルが Amazon QLDB を使用して顧客に高級品の真正性を知らせ、製品の購入および流通履歴を提供するデジタル認証サービスを構築した方法について説明しています。
- BungkusIT uses Amazon QLDB and VeriDoc Global's ISV technology to improve the customer and delivery agent experience (2022 年 4 月 29 日) – ある物流会社の BungKuit が QLDB を使用して、不変で暗号的に検証可能なトランザクションログを備えた業界間通信のための一元化された透過的なプラットフォームを実装した方法を示しています。
- Use Amazon QLDB as an immutable key-value store with a REST API and JSON (2022 年 2 月 14 日) – QLDB を不変のキーバリューストアとして扱い、REST API を通じてその監査機能を使用する方法を紹介します。

- Building a core banking system with Amazon QLDB (2022 年 1 月 21 日) – QLDB を使用してコアバンキング台帳システムを構築する方法を説明します。また、QLDB が金融サービス業界のニーズに適した、目的に合ったデータベースである理由も示しています。
- How Specright uses Amazon QLDB to create a traceable supply chain network (2022 年 1 月 21 日) – Specright が QLDB を使用して、卸売ブランド、小売業者、製造業者が重要なサプライチェーンデータやパッケージ仕様データを共有できるようにする追跡可能なサプライチェーンネットワークを構築した方法を示しています。
- How fEMR Delivers Cryptographically Secure and Verifiable Medical Data with Amazon QLDB (2021 年 12 月 23 日) – Team fEMR が QLDB やその他のAWSマネージドサービスを活用して救援活動を実現した方法を探ります。
- Monitor Amazon QLDB query access patterns (2021 年 11 月 8 日) – QLDB トランザクションと、トランザクションで実行された PartiQL ステートメントを、Amazon API Gateway を介して受信した API リクエストに関連付ける方法を示します。
- Build a simple CRUD operation and data stream on Amazon QLDB using AWS Lambda (2021 年 9 月 28 日) - AWS Lambda 関数を使用して QLDB で CRUD (作成、読み取り、更新、削除) オペレーションを実行する方法を示しています。
- Real-world cryptographic verification with Amazon QLDB (2021 年 8 月 26 日) – 現実的なユースケースに基づいて、QLDB 台帳における暗号検証の価値について解説しています。
- Verify delivery conditions with the Accord Project and Amazon QLDB: Part 1、Part 2 (2021 年 6 月 28 日) – オープンソースの Accord Project と QLDB を使用して納入条件を検証するスマートな適法契約テクノロジーの応用方法について解説しています。
- Amazon QLDB data streaming via AWS CDK (2021 年 6 月 7 日) – AWS Cloud Development Kit (AWS CDK) を使用した QLDB の設定、AWS Lambda 関数を使用した QLDB データの入力、および台帳データの耐障害性を提供する QLDB ストリーミングの設定を行う方法を示しています。
- Demo fine grained access control in QLDB (2021 年 6 月 1 日) – QLDB 台帳用の詳細な AWS Identity and Access Management (IAM) アクセス許可を始める方法について示しています。
- Stream Processing with DynamoDB Streams and QLDB Streams (2020 年 11 月 23 日) – DynamoDB Streams と QLDB ストリーミングを簡潔に比較し、これらの使用を開始するヒントを提供しています。
- Streaming data from Amazon QLDB to OpenSearch (2020 年 8 月 19 日) – QLDB から Amazon OpenSearch Service にデータをストリーミングして、リッチテキスト検索や、レコード間の集計やメトリクスなどのダウンストリーム分析をサポートする方法について説明しています。

- How I streamed data from Amazon QLDB to DynamoDB using Nodejs in near-real time (2020 年 7 月 7 日) – QLDB から DynamoDB へデータをストリーミングして、1 桁の遅延と無制限にスケーラブルなキーと値の照合をサポートする方法について説明しています。
- Building a GraphQL interface to Amazon QLDB with AWS AppSync: Part 1、Part 2 (2020 年 5 月 4 日) – QLDB と AWS AppSync を統合し、汎用的で GraphQL を活用した API を QLDB 台帳で提供する方法を解説しています。

メディア

AWS 動画

- AWS Tutorials & Demos: QLDB to Aurora Streaming (2023 年 3 月 17 日、21 分) – このビデオでは、QLDB ストリーミング機能を実装するための基本概念を説明し、QLDB から Amazon Aurora MySQL ダウンストリーム DB へのデータストリーミングを設定する方法を示します。
- ArcBlock: Leveraging Amazon QLDB to Build a Decentralized Identity Solution (2022 年 5 月 31 日、4 分) – ArcBlock は、QLDB を使用して構築した分散型 ID ソリューションについて順を追って説明します。
- AWS re:Invent 2020: Using Amazon QLDB as a system-of-trust database for core business apps (2021 年 2 月 5 日、30 分) – 初期の Amazon QLDB ユーザーが台帳データベースのデータ来歴と暗号検証可能性に関する独自のプロパティをどのように適用してデータの整合性が組み込まれたレコードシステムを実装したかについて説明します。
- AWS re:Invent 2020: Building out a serverless application with Amazon QLDB (2021 年 2 月 5 日、28 分) – Amazon QLDB と AWS Lambda、Amazon Kinesis、Amazon DynamoDB などのサービスを組み合わせて、完全に機能するサーバーレスアプリケーションを構築、テスト、最適化する方法について説明します。
- AWS re:Invent 2020: Building audit-based apps that maintain data integrity with Amazon QLDB (2021 年 2 月 5 日、18 分) – このセッションでは、Amazon QLDB で解決できる問題を解説し、台帳データベースを使用する時期と理由に関する質問に回答し、Osano などの顧客のユースケースを共有します。
- How to Centrally Store Immutable Logs using Amazon QLDB with .NET Applications (2020 年 12 月 7 日、10 分) – .NET アプリケーションで QLDB を使用して、イミュータブルログを一元的に保存する方法のデモ。
- Virtual Workshop: Building Ledger-Based Systems of Record with QLDB and Java - AWS Online Tech Talks (2020 年 7 月 29 日、87 分) – Amazon Ion ライブラリと Java 用 QLDB ドライバーを

使用して、データローダープログラムを構築する Working With Ion (Ion の使用) Immersion Day ラボについて解説するインストラクター主導のワークショップ。

- Developer Workshop: Using AWS's Immutable Ledger Database QLDB on ABT Node (2020 年 6 月 22 日、34 分) - ABT Node 用 QLDB のセットアップと構成に関するステップバイステップガイド。また、ArcBlock のブロックチェーンエクスプローラーとボーディングゲートブロックレットをインストールして QLDB に接続する方法についても説明しています。
- AWS Tech Talk: A Customer's Perspective on Building an Event-Triggered System-of-Record Application with Amazon QLDB (2020 年 3 月 31 日、29 分) – AWS Data Hero で、英国の Driver and Vehicle Licensing Agency (DVLA) のチーフアーキテクトである Matt Lewis 氏によるテクノロジーに関する講演。QLDB と DVLA のアプリケーションのイベント駆動型アーキテクチャに関するユースケースについて説明します。
- AWS re:Invent 2019: Why you need a ledger database: BMW, DVLA, & Sage discuss use cases (2019 年 12 月 5 日、47 分) – 顧客である BMW、DVLA (英国の政府機関) および Sage によるプレゼンテーション。台帳データベースを使用する理由についてディスカッションし、QLDB のユースケースを共有します。
- AWS re:Invent 2019: Amazon QLDB: An engineer's deep dive on why this is a game changer (2019 年 12 月 5 日、50 分) – Andrew Certain (AWS Distinguished Engineer) によるプレゼンテーション。QLDB のユニークなジャーナル指向アーキテクチャとさまざまなイノベーションについて解説します。暗号化ハッシュ、マークルツリー、複数のアベイラビリティーゾーンレプリケーション、PartiQL サポートが含まれます。
- Building Applications with Amazon QLDB, a First-of-Its-Kind Ledger Database - AWS Online Tech Talks (2019 年 11 月 19 日、51 分) – QLDB のユニークな特徴および主要な機能の使用方法の技術的な詳細について説明します。これには、データの全履歴のクエリ、ドキュメントの暗号による検証、データモデルの設計が含まれます。

ポッドキャスト

- Why are customers choosing Amazon QLDB? (お客様が Amazon QLDB を選ぶ理由) (2020 年 7 月 5 日、33 分) – 台帳データベースの定義、ブロックチェーンとの違い、お客様が現在どのように使用しているかについてのディスカッションです。

一般的な AWS リソース

- クラスとワークショップ – AWS のスキルを磨き、実践的な経験が得るために役立つセルフペースラボに加えて、ロールベースのコースと特別コースへのリンクです。

- AWS デベロッパーセンター – チュートリアルの検索、ツールのダウンロード、AWS デベロッパーイベントの確認を行います。
- AWS デベロッパーツール - AWS アプリケーションを開発および管理するためのデベロッパーツール、SDK、IDE ツールキット、およびコマンドラインツールへのリンクです。
- ご利用開始のためのリソースセンター — AWS アカウント をセットアップする方法、AWS コミュニティに参加する方法、最初のアプリケーションを起動する方法を説明します。
- ハンズオンチュートリアル - ステップ バイ ステップのチュートリアルに従って、最初のアプリケーションを AWS で起動します。
- AWS ホワイトペーパー – アーキテクチャ、セキュリティ、エコノミクスなどのトピックについて、AWS のソリューションアーキテクトや他の技術エキスパートが記述した AWS の技術ホワイトペーパーの包括的なリストへのリンクです。
- AWS Support Center – AWS Support のケースを作成して管理するためのハブです。フォーラム、技術上のよくある質問、サービスヘルスステータス、AWS Trusted Advisor など、他の役立つリソースへのリンクも含まれています。
- AWS Support – AWS Support に関する情報のメインウェブページです。クラウド内でのアプリケーションの構築および実行を支援するために 1 対 1 での迅速な対応を行うサポートチャネルとして機能します。
- お問い合わせ - AWS の請求、アカウント、イベント、不正使用、その他の問題などに関するお問い合わせの受付窓口です。
- AWS サイトの利用規約 – 当社の著作権、商標、お客様のアカウント、ライセンス、サイトへのアクセス、その他のトピックに関する詳細情報。

Amazon QLDB のリリース履歴

以下の表では、Amazon QLDB の各リリースの重要な変更点と、それに対応する Amazon QLDB デベロッパーガイドの更新について説明します。このドキュメントの更新に関する通知については、RSS フィードにサブスクライブできます。

- API バージョン: 2019-01-02
- ドキュメントの最終更新: 2023 年 1 月 3 日

変更	説明	日付
IAM ガイダンスを更新	IAM のベストプラクティスに合わせてガイドを更新しました。詳細については、「Security best practices in IAM」(IAM のセキュリティのベストプラクティス) を参照してください。	2023 年 1 月 3 日
AWS マネージドポリシーの更新	Amazon QLDB では、既存の AWS マネージドポリシーである `AmazonQLDBFullAccess` および `AmazonQLDBConsoleFullAccess` が更新されました。これらのポリシーには、プリンシパルが PartiQL ストアドプロシージャを使用してドキュメントのリビジョンを秘匿化できるようにする新しい権限が追加されました。詳細については、「AWS managed policies for Amazon QLDB」を参照してください。	2022 年 11 月 4 日

データの秘匿化	Amazon QLDB は、2021 年 7 月 22 日以降に作成されたレジャーの `REDACT_REVISION` PartiQL ストアドプロシージャをサポートするようになりました。このストアドプロシージャを使用すると、履歴に残っている使用頻度の低いドキュメントリビジョンを完全に削除しても、台帳の全体的なデータ整合性を維持できます。詳細については、「Redacting document revisions」を参照してください。	2022 年 11 月 3 日
Node.js ドライバー v3	Node.js バージョン 3.0 用 Amazon QLDB ドライバーが公開されました。このバージョンでは AWS SDK for JavaScript v3 のサポートが導入されています。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-nodejs を参照してください。	2022 年 9 月 26 日
Go ドライバー v3	Go バージョン 3.0 用 Amazon QLDB ドライバーが公開されました。このバージョンでは AWS SDK for Go v2 のサポートが導入されています。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-go を参照してください。	2022 年 8 月 11 日

新しい PartiQL クエリエディタ	Amazon QLDB コンソールの新しい PartiQL クエリエディタが、一般利用可能になりました。新しい QLDB PartiQL エディタは、クエリの作成、トランザクションのデバッグ、結果の調査を行うための改良されたインターフェイスを提供します。エディタを開いて使用する方法については、「Accessing Amazon QLDB using the console」を参照してください。	2022 年 6 月 22 日
.NET ドライバー用の Ion オブジェクトマッパー	.NET バージョン 1.3 用の Amazon QLDB ドライバーでは、Ion オブジェクトマッパーのサポートが導入されています。この機能により、Amazon Ion タイプとネイティブ C# タイプを手動で変換する必要がなくなります。Ion オブジェクトマッパーのすべての変更履歴については、GitHub リポジトリ `amzn/ion-object-mapper-dotnet` の CHANGELOG.md ファイルを参照してください。	2022 年 1 月 19 日
JSON ジャーナルエクスポートフォーマット	Amazon QLDB は、ジャーナルエクスポート用の JSON ライン出力形式をサポートするようになりました。詳細については、「Exporting journal data from Amazon QLDB」を参照してください。	2021 年 12 月 21 日

Amazon QLDB のトラブルシューティング	Amazon QLDB の使用時に発生する可能性のある一般的なエラーを集約したリストのガイダンスを提供する新しいトラブルシューティングトピックを追加しました。	2021 年 12 月 8 日
新しいリージョンへの対応	Amazon QLDB がカナダ (中部) リージョンで使用可能になりました。利用可能なリージョンの完全なリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。	2021 年 11 月 11 日
サービス間の混乱した代理の防止	Amazon QLDB は混乱した代理の問題を防止するため、IAM リソースポリシーで `aws:SourceArn` および `aws:SourceAccount` のグローバル条件コンテキストキーの使用をサポートするようになりました。詳細については、クロスサービスでの混乱した代理処理を防止するを参照してください。	2021 年 11 月 8 日
Amazon QLDB シェル v2	Rust で書かれた、Amazon QLDB シェルのバージョン 2.0 が一般公開されました。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-shell を参照してください。	2021 年 10 月 14 日

AWS マネージドポリシーの更新	Amazon QLDB では、既存の AWS マネージドポリシーである `AmazonQLDBFullAccess` および `AmazonQLDBConsoleFullAccess` が更新されました。これらのポリシーには、プリンシパルがアカウント内の任意の IAM ロールリソースを QLDB サービスに渡せるようにするアクセス許可が新たに追加されました。これは、ジャーナルのどのエクスポートとストリーミングリクエストにも必要です。詳細については、「AWS managed policies for Amazon QLDB」を参照してください。	2021 年 9 月 2 日
カスタマーマネージド AWS KMS キー	Amazon QLDB では、新しい台帳リソース向けに、AWS Key Management Service (AWS KMS) のカスタマーマネージドキーを使用した保管時の暗号化がサポートされるようになりました。詳細については、「Amazon QLDB での保管時の暗号化」を参照してください。	2021 年 7 月 22 日

AWS マネージドポリシーの更新	Amazon QLDB では、以前 2 回リストされ、重複していた AWS アクションを削除するために、既存の `AmazonQLDBReadOnly` マネージドポリシーである `qldb:GetBlock` が更新されました。詳細については、「AWS managed policies for Amazon QLDB」を参照してください。	2021 年 7 月 1 日
新しいリージョンへの対応	Amazon QLDB が欧州 (ロンドン) リージョンで利用可能になりました。利用可能なリージョンの完全なリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。	2021 年 6 月 24 日

AWS マネージドポリシーの更新	Amazon QLDB では、既存の AWS マネージドポリシーである `AmazonQLDBFullAccess` および `AmazonQLDBConsoleFullAccess` が更新されました。これらのポリシーに新しく追加されたアクセス許可により、プリンシパルがすべての台帳のアクセス許可モードを更新するとともに、すべての STANDARD アクセス許可の台帳ですべての PartiQL コマンドを実行できるようになりました。詳細については、「AWS managed policies for Amazon QLDB」を参照してください。	2021 年 5 月 27 日
標準アクセス許可モード	Amazon QLDB では、台帳リソースの STANDARD アクセス許可モードがサポートされるようになりました。標準アクセス許可モードでは、台帳、テーブル、PartiQL コマンドのアクセスを非常に細かく制御できます。詳細については、「標準アクセス許可モードの開始方法」を参照してください。	2021 年 5 月 27 日

PartiQL ステートメントの統計	PartiQL ステートメントの統計機能が、Amazon QLDB コンソールのクエリエディタで利用可能になりました。詳細については、「Getting statement statistics」(ステートメント統計の取得) を参照してください。	2021 年 5 月 24 日
チュートリアル: AWS SDK を使用したデータ検証	コード例付きのステップバイステップのチュートリアルが追加されました。このチュートリアルでは、AWS SDK による QLDB API を使用して、Amazon QLDB のリビジョンハッシュとブロックハッシュを検証する方法を説明します。詳細については、「チュートリアル: AWS SDK を使用したデータ検証」を参照してください。	2021 年 5 月 6 日
.NET ドライバー v1.2	.NET バージョン 1.2 用 Amazon QLDB ドライバーが公開されました。このバージョンには、非同期 API が導入されています。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-dotnet を参照してください。	2021 年 4 月 1 日

PartiQL DROP INDEX ステートメント	Amazon QLDB の PartiQL で、DROP INDEX ステートメントがサポートされるようになりました。詳細については、「インデックスの削除」を参照してください。	2021 年 3 月 3 日
PartiQL ステートメントの統計	PartiQL ステートメントの統計機能が、すべてのサポート対象言語 (.NET、Go、Python など) の Amazon QLDB ドライバーの最新バージョンで利用可能になりました。詳細については、「Getting statement statistics」(ステートメント統計の取得) を参照してください。	2021 年 2 月 25 日
TypeScript クイックスタートチュートリアル	Node.js 用 Amazon QLDB ドライバー向け「クイックスタートチュートリアル」に TypeScript のコード例が追加されました。	2020 年 12 月 28 日
PartiQL ステートメントの統計	Java および Node.js 用 Amazon QLDB ドライバーの最新バージョンでは、ステートメント実行の統計を利用できるようになりました。これにより、PartiQL ステートメントを効率よく実行できます。詳細については、「Getting statement statistics」(ステートメント統計の取得) を参照してください。	2020 年 12 月 22 日

ドライバーのクックブックリファレンスとクイックスタートチュートリアル	Go と Node.js 用ドライバーのクックブックリファレンス、Java と Python 用ドライバーのクイックスタートチュートリアル、QLDB で Amazon Ion を操作するためのガイドが追加されました。詳細については、「Amazon QLDB ドライバーの使用方法」を参照してください。	2020 年 11 月 24 日
Amazon QLDB シェル v1.1	Amazon QLDB シェルのバージョン 1.1 が公開されました。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-shell を参照してください。	2020 年 10 月 26 日
Go 用 Amazon QLDB ドライバー	Go 用 Amazon QLDB ドライバーが公開されました。このドライバーは GitHub でオープンソースとして公開されており、AWS SDK for Go を使用して QLDB のトランザクションデータ API とやり取りできます。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-go を参照してください。	2020 年 10 月 20 日

Node.js ドライバーセットアップの推奨事項	Node.js 用 Amazon QLDB ドライバーのセットアップの推奨事項の新しいセクションが追加されました。これには、Keep-Alive によって再接続することでレイテンシーを減らす方法などが含まれます。	2020 年 10 月 16 日
空でないテーブルでのインデックス作成	Amazon QLDB では、空でないテーブルでのインデックス作成がサポートされるようになりました。詳細については、「インデックス管理」を参照してください。	2020 年 9 月 30 日
クエリパフォーマンスの最適化	Amazon QLDB のクエリ上の制約について説明し、こうした制約下でのクエリパフォーマンスの最適化のガイダンスとなる新しいセクションが追加されました。	2020 年 9 月 18 日
Java ドライバーのクックブッククリファレンス	Java 用 Amazon QLDB ドライバーの一般的なユースケースのコード例を示すクックブッククリファレンスが追加されました。	2020 年 9 月 3 日
ドライバーによるセッション管理	Amazon QLDB のドライバーによるセッション管理の概要を示し、データトランザクションの実行時に QLDB ドライバーがセッションを処理する方法について説明する新しいセクションが追加されました。	2020 年 9 月 1 日

Java ドライバー v2.0	Java バージョン 2.0 用 Amazon QLDB ドライバーが公開されました。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-java を参照してください。	2020 年 8 月 28 日
Node.js ドライバー v2.0	Node.js バージョン 2.0 用 Amazon QLDB ドライバーが公開されました。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-nodejs を参照してください。	2020 年 8 月 27 日
関連情報	関連情報のリンクのほか、Amazon QLDB の理解と操作に役立つその他のリソースのリンクがある新しいトピックが追加されました。	2020 年 8 月 24 日
Python ドライバー v3.0	Python バージョン 3.0 用 Amazon QLDB ドライバーが公開されました。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-python を参照してください。	2020 年 8 月 20 日

Go 用 Amazon QLDB ドライバー	Go 用 Amazon QLDB ドライバーのプレビューリリースが公開されました。このドライバーにより、AWS SDK for Go を使用して QLDB のトランザクションデータ API とやり取りできます。詳細については、「Go 用 Amazon QLDB ドライバー (プレビュー)」を参照してください。	2020 年 8 月 6 日
Python ドライバーのクックブックリファレンス	Python 用 Amazon QLDB ドライバーの一般的なユースケースのコード例を示すクックブックリファレンスが追加されました。	2020 年 7 月 24 日
Amazon QLDB で割り当てられる一意の ID	Amazon QLDB の一意の ID のプロパティと使用ガイドラインについて説明する新しいセクションが追加されました。	2020 年 7 月 9 日
ジャーナルストリームの AWS CloudFormation リソース	Amazon QLDB で、AWS CloudFormation テンプレートを使用してジャーナルストリームを作成するためのリソースがサポートされるようになりました。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS::QLDB::Stream」リソースを参照してください。	2020 年 7 月 9 日

.NET ドライバーのクックブッククリファレンス	.NET 用 Amazon QLDB ドライバーの一般的なユースケースのコード例を示すクックブックリファレンスが追加されました。	2020 年 7 月 1 日
.NET 用 Amazon QLDB ドライバー	.NET 用 Amazon QLDB ドライバーが公開されました。このドライバーは GitHub でオープンソースとして公開されており、AWS SDK for .NET を使用して QLDB のトランザクションデータ API とやり取りできます。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-dotnet を参照してください。	2020 年 6 月 26 日
Node.js 用 Amazon QLDB ドライバー	Node.js 用 Amazon QLDB ドライバーが公開されました。このドライバーは GitHub でオープンソースとして公開されており、AWS SDK for JavaScript in Node.js を使用して QLDB のトランザクションデータ API とやり取りできます。リリースノートについては、GitHub リポジトリ awslabs/amazon-qldb-driver-nodejs を参照してください。	2020 年 6 月 5 日

Amazon QLDB ジャーナルストリーム	Amazon QLDB では、ジャーナルにコミットしたドキュメントリビジョンをすべてキャプチャするストリーミングを作成し、そのデータを Amazon Kinesis Data Streams にほぼリアルタイムで配信できるようになりました。詳細については、「Amazon QLDB からのジャーナルデータのストリーミング」を参照してください。	2020 年 5 月 19 日
Amazon Ion コード例	Java、Node.js、および Python 用 QLDB ドライバーを使用して、Amazon QLDB 台帳内の Amazon Ion データをクエリおよび処理するコード例が追加されました。詳細については、「Amazon QLDB の Ion コード例」を参照してください。	2020 年 5 月 12 日
同時実行モデルとジャーナルコンテンツ	Amazon QLDB 同時実行モデルのトピックに、インデックスを使用してオプティミスティック同時実行制御 (OCC) の競合を制限する方法についての情報が追加されました。Amazon QLDB のジャーナルコンテンツについて説明する新しいセクションが追加されました。	2020 年 5 月 4 日
Node.js ドライバーのクイックスタートガイド	Node.js 用 Amazon QLDB ドライバー向けクイックスタートガイドが追加されました。	2020 年 5 月 1 日

Amazon QLDB シェル	Amazon QLDB シェルが公開されました。このシェルは GitHub 上でオープンソースとして公開されており、台帳データに対して PartiQL ステートメントを実行できるコマンドラインインターフェイスを提供します。リリースノートについては、awslabs/amazon-qldb-shell GitHub リポジトリを参照してください。	2020 年 4 月 20 日
コンプライアンス証明書	Amazon QLDB は、HIPAA や ISO などの AWS コンプライアンスプログラムの認定を受けています。詳細については、「Amazon QLDB のコンプライアンス検証」を参照してください。	2020 年 4 月 3 日
ドライバーに関する推奨事項の拡大	Amazon QLDB ドライバーに関する推奨事項のセクションが、すべてのサポート対象プログラミング言語に適用できるように拡大されました。	2020 年 4 月 2 日
Amazon QLDB シェル	Amazon QLDB シェルのプレビューリリースが利用可能になりました。このシェルは、台帳データに対して PartiQL ステートメントを実行できるコマンドラインインターフェイスを提供します。詳細については、「Accessing Amazon QLDB using the QLDB shell (data API only) (preview)」を参照してください。	2020年3月23日

Java ドライバー v1.1	Java バージョン 1.1 用 Amazon QLDB ドライバーが公開されました。リリースノートについては、awslabs/amazon-qldb-driver-java GitHub リポジトリを参照してください。	2020 年 3 月 20 日
.NET 用 Amazon QLDB ドライバー	Amazon QLDB で、.NET ドライバーのプレビューリリースが提供されるようになりました。このドライバーにより、AWS SDK for .NET を使用して QLDB のトランザクションデータ API とやり取りできます。詳細については、「.NET 用 Amazon QLDB ドライバー (プレビュー)」を参照してください。	2020 年 3 月 13 日
Python 用 Amazon QLDB ドライバー	Python 用 Amazon QLDB ドライバーが公開されました。このドライバーは GitHub でオープンソースとして公開されており、AWS SDK for Python (Boto3) を使用して QLDB のトランザクションデータ API とやり取りできます。リリースノートについては、awslabs/amazon-qldb-driver-python GitHub リポジトリを参照してください。	2020 年 3 月 11 日

PartiQL の UNDROP TABLE ステートメントとネストされている DML	Amazon QLDB の PartiQL で、ネストされたコレクションを FROM 句のソースとして指定するデータ操作言語 (DML) ステートメントがサポートされるようになりました。詳細については、「PartiQL リファレンス」の FROM ステートメントを参照してください。QLDB の PartiQL では、UNDROP TABLE ステートメントもサポートされています。詳細については、「テーブルの削除の取り消し」を参照してください。	2020 年 2 月 26 日
紹介トピックの拡大	はじめの「Amazon QLDB とは」のトピックを拡大して、新しいセクション「Amazon QLDB の概要」と「リレーショナルから台帳へ」を追加しました。	2020 年 1 月 24 日
サポートされている関数の PartiQL リファレンス	「Amazon QLDB PartiQL リファレンス」を拡大し、サポート対象の SQL 関数に関する詳細な使用方法の情報を含めました。詳細については、「PartiQL 関数」を参照してください。	2020 年 1 月 2 日
ドライバーの推奨事項と一般的なエラー	Java 用 Amazon QLDB ドライバーの推奨事項と、すべてのドライバー言語の一般的なエラーについて説明する新しいセクションが追加されました。	2020 年 1 月 2 日

新しいリージョンへの対応	Amazon QLDB が、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、欧州 (フランクフルト) の各リージョンで利用可能になりました。利用可能なリージョンの完全なリストについては、「Amazon Web Services 全般のリファレンス」の「Amazon QLDB エンドポイントとクォータ」を参照してください。	2019 年 11 月 19 日
Node.js 用 Amazon QLDB ドライバー	Amazon QLDB で、Node.js ドライバーのプレビューリリースが提供されるようになりました。このドライバーにより、AWS SDK for JavaScript in Node.js を使用して QLDB のトランザクションデータ API とやり取りできます。詳細については、「Node.js 用 Amazon QLDB ドライバー (プレビュー)」を参照してください。	2019 年 11 月 13 日

Python 用 Amazon QLDB ドライバー	Amazon QLDB で、Python ドライバーのプレビューリリースが提供されるようになりました。このドライバーにより、AWS SDK for Python (Boto3) を使用して QLDB のトランザクションデータ API とやり取りできます。詳細については、「Python 用 Amazon QLDB ドライバー (プレビュー)」を参照してください。	2019 年 10 月 29 日
パブリックリリース	これは、Amazon QLDB の最初の一般リリースです。このリリースには、デベロッパーガイドと統合台帳管理 API リファレンスが含まれています。	2019 年 9 月 10 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。