ユーザーガイド

Amazon Athena

Amazon Athena: ユーザーガイド

Table of Contents

Amazon Athena とは

Amazon Athena は、標準的な SQL を使用して Amazon Simple Storage Service (Amazon S3) 内のデータを直接分析することを容易にするインタラクティブなクエリサービスです。AWS Management Console でいくつかのアクションを実行するだけで、Athena にデータの保存先の Amazon S3 を設定し、標準 SQL を使用してアドホッククエリの実行を開始できます。結果は数秒で返されます。

詳細については、「開始」を参照してください。

また、Amazon Athena により、リソースの計画、設定、管理をしなくても、Apache Spark を使用してデータ分析をインタラクティブに実行することも簡単になります。Athena で Apache Spark アプリケーションを実行する場合は、処理用の Spark コードを送信して、結果を直接受け取ります。Amazon Athena コンソールのシンプルなノートブックエクスペリエンスを使用して、Python または Athena ノートブック API を使用して Apache Spark アプリケーションを開発できます。

詳細については、「Amazon Athena で Apache Spark を開始」を参照してください。

Athena SQL および Apache Spark on Amazon Athena はサーバーレスであるため、インフラストラクチャのセットアップや管理は不要です。また、実行したクエリにのみ課金されます。Athena は、自動的にスケールしてクエリを並列実行するため、大規模なデータベースや複合型のクエリでも結果がすぐに返されます。

トピック

Athena の用途

Amazon Athena などのクエリサービス、Amazon Redshift などのデータウェアハウス、Amazon EMR などの高度なデータ処理フレームワークはすべて、それぞれが異なるニーズとユースケースに対応します。以下のガイダンスは、要件に基づいて 1 つ以上のサービスを選択する際に役に立ちます。

Amazon Athena

Athena は、Amazon S3 に保存された非構造化データ、半構造化データ、および構造化データの分析に役立ちます。たとえば、CSV 形式、JSON 形式、列データ形式 (Apache Parquet や Apache ORC など) に対応しています。Athena は ANSI SQL を使用したアドホッククエリの実行に利用でき、データを集約したり、データを Athena にロードしたりする必要はありません。

Athena は Amazon QuickSight と統合して、データを簡単に可視化できるようにします。Athena を使用して、JDBC や ODBC ドライバーで接続されたビジネスインテリジェンスツールや SQL クライアントでレポートを生成、またはデータを探索できます。詳細については、「Amazon QuickSight ユーザーガイド」の「Amazon QuickSight とは」、および「ODBC および JDBC ドライバーを使用した Amazon Athena への接続」を参照してください。

Athena は、Amazon S3 内のデータに永続的なメタデータストアを提供する AWS Glue Data Catalog と統合します。これにより、Amazon Web Services アカウント全体で利用でき、AWS Glue の ETL およびデータ検出機能と統合された中央メタデータストアに基づいて、Athena でのテーブルの作成とデータのクエリを行うことが可能になります。詳細については、AWS Glue デベロッパーガイドの「AWS Glue との統合」および「AWS Glue とは」を参照してください。

Amazon Athena を使用すると、データをフォーマットしたり、インフラストラクチャを管理したりすることなく、Simple Storage Service (Amazon S3) 内のデータに対してインタラクティブなクエリを簡単に実行できます。たとえば、Athena は、Web ログでクイッククエリを素早く実行し、サイトのパフォーマンス上の問題をトラブルシューティングする場合に便利です。Athena では、データのテーブルを定義し、標準 SQL を使用してクエリを開始するだけで、すばやく開始できます。

インフラストラクチャやクラスターを管理することなく、Simple Storage Service (Amazon S3) のデータに対してインタラクティブなアドホック SQL クエリを実行する場合は、Amazon Athena を使用してください。Amazon Athena は、サーバーをセットアップしたり管理したりすることなく、Simple Storage Service (Amazon S3) のデータに対してアドホッククエリを実行できる、最も簡単な方法となります。

Athena で活用または統合できる AWS のサービス のリストについては、「the section called “AWS のサービス における Athena との統合”」を参照してください。

Amazon EMR

Amazon EMR では、オンプレミスのデプロイと比較すると、Hadoop、Spark、Presto などの高度に分散された処理フレームワークをシンプルかつコスト効率よく実行できます。Amazon EMR は柔軟

性があります。カスタムアプリケーションやコードを実行して、特定のコンピューティング、メモリ、ストレージ、およびアプリケーションパラメータを定義して、分析要件を最適化することができます。

SQL クエリを実行することに加えて、Amazon EMR では、機械学習、グラフ分析、データ変換、ストリーミングデータなど、コーディングできるほぼすべての用途に対して、さまざまなスケールアウトデータ処理タスクを実行できます。カスタムコードを使用して Spark、Hadoop、Presto、Hbase などの最新のビッグデータ処理フレームワークを使用した非常に膨大なデータセットを処理および分析する場合は、Amazon EMR を使用する必要があります。Amazon EMR では、クラスターとクラスターにインストールされているソフトウェアの設定を完全に制御できます。

Amazon Athena を使用すれば、Amazon EMR を使用して処理するデータをクエリすることができます。Amazon Athena は、Amazon EMR と同じデータ形式の多くをサポートしています。Athena のデータカタログは Hive メタストアとの互換性があります。EMR を使用しており、すでに Hive メタストアがある場合は、Amazon Athena で DDL ステートメントを実行して、Amazon EMR ジョブに影響を与えることなくすぐにデータをクエリできます。

Amazon Redshift

Amazon Redshift などのデータウェアハウスは、在庫システム、金融システム、小売販売システムなどのさまざまなソースからデータを共通の形式にまとめ、長期間保存する必要がある場合に最適です。履歴データから高度なビジネスレポートを作成する場合は、Amazon Redshift のようなデータウェアハウスが最適です。Amazon Redshift のクエリエンジンは、多数の非常に大きなデータベーステーブルを結合する複合型のクエリの実行時に、特にうまく機能するように最適化されています。非常に大きな多数のテーブルを使って、結合を多数実行する高度に構造化されたデータに対してクエリを実行する必要がある場合は、Amazon Redshift を選択してください。

Athena を使用する状況の詳細については、以下のリソースを参照してください。

- 「ご利用のためのリソースセンター」の「AWS 分析サービスの選択」
- 「Amazon Athena のよくある質問」の「Athena と他のビッグデータサービスを比較する場合」
- Amazon Athena の概要
- Amazon Athena の特徴
- Amazon Athena のよくある質問
- Amazon Athena ブログ記事

AWS のサービス における Athena との統合

Athena を使用して、このセクションにリストされている AWS のサービス からのデータをクエリできます。各サービスでサポートされるリージョンについては、「Amazon Web Services 全般のリファレンス」の「リージョンとエンドポイント」を参照してください。

Athena と統合されている AWS のサービス

- AWS CloudFormation
- Amazon CloudFront
- AWS CloudTrail
- Amazon DataZone
- Elastic Load Balancing
- Amazon EMR Studio
- AWS Glue Data Catalog
- AWS Identity and Access Management (IAM)
- Amazon QuickSight
- Amazon S3 インベントリ
- AWS Step Functions
- AWS Systems Manager インベントリ
- Amazon Virtual Private Cloud

各統合の詳細については、以下のセクションを参照してください。

AWS CloudFormation

キャパシティーの予約

参照トピック:「AWS CloudFormation ユーザーガイド」の「AWS::Athena::CapacityReservation」

指定された名前と要求されたデータ処理単位の数で、キャパシティ予約を指定します。詳細については、「Amazon Athena ユーザーガイド」の「クエリ処理キャパシティの管理」、および「Amazon Athena API リファレンス」の「CreateCapacityReservation」を参照してください。

データカタログ

リファレンストピック:「AWS CloudFormation ユーザーガイド」の「AWS::Athena::DataCatalog」

名前、説明、型、パラメータ、およびタグを含めた Athena データカタログを指定します。詳細については、「Amazon Athena ユーザーガイド」の「テーブル、データベース、およびデータカタログの理解」、および「Amazon Athena API リファレンス」の「CreateDataCatalog」を参照してください。

名前付きクエリ

リファレンストピック:「AWS CloudFormation ユーザーガイド」の「AWS::Athena::NamedQuery」

AWS CloudFormation を使用して名前付きクエリを指定し、それらを Athena で実行します。名前付きクエリは、クエリ名をクエリにマップしてから、そのクエリを保存されたクエリとして Athena コンソールから実行することを可能にします。詳細については、「Amazon Athena ユーザーガイド」の「保存されたクエリの使用」、および「Amazon Athena API リファレンス」の「CreateNamedQuery」を参照してください。

プリペアドステートメント

参考トピック:「AWS CloudFormation ユーザーガイド」の「AWS::Athena::PreparedStatement」

Athena での SQL クエリで使用する準備済みステートメントを指定します。準備済みステートメントにはパラメータプレースホルダが含まれており、それらの値は実行時に提供されます。詳細については、「Amazon Athena ユーザーガイド」の「パラメータ化されたクエリの使用」、および「Amazon Athena API リファレンス」の「CreatePreparedStatement」を参照してください。

Workgroup

参照トピック:「AWS CloudFormation ユーザーガイド」の「AWS::Athena::WorkGroup」

AWS CloudFormation を使用して Athena ワークグループを指定します。Athena ワークグループは、ユーザーまたはユーザーのグループのクエリを同じアカウント内の他のクエリから分離するために使用します。詳細については、「Amazon Athena ユーザーガイド」の「ワークグループを使用してクエリのアクセスとコストを制御する」、および「Amazon Athena API リファレンス」の「CreateWorkGroup」を参照してください。

Amazon CloudFront

参照トピック: Amazon CloudFront ログのクエリ

Athena を使用して Amazon CloudFront ログをクエリします。CloudFront の使用に関する詳細については、「Amazon CloudFront デベロッパーガイド」を参照してください。

AWS CloudTrail

参照トピック: AWS CloudTrail ログのクエリ

CloudTrail ログでの Athena の使用は、AWS のサービスのアクティビティ分析を強化するための優れた手段です。たとえば、クエリを使用して傾向を識別したり、ソース IP アドレスやユーザーなど属性でアクティビティをさらに分離したりすることが可能です。CloudTrail コンソールからログを直接クエリするためのテーブルを作成して、これらのテーブルを Athena でのクエリの実行に使用できます。詳細については、「CloudTrail ログ用の Athena テーブルを作成するための CloudTrail コンソールの使用 」を参照してください。

Amazon DataZone

参照トピック: Athena で Amazon DataZone を使用する

「Amazon DataZone」を使用して組織の境界を越えてデータを大規模に共有、検索、検出します。DataZone は、Athena、AWS Glue、AWS Lake Formation などの AWS 分析サービス全体のエクスペリエンスを簡素化します。さまざまなデータソースに大量のデータがある場合、Amazon DataZone を使用して、人、データ、ツールをビジネスユースケースに基づいてグループ化できます。

Athena では、クエリエディターを使用して DataZone 環境にアクセスしてクエリを実行できます。詳細については、「Athena で Amazon DataZone を使用する」を参照してください。

Elastic Load Balancing

参照トピック: Application Load Balancer ログのクエリ

Application Load Balancer のログをクエリすることで、トラフィックの送信元、レイテンシー、Elastic Load Balancing インスタンスとバックエンドアプリケーションとの間で転送されるバイト数を確認できます。詳細については、「Application Load Balancer ログのクエリ」を参照してください。

参照トピック: Classic Load Balancer ログのクエリ

Classic Load Balancer ログをクエリして、Elastic Load Balancing インスタンスとバックエンドアプリケーションで送受信されるトラフィックのパターンを分析し、理解します。トラフィックの送信元、レイテンシー、および転送されたバイト数を確認できます。詳細については、「ELB ログ用のテーブルの作成」を参照してください。

Amazon EMR Studio

参照トピック: Use the Amazon Athena SQL editor in EMR Studio

EMR Studio で Athena を使用し、インタラクティブなクエリを開発して実行できます。これにより、Spark、Scala、その他のワークロードに使用するときと同じ Amazon EMR インターフェイスから、Athena 上の SQL 分析に EMR Studio を使用できるようになります。EMR Studio に Athena を統合すると、次のタスクを実行できます。

- Athena SQL クエリの実行
- クエリ結果を表示する
- クエリ履歴の表示
- 保存されたクエリの表示
- パラメータ化されたクエリの実行
- データカタログのデータベース、テーブル、ビューの表示

次の Athena 機能は Amazon EMR Studio では利用できません。

- Athena ワークグループ、データソース、キャパシティ予約の作成または更新などの管理者機能
- Athena for Spark または Spark ノートブック
- DataZone 統合
- Step Functions

EMR Studio と Athena の統合は、EMR Studio および Athena が利用可能なすべての AWS リージョン で利用可能です。EMR Studio での Athena の使用についての詳細は、「Amazon EMR 管理ガイド」の「Use the Amazon Athena SQL editor in EMR Studio」を参照してください。

AWS Glue Data Catalog

参照トピック: AWS Glue との統合

Athena は、Amazon S3 内のデータに永続的なメタデータストアを提供する AWS Glue Data Catalog と統合します。これにより、Amazon Web Services アカウント全体で利用でき、AWS Glue の ETL およびデータ検出機能と統合された中央メタデータストアに基づいて、Athena での

テーブルの作成とデータのクエリを行うことが可能になります。詳細については、「AWS Glue との統合」と、「AWS Glue デベロッパーガイド」の「AWS Glue とは」を参照してください。

AWS Identity and Access Management (IAM)

リファレンストピック: Amazon Athena 向けのアクション

IAM アクセス許可ポリシーで Athena API アクションを使用できます。詳細については、「Amazon Athena 向けのアクション」と「Athena でのアイデンティティとアクセス権の管理」を参照してください。

Amazon QuickSight

参照トピック: ODBC および JDBC ドライバーを使用した Amazon Athena への接続

Athena は Amazon QuickSight と統合して、データを簡単に可視化できるようにします。Athena を使用して、JDBC や ODBC ドライバーで接続されたビジネスインテリジェンスツールや SQL クライアントでレポートを生成、またはデータを探索できます。Amazon QuickSight の詳細については、「Amazon QuickSight ユーザーガイド」の「Amazon QuickSight とは」を参照してください。Athena での JDBC および ODBC ドライバーの使用に関する詳細については、「ODBC および JDBC ドライバーを使用した Amazon Athena への接続」を参照してください。

Amazon S3 インベントリ

リファレンストピック:「Amazon Simple Storage Service ユーザーガイド」の「Amazon Athena でインベントリをクエリする」

Amazon Athena を使用して、標準 SQL を使用した Amazon S3 インベントリのクエリを実行できます。Amazon S3 インベントリは、ビジネス、コンプライアンス、および規制上のニーズに関するオブジェクトのレプリケーションと暗号化のステータスを監査し、報告するために使用できます。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 インベントリ」を参照してください。

AWS Step Functions

リファレンストピック: AWS Step Functions デベロッパーガイドの「Step Functions で Athena を呼び出す」

Athena を呼び出すため AWS Step Functions を使用します。AWS Step Functions では、Amazon States Language を使用することで、AWS のサービス の選択を直接制御できます。Step Functions を Athena で使用して、クエリの実行の開始と停止、クエリ結果の取得、アドホックまたはスケジュールされたデータクエリの実行、Simple Storage Service (Amazon S3) 内のデータレイクからの結果の取得を行うことができます。Step Functions ロールには Athena を使用する

権限が必要です。詳細については、「AWS Step Functions デベロッパーガイド」を参照してください。

動画: AWS Step Functions を使用した Amazon Athena クエリのオーケストレーション

以下の動画では、Amazon Athena と AWS Step Functions を使用して定期的にスケジュールされた Athena クエリを実行し、対応するレポートを生成する方法が紹介されています。

AWS Step Functions を使用した Amazon Athena クエリのオーケストレーション

Step Functions と Amazon EventBridge を使用して AWS Glue DataBrew、Athena、および Amazon QuickSight のオーケストレーションを実行する例については、「AWS ビッグデータブログ」の「AWS Step Functions を使用した AWS Glue DataBrew ジョブと Amazon Athena クエリ のオーケストレーション」を参照してください。

AWS Systems Manager インベントリ

リファレンストピック:「AWS Systems Manager ユーザーガイド」の「複数のリージョンとアカウントからのインベントリデータをクエリする」

AWS Systems Manager インベントリは Amazon Athena と統合されており、複数の AWS リージョン およびアカウントからインベントリデータをクエリできます。詳細については、「AWS Systems Managerユーザーガイド」を参照してください。

Amazon Virtual Private Cloud

参照トピック: Amazon VPC フローログのクエリ

Amazon Virtual Private Cloud フローログは、VPC 内のネットワークインターフェイス間で送受信される IP トラフィックに関する情報を取得します。Athena のログをクエリしてネットワークトラフィックパターンを調査し、Amazon VPC ネットワーク全体の脅威とリスクを特定します。Amazon VPC の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

セットアップ

Amazon Web Services に既にサインアップしている場合は、Amazon Athena の使用を今すぐ開始できます。AWS にまだサインアップしていない場合、または開始するのにサポートが必要な場合は、必ず次のタスクを完了します。

AWS アカウントへのサインアップ

AWS アカウントがない場合は、以下のステップを実行して作成します。

AWS アカウントにサインアップするには

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。

   AWS アカウントにサインアップすると、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。

サインアップ処理が完了すると、AWS からユーザーに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

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

AWS アカウントのルートユーザーをセキュリティで保護する

1. [ルートユーザー] を選択し、AWS アカウントのメールアドレスを入力して、アカウント所有者として AWS Management Console 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 の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許可します。

   IAM アイデンティティセンターディレクトリ をアイデンティティソースとして使用するチュートリアルについては、「AWS IAM Identity Center ユーザーガイド」の「デフォルト IAM アイデンティティセンターディレクトリを使用したユーザーアクセスの設定」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「Create a permission set」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「Add groups」を参照してください。

プログラム的なアクセス権を付与する

AWS Management Console の外部で AWS を操作するには、プログラマチックアクセス権が必要です。プログラマチックアクセス権を付与する方法は、AWS にアクセスしているユーザーのタイプによって異なります。

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 CLI、AWS SDK、または AWS API へのプログラマチックリクエストに署名します。	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。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「AWS IAM Identity Center を使用するための AWS CLI の設定」を参照してください。 • AWS SDK、ツール、および AWS API については、AWS SDK とツールリファレンスガイドの「IAM Identity Center 認証」を参照してください。
IAM	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラムによるリクエストに署名します。	「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」の指示に従ってください。
IAM	(非推奨) 長期的な認証情報を使用して、AWS CLI、AWS SDK、AWS API へのプログラムによるリクエストに署名します。	使用するインターフェイス用の手順に従ってください。 • AWS CLI については、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 SDK とツールについては、AWS SDK とツールリファレンスガイドの「長

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
		期認証情報を使用して認証する」を参照してください。 • AWS API については、IAM 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。

Athena を使用するためのマネージドポリシーをアタッチする

Athena マネージドポリシーは Athena 機能を使用するアクセス許可を付与します。これらのマネージドポリシーを、ユーザーが Athena を使用するために割り当てられる 1 つ以上の IAM ロールにアタッチできます。

IAM ロールは、特定の許可があり、アカウントで作成できるもう 1 つの IAM アイデンティティです。IAM ロールは、ID が AWS で実行できることとできないことを決定する許可ポリシーを持つ AWS ID であるという点で IAM ユーザーと似ています。ただし、ユーザーは 1 人の特定の人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。また、ロールには標準の長期認証情報 (パスワードやアクセスキーなど) も関連付けられません。代わりに、ロールを引き受けると、ロールセッション用の一時的なセキュリティ認証情報が提供されます。

ロールの詳細については、「IAM ユーザーガイド」の「IAM ロール」および「IAM ロールの作成」を参照してください。

Athena へのアクセスを付与するロールを作成するには、そのロールに Athena マネージドポリシーをアタッチします。Athena には `AmazonAthenaFullAccess` と `AWSQuicksightAthenaAccess` の 2 つのマネージドポリシーがあります。これらのポリシーは、代わりに、Amazon S3 にクエリを実行し、クエリ結果を別のバケットに書き込むためのアクセス許可を Athena に付与します。Athena のポリシーの内容を確認するには、「Amazon Athena の AWS 管理ポリシー」を参照してください。

Athena マネージドポリシーをロールにアタッチする手順については、「IAM ユーザーガイド」の「IAM ID アクセス許可の追加 (コンソール)」に従って、AmazonAthenaFullAccess および AWSQuicksightAthenaAccess のマネージドポリシーを、作成したロールに追加します。

> **Note**
>
> Amazon S3 内の基盤となるデータセットにアクセスするには、追加の許可が必要になる場合があります。アカウントの所有者ではないか、バケットへのアクセスが制限されている場合は、バケット所有者に連絡してリソースベースのバケットポリシーを使用するアクセス権を付与してもらうか、アカウント管理者に連絡してロールベースのポリシーを使用するアクセス権を付与してもらいます。詳細については、「Amazon S3 へのアクセス権」を参照してください。データセットまたは Athena クエリ結果が暗号化されている場合は、追加の許可が必要になる場合があります。詳細については、「保管中の暗号化」を参照してください。

Athena へのアクセス

Athena へのアクセスには、AWS Management Console、JDBC または ODBC 接続、Athena API、Athena CLI、AWS SDK、または AWS Tools for Windows PowerShell を使用できます。

- コンソールで Athena SQL の使用を開始するには、「開始」を参照してください。
- Python を使用する Jupyter 互換ノートブックと Apache Spark アプリケーションを作成するには、「Amazon Athena での Apache Spark の使用」を参照してください。
- JDBC または ODBC ドライバーの使用方法については、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。
- Athena API を使用するには、Amazon Athena API リファレンスを参照してください。
- CLI を使用するには、AWS CLI をインストールし、次にコマンドラインから「`aws athena help`」と入力して使用可能なコマンドを確認します。使用可能なコマンドの詳細については、Amazon Athena コマンドラインリファレンスを参照してください。
- AWS SDK for Java 2.x を使用するには、「AWS SDK for Java 2.x API リファレンス」の Athena の項、GitHub.com の Athena Java V2 の例、および「AWS SDK for Java 2.x デベロッパーガイド」を参照してください。
- AWS SDK for .NET を使用するには、AWS SDK for .NET API リファレンスの Amazon.Athena 名前空間、GitHub.com の .NET Athena の例、および AWS SDK for .NET デベロッパーガイドを参照してください。

- AWS Tools for Windows PowerShell を使用するには、AWS Tools for PowerShell - Amazon Athena コマンドレットのリファレンス、AWS Tools for PowerShell ポータルページ、および AWS Tools for Windows PowerShell ユーザーガイドを参照してください。
- プログラム的に接続できる Athena サービスエンドポイントについては、Amazon Web Services 全般のリファレンス の「Amazon Athena エンドポイントとクォータ」を参照してください。

Athena SQL の使用

Athena SQL を使用して、AWS Glue Data Catalog、外部 Hive メタストア、または他のデータソースへのさまざまな事前構築済みコネクタを使用したフェデレーションクエリを使用して、Amazon S3 でデータをインプレースでクエリできます。

以下の操作も可能です。

- ビジネスインテリジェンスツールやその他アプリケーションに Athena の JDBC ドライバーおよび ODBC ドライバーを使用します。
- AWS サービスログをクエリします。
- タイムトラベルクエリを含む Apache Iceberg テーブルおよび Apache Hudi データセットのクエリを実行します。
- 地理空間データをクエリします。
- Amazon SageMaker からの機械学習推論を使用してクエリを実行します。
- 独自のユーザー定義関数を使用してクエリを実行します。
- パーティションプロジェクションを使用して、高度にパーティションされたテーブルのクエリ処理を高速化してパーティション管理を自動化します。

トピック

- Athena の SQL リファレンス
- Athena のトラブルシューティング
- コードサンプル

テーブル、データベース、およびデータカタログの理解

Athena では、カタログ、データベース、およびテーブルは、基盤となるソースデータのスキーマを定義するメタデータ定義のコンテナです。

Athena はデータオブジェクトの階層を指すのに次の用語を使います。

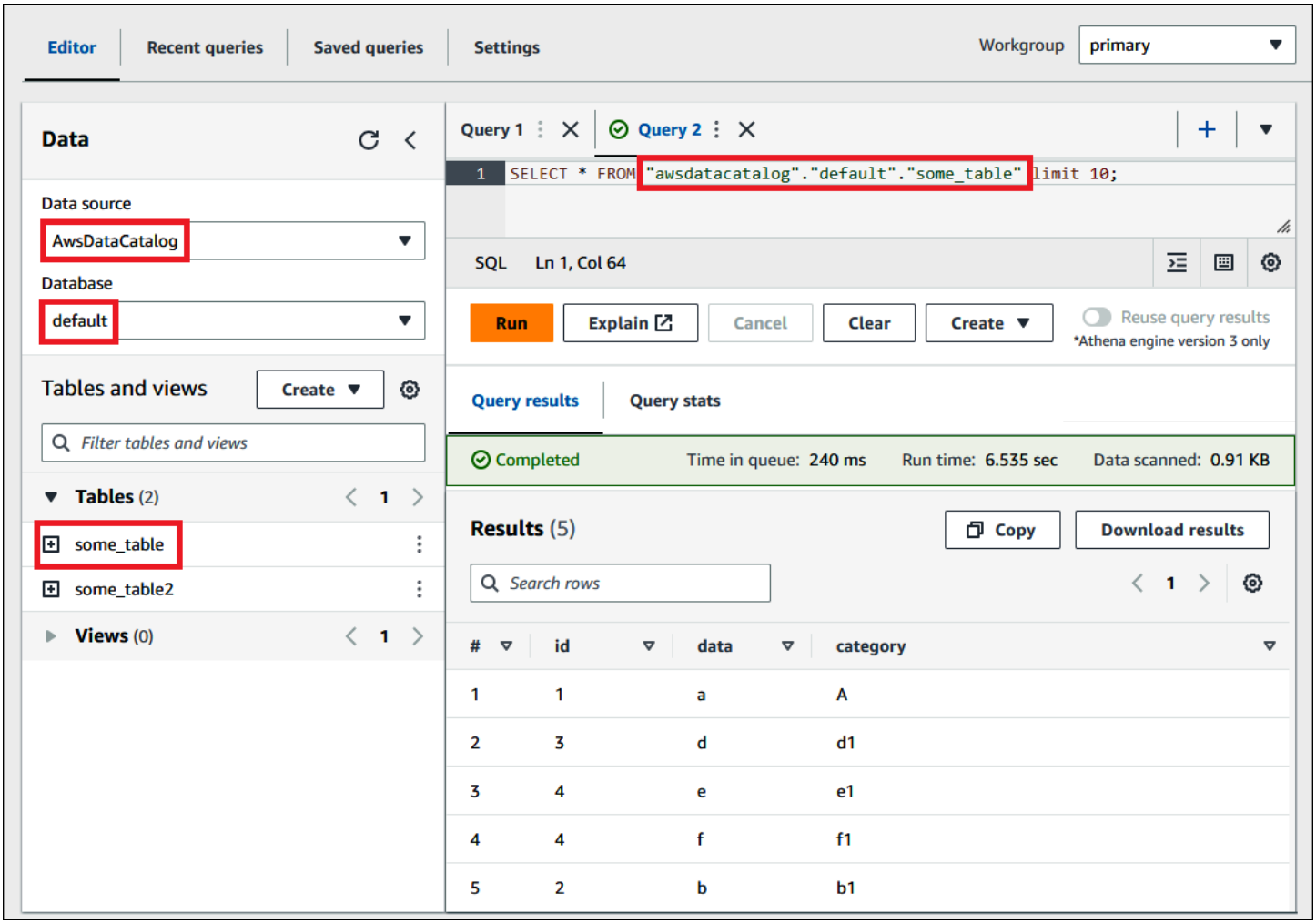
- データソース — データベースのグループ
- データベース — テーブルのグループ
- テーブル — 行または列のグループとして整理されたデータ

これらのオブジェクトは、次のように代替だが同等の名前で呼ばれることもあります。

- データソースはカタログと呼ばれることもあります。
- データベースはスキーマと呼ばれることもあります。

> **Note**
>
> この用語は、Athena で使用するフェデレーションデータソースによって異なる場合があります。詳細については、「Athena とフェデレーションテーブル名修飾子」を参照してください。

Athena コンソールの次のクエリ例では、`awsdatacatalog` データソース、`default` データベース、および `some_table` テーブルを使用しています。



Athena では、データセットごとにテーブルが存在する必要があります。テーブルのメタデータは、Amazon S3 のどこにデータがあるかを Athena に伝え、列名、データ型、およびテーブルの名前などのデータの構造を指定します。データベースは、テーブルを論理的なグループにまとめたものであり、データセットのメタデータおよびスキーマ情報のみを保持します。

Athena では、クエリするデータセットごとに、クエリ結果を取得して返すために Athena が使用する基盤となるテーブルが必要です。このため、データをクエリする前に、Athena にテーブルを登録しておく必要があります。登録は、テーブルの自動作成または手動作成に伴って行われます。

AWS Glue クローラーを使用してテーブルを自動的に作成できます。AWS Glue およびクローラの詳細については、「AWS Glue との統合」を参照してください。AWS Glue がテーブルを作成するときは、独自の AWS Glue データカタログにテーブルを登録します。Athena は、AWS Glue データカタログを使用してこのメタデータの保存と取得を行い、基盤となるデータセットを分析するためのクエリを実行するときにそれを使用します。

テーブルの作成方法にかかわらず、テーブル作成プロセスによってデータセットが Athena に登録されます。登録は AWS Glue Data Catalog で行われ、Athena がデータをクエリすることを可能にします。Athena クエリエディタでは、このカタログ (またはデータソース) は、ラベル `AwsDataCatalog` で参照されます。

テーブルの作成後は、SQL SELECT ステートメントを使用してテーブルをクエリすることができます。これには、ソースデータの特定のファイルの場所の取得が含まれます。クエリの結果は、Amazon S3 にあるユーザー指定のクエリ結果の場所に保存されます。

Amazon Web Services アカウント全体で AWS Glue データカタログにアクセスできます。AWS Glue データカタログは他の AWS のサービス と共有できるため、Athena を使用して組織全体で作成されたデータベースやテーブルを確認できます。また、その逆も可能です。

- テーブルを手動で作成する方法は以下のとおりです。
  - Athena コンソールを使用して、テーブル作成ウィザードを実行する。
  - Athena コンソールを使用して、クエリエディタに Hive DDL ステートメントを記述する。
  - Athena API または CLI を使用して、DDL ステートメントで SQL クエリ文字列を実行する。
  - Athena JDBC または ODBC ドライバーを使用する。

テーブルとデータベースを手動で作成すると、Athena が内部で `CREATE TABLE`、`CREATE DATABASE`、`DROP TABLE` などの HiveQL データ定義言語 (DDL) ステートメントを使用して、AWS Glue Data Catalog にテーブルとデータベースを作成します。

始めるには、Athena コンソールのチュートリアルを使用するか、Athena ドキュメントのステップバイステップガイドで作業することができます。

- Athena コンソールでチュートリアルを使用するには、コンソールの右上にある情報アイコンを選択し、[チュートリアル]タブを選択します。
- テーブルの作成と Athena クエリエディタでのクエリの記述に関するステップバイステップのチュートリアルについては、「開始」を参照してください。

開始

このチュートリアルは、Amazon Athena を使用したデータのクエリについて詳しく説明するもので、Amazon Simple Storage Service に保存されたサンプルデータに基づいたテーブルを作成し、テーブルをクエリして、クエリ結果を確認します。

チュートリアルではライブリソースを使用するため、実行したクエリに対する料金が発生します。このチュートリアルが使用する場所にあるサンプルデータに対する料金は発生しませんが、独自のデータファイルを Amazon S3 にアップロードする場合は、料金が適用されます。

前提条件

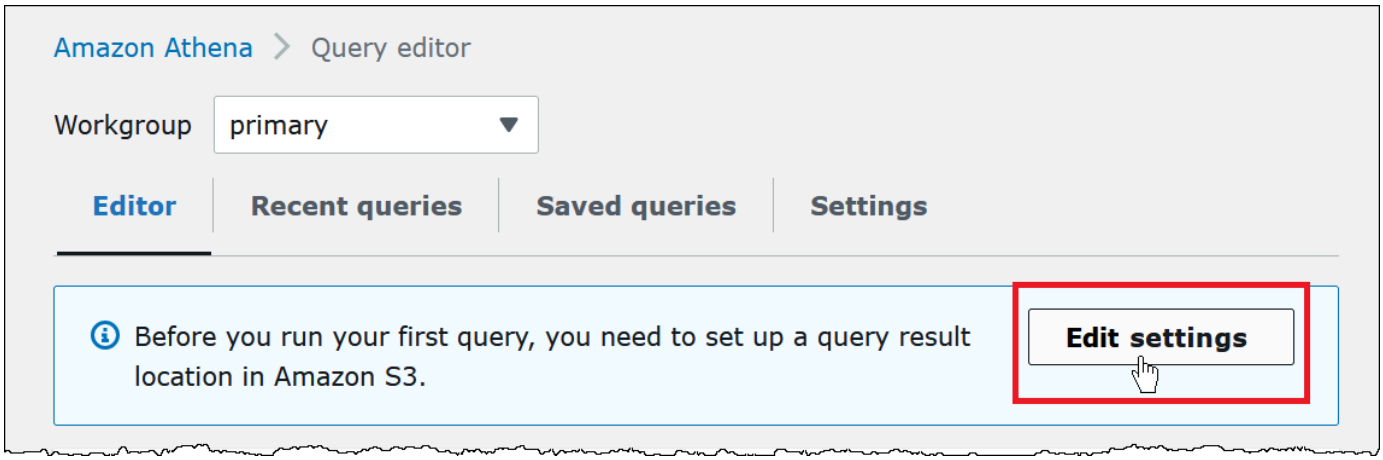
- まだサインアップしていない場合は、AWS アカウントにサインアップします。
- Athena で使用しているものと同じ AWS リージョン (米国西部 (オレゴン) など) とアカウントを使用して、Athena からのクエリ結果を保存するバケットを Amazon S3 に作成するためのステップを実行します。このバケットをクエリ出力の場所として設定します。

ステップ 1: データベースを作成する

まず、Athena でデータベースを作成する必要があります。

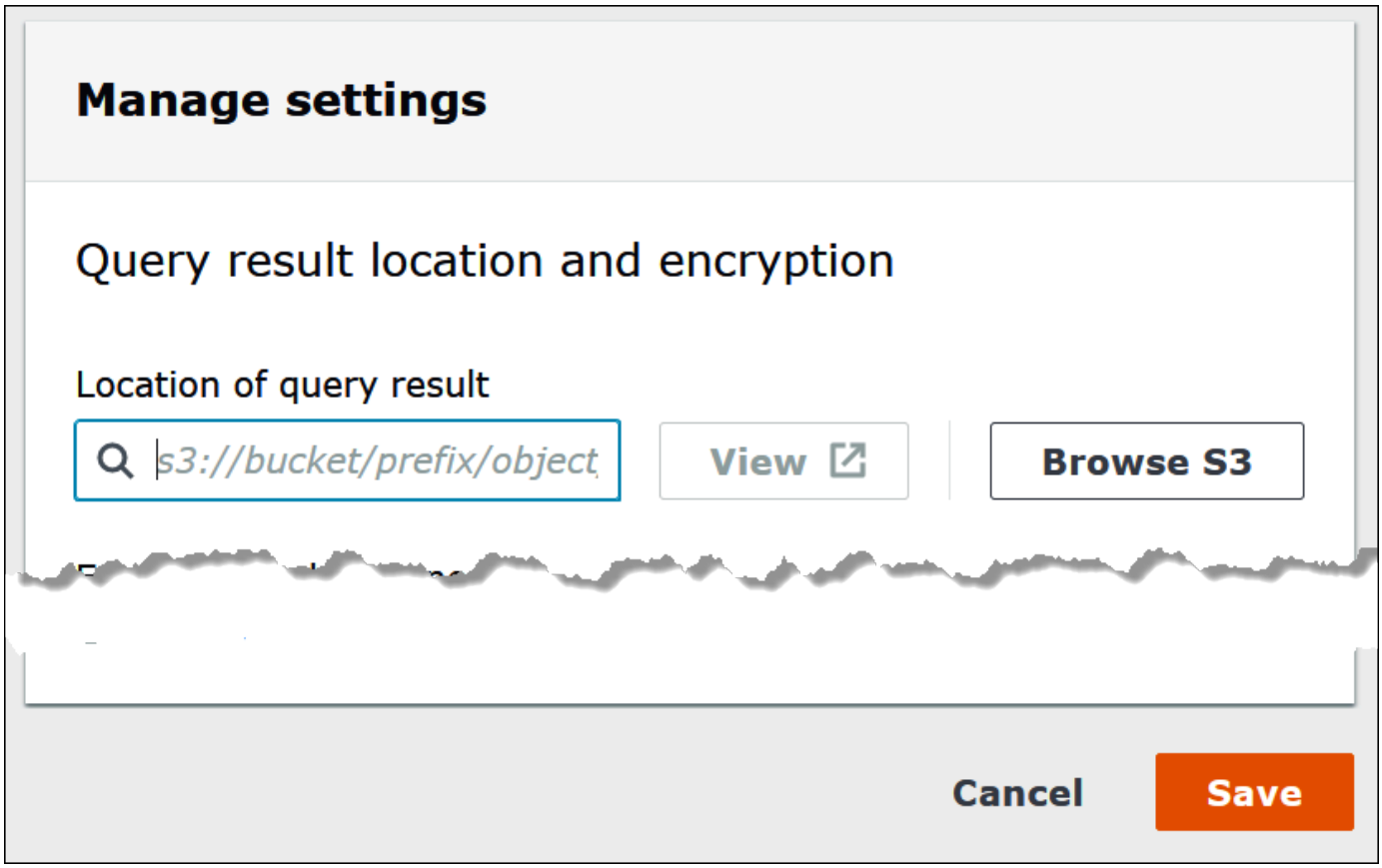
Athena データベースを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. 現在の AWS リージョン で Athena コンソールを初めて使用する場合は、[Explore the query editor] (クエリエディタを試す) をクリックしてクエリエディタを開きます。初めてではない場合は、クエリエディタで Athena が開きます。
3. [Edit Settings] (設定を編集) をクリックし、Amazon S3 内のクエリ結果の保存場所をセットアップします。

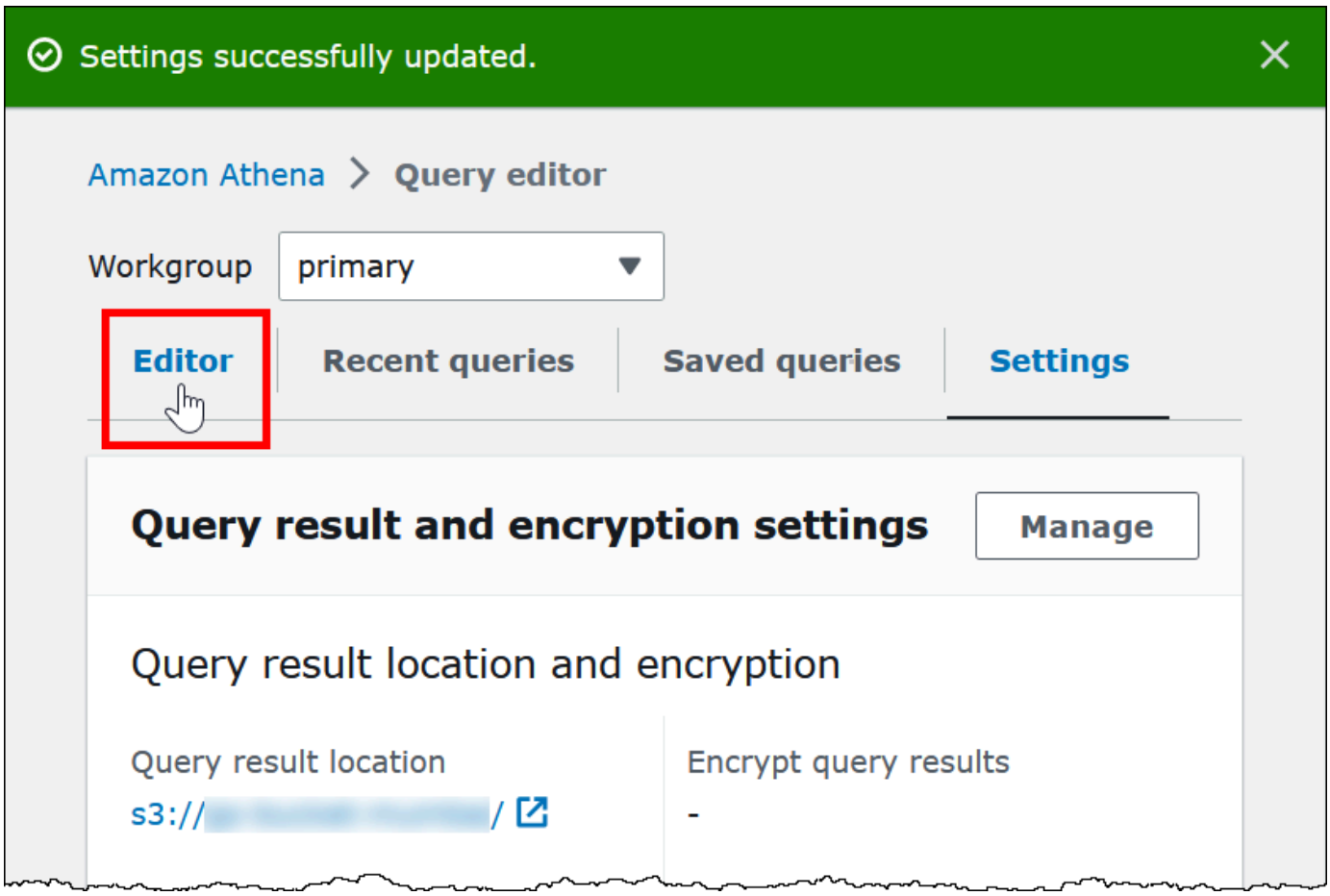


4. [Manage settings] (設定の管理) で、以下のいずれかを実行します。
   - [Location of query result] (クエリ結果の場所) ボックスで、クエリ結果のために Amazon S3 に作成したバケットへのパスを入力します。パスの先頭に `s3://` を付けます。

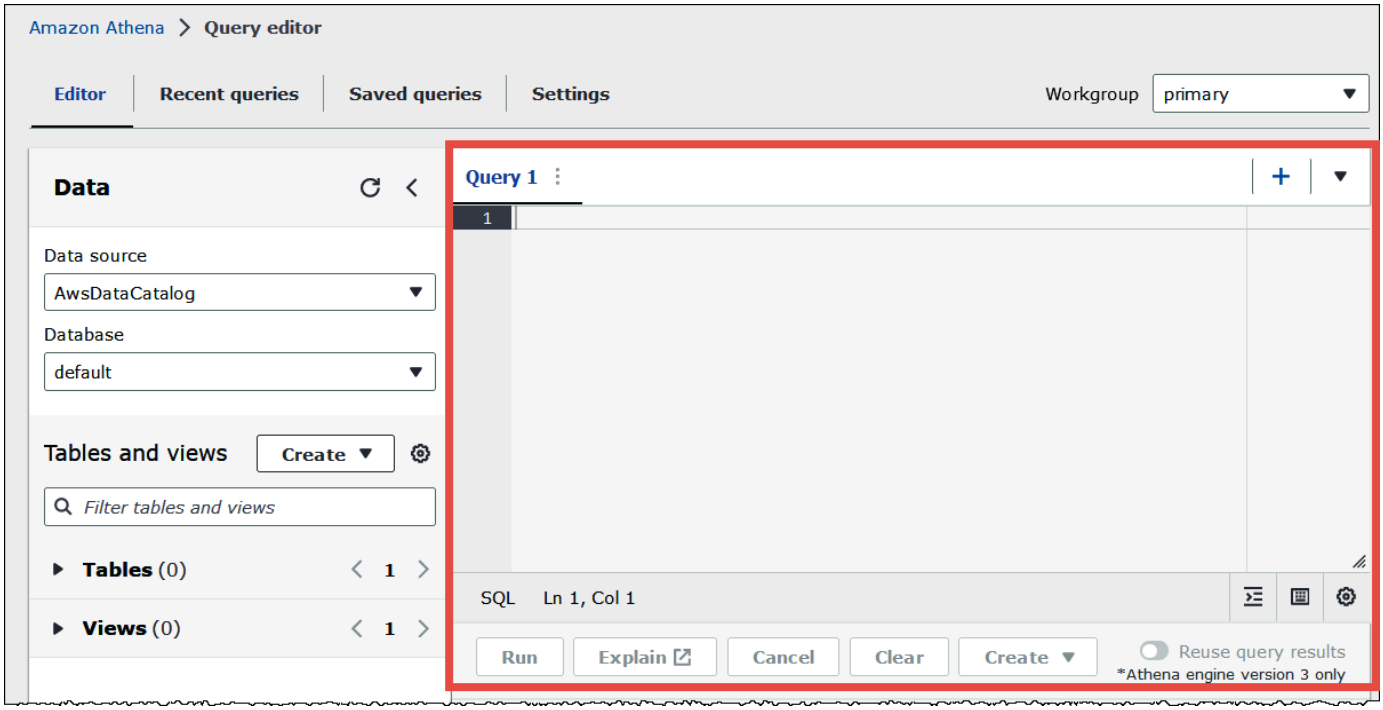
- [Browse S3] (S3 をブラウズ) をクリックし、現在のリージョンで作成した Amazon S3 バケットを選択した上で、[Choose] (選択) をクリックします。



5. [Save] を選択します。
6. [Editor] (エディタ) をクリックして、クエリエディタに切り替えます。



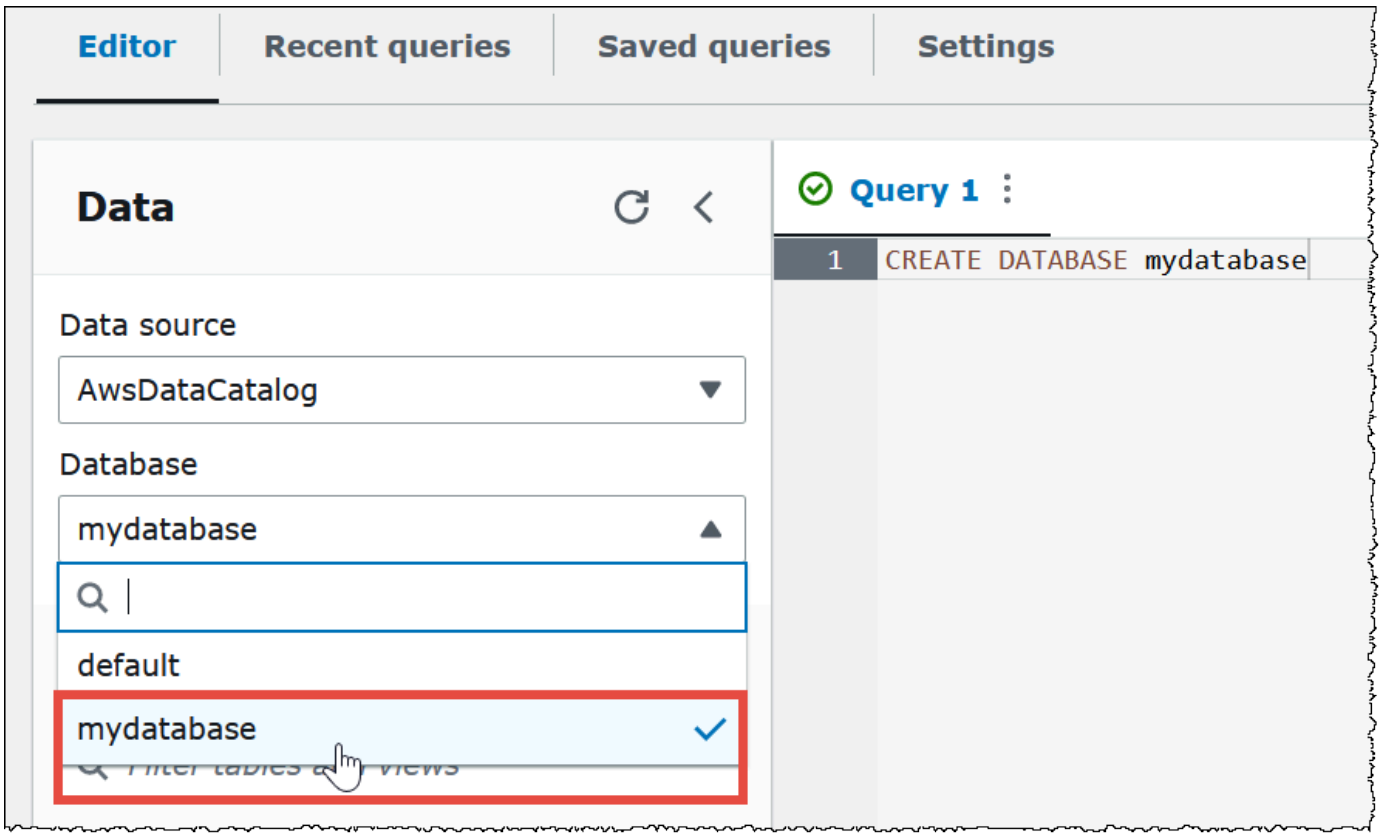
7. ナビゲーションペインの右側から、Athena クエリエディタを使用してクエリとステートメントの入力と実行が行えます。



8. mydatabase という名前のデータベースを作成するには、次の CREATE DATABASE ステートメントを入力します。

```
CREATE DATABASE mydatabase
```

9. [Run] (実行) をクリックするか、**Ctrl+ENTER** キーを押します。
10. 左側の [Database] (データベース) リストから、現在のデータベースとして mydatabase を選択します。



ステップ 2: テーブルを作成する

データベースが作成されたので、Athena テーブルを作成できます。作成するテーブルは、ロケーション `s3://athena-examples-myregion/cloudfront/plaintext/` にあるサンプルの Amazon CloudFront ログデータをベースにします。ここで、*myregion* は現在の AWS リージョンです。

サンプルログデータは、タブ区切り (TSV) 形式で、フィールドがタブ文字で区切られています。結果は以下の例のようになります。読みやすくするため、抜粋箇所のタブがスペースに変換され、最後のフィールドが短縮されています。

```
2014-07-05 20:00:09 DFW3 4260 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-1.jpeg 200 - Mozilla/5.0[...]
2014-07-05 20:00:09 DFW3 4252 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-2.jpeg 200 - Mozilla/5.0[...]
2014-07-05 20:00:10 AMS1 4261 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-3.jpeg 200 - Mozilla/5.0[...]
```

Athena がこのデータを読み取れるようにするには、以下にあるような単純な CREATE EXTERNAL TABLE ステートメントを作成できます。テーブルを作成するステートメントは、データにマッピングする列を定義し、データを区切る方法を指定して、サンプルデータが含まれる Amazon S3 の場所を指定します。Athena はフォルダ内のすべてのファイルをスキャンすることを想定しているため、LOCATION 句は特定のファイルではなく、Amazon S3 フォルダの場所を指定します。

この後すぐに説明する重要な制限があることから、この例はまだ使用しないでください。

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  ClientInfo STRING
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  LINES TERMINATED BY '\n'
  LOCATION 's3://athena-examples-my-region/cloudfront/plaintext/';
```

この例は、cloudfront_logs という名前のテーブルを作成し、各フィールドの名前とデータ型を指定します。これらのフィールドがテーブルの列になります。date は予約語であるため、バッククォート (`) 文字で エスケープされます。ROW FORMAT DELIMITED は、Athena が実際のデータ解析作業の実行に LazySimpleSerDe という名前のデフォルトライブラリを使用することを意味します。この例は、フィールドがタブ区切り (FIELDS TERMINATED BY '\t') であり、ファイル内の各レコードが改行文字 (LINES TERMINATED BY '\n) で終わることも指定します。最後に、LOCATION 句が、読み取られる実際のデータが格納されている Amazon S3 のパスを指定します。

独自のタブ区切りまたはカンマ区切りのデータがある場合は、フィールドにネストされた情報が含まれていない限り、先程紹介した例のような CREATE TABLE ステートメントを使用できます。ただし、異なる区切り形式を使用するネストされた情報が含まれる ClientInfo のような列がある場合は、異なる方法を使用する必要があります。

ClientInfo フィールドからのデータの抽出

サンプルデータでは、以下が最後のフィールドである ClientInfo の完全な例になります。

```
Mozilla/5.0%20(Android;%20U;%20Windows%20NT%205.1;%20en-US;
%20rv:1.9.0.9)%20Gecko/2009040821%20IE/3.0.9
```

ご覧のとおり、このフィールドは複数値フィールドです。先程紹介した CREATE TABLE ステートメント例はフィールドの区切り形式としてタブを指定していることから、ClientInfo フィールド内の個別のコンポーネントを個別の列に分割できません。そのため、新しい CREATE TABLE ステートメントが必要になります。

ClientInfo フィールド内の値から列を作成するには、正規表現 (regex) グループを含む regex を使用できます。指定する regex グループは、個別のテーブル列になります。CREATE TABLE ステートメントで正規表現を使用するには、次のような構文を使用します。この構文によって、Athena が Regex SerDe ライブラリと指定した正規表現を使用するように指示されます。

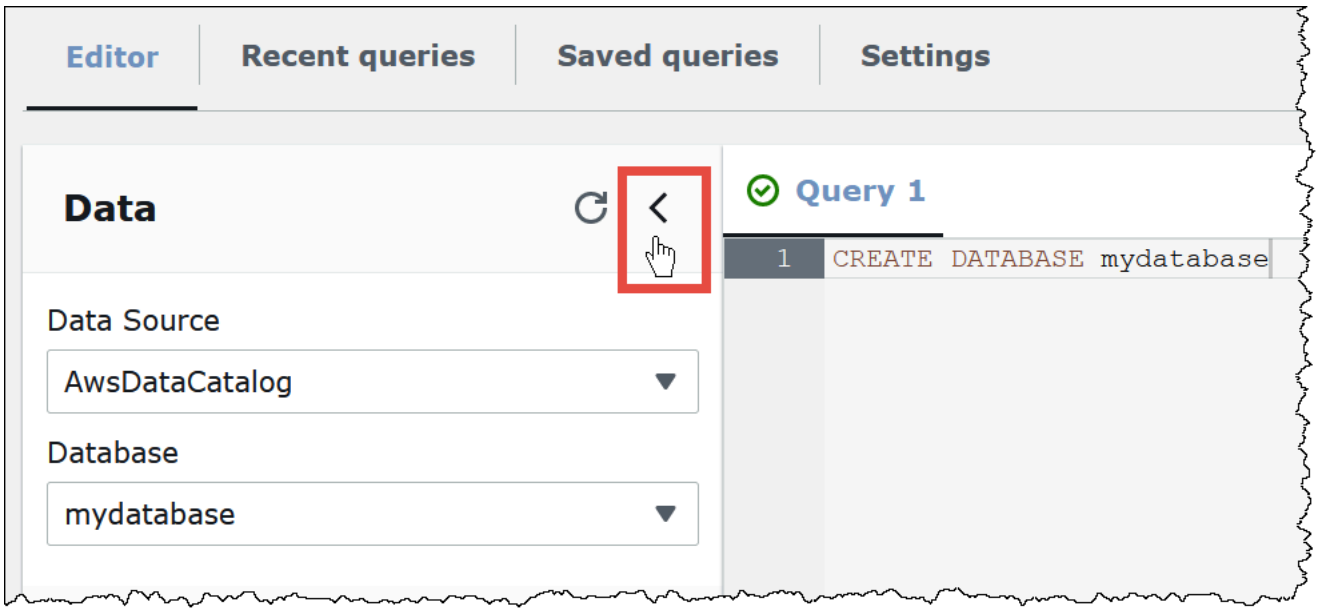
```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES ("input.regex" = "regular_expression")
```

正規表現は、複合型の CSV データまたは TSV データからのテーブルの作成に便利ですが、記述や保守が困難な場合があります。幸いに、JSON、Parquet、および ORC といった形式に使用できるライブラリは他にもあります。詳細については、「サポートされる SerDes とデータ形式」を参照してください。

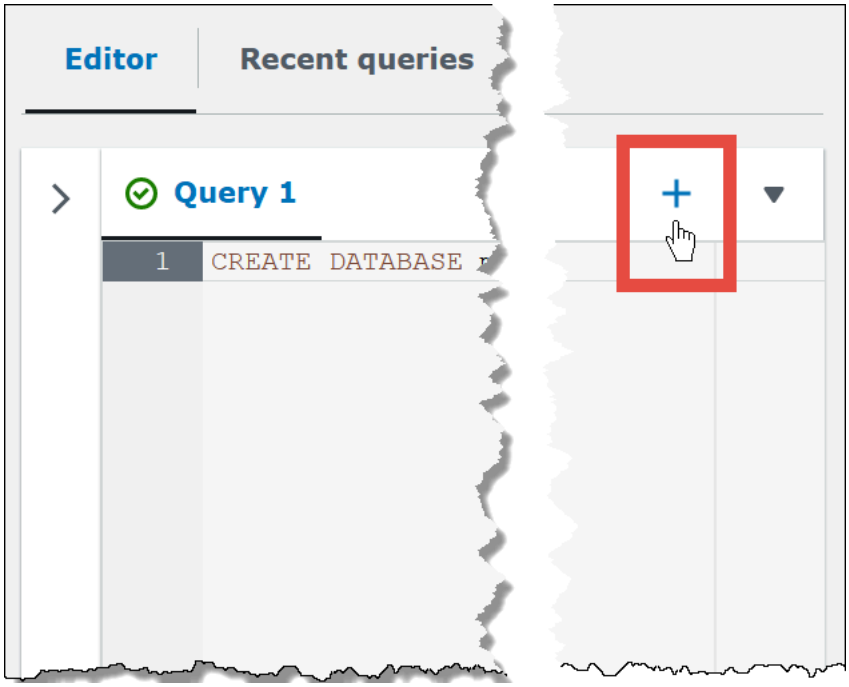
これで、Athena クエリエディタでテーブルを作成する準備が整いました。CREATE TABLE ステートメントと正規表現が用意されています。

Athena でテーブルを作成するには

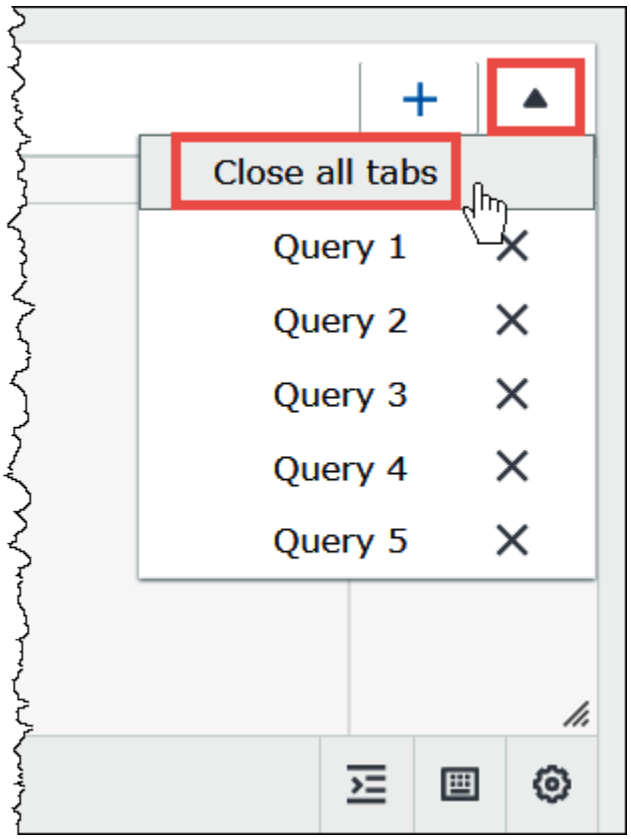
1. ナビゲーションペインにある [Database] (データベース) で、mydatabase が選択されていることを確認します。
2. クエリエディタ内で表示スペースを拡大するために、矢印アイコンを選択してナビゲーションペインを折りたたむことができます。



3. クエリエディタでプラス記号 (+) をクリックして、新しいクエリのタブを作成します。一度に最大 10 個のクエリタブを開くことができます。



4. 1 つ以上のクエリタブを閉じるには、プラス記号の横にある矢印をクリックします。すべてのタブを一度に閉じるには、矢印を選択してから、[Close all tabs] (すべてのタブを閉じる) をクリックします。



5. クエリペインに、次の CREATE EXTERNAL TABLE クエリを入力します。この正規表現で、ログデータの ClientInfo フィールドから、オペレーティングシステム、ブラウザ、およびブラウザバージョンの情報を抜き出すことができます。

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  os STRING,
  Browser STRING,
  BrowserVersion STRING
  )
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES (
```

```
  "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s
+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^
\/]+)[\/](.*)$"
  ) LOCATION 's3://athena-examples-myregion/cloudfront/plaintext/';
```

6. LOCATION ステートメントの中の *myregion* を、現在使用している AWS リージョン (us-west-1 など) に置き換えます。
7. [Run] (実行) を選択します。

   テーブル cloudfront_logs が作成され、mydatabase データベースの [Tables] (テーブル) リストに表示されます。

ステップ 3: データをクエリする

Amazon S3 内のデータに基づいて Athena で cloudfront_logs テーブルを作成したところで、次は Athena でこのテーブルに SQL クエリを実行して、結果を確認できます。Athena での SQL の使用に関する詳細については、「Athena の SQL リファレンス」を参照してください。

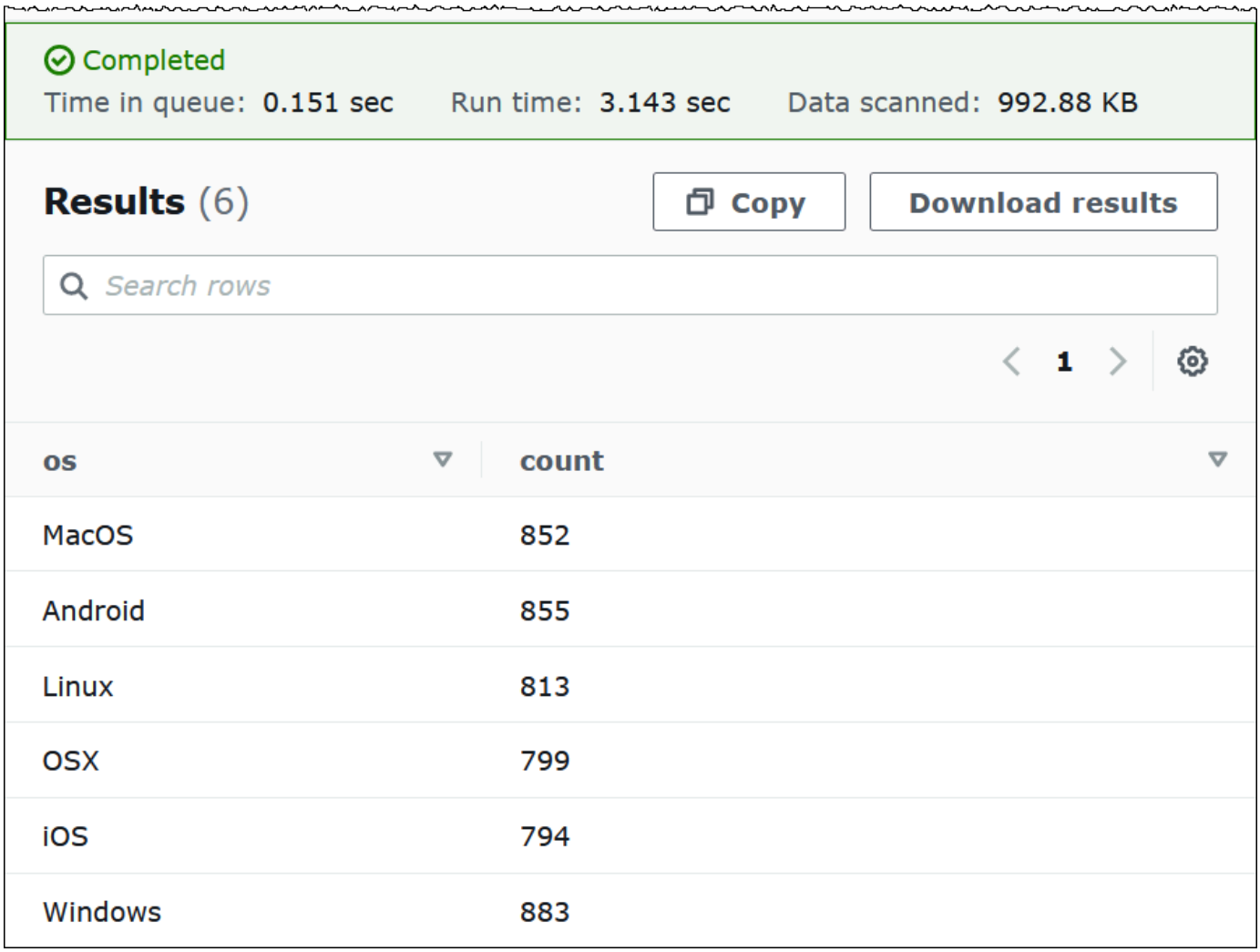
クエリを実行する

1. プラス (+) 記号をクリックして新しいクエリタブを開き、クエリペインに次の SQL ステートメントを入力します。

```
SELECT os, COUNT(*) count
FROM cloudfront_logs
WHERE date BETWEEN date '2014-07-05' AND date '2014-08-05'
GROUP BY os
```

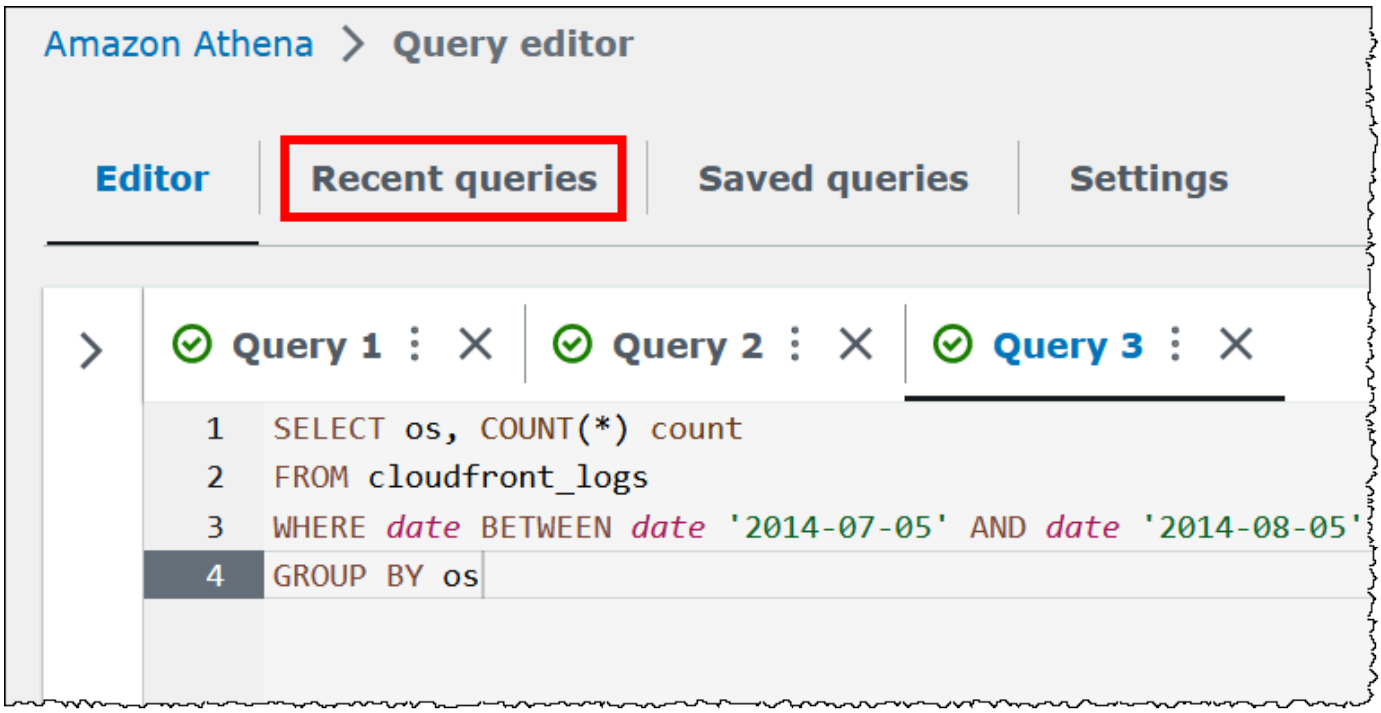
2. [Run] (実行) を選択します。

   結果は以下のようになります。

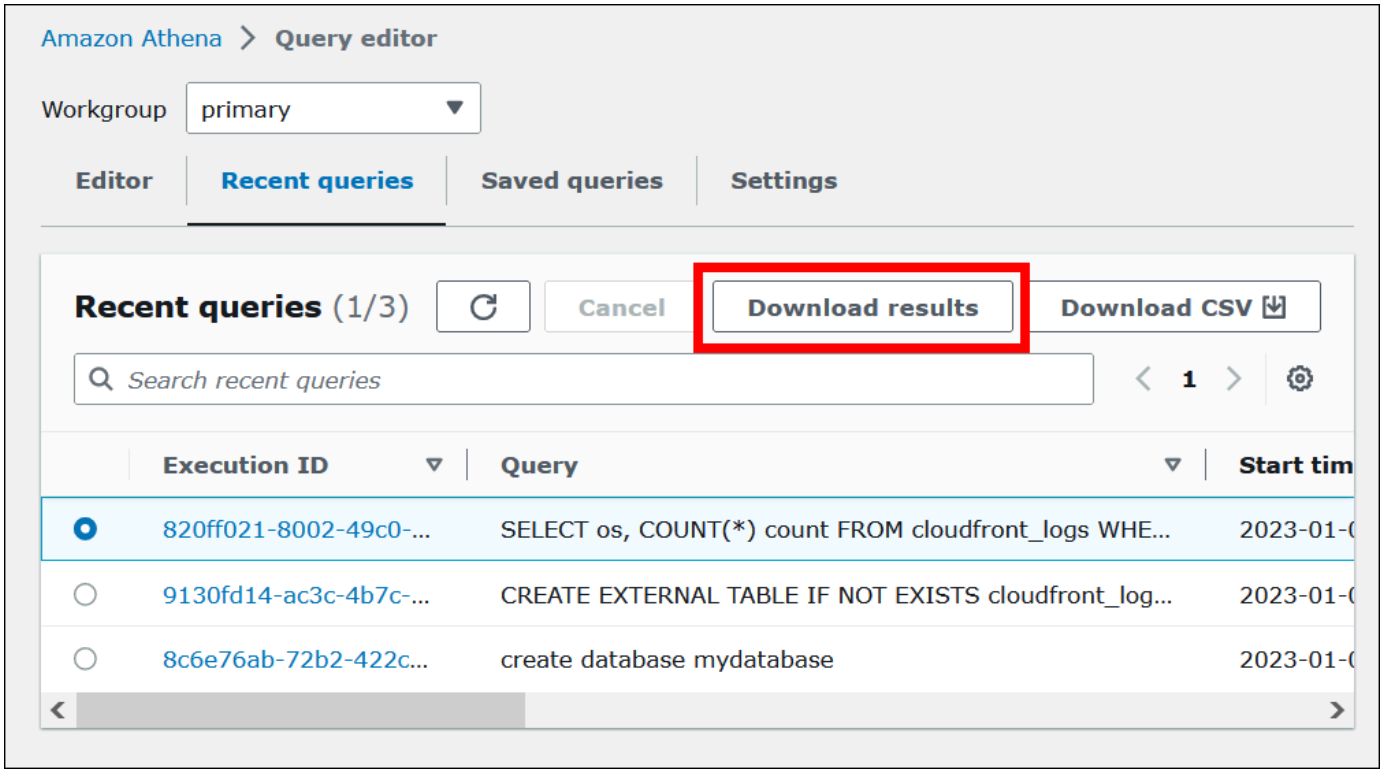


3. クエリ結果を `.csv` ファイルに保存するには、[Download results] (結果をダウンロード) をクリックします。

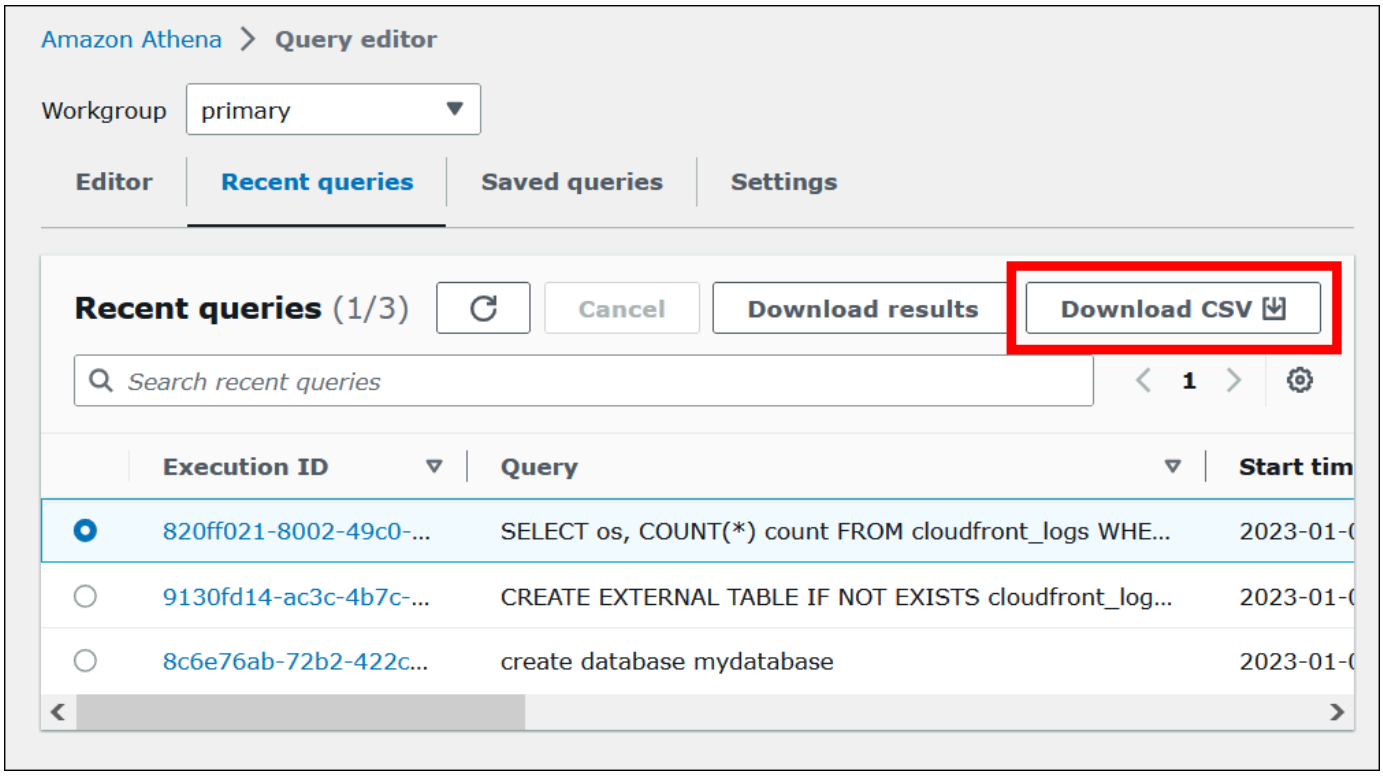
4. 以前のクエリを表示または実行するには、[Recent queries] (最近のクエリ) タブを開きます。



5. 以前のクエリの結果を、[Recent queries] (最近のクエリ) タブからダウンロードするには、クエリを選択した上で、[Download results] (結果をダウンロード) をクリックします。クエリは 45 日間保持されます。



6. 1 つ以上の最新の SQL クエリ文字列を CSV ファイルにダウンロードするには、[Download CSV] (CSV をダウンロード) を選択します。



詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。

クエリを保存する

クエリエディタで作成または編集したクエリは、名前を付けて保存できます。Athena はこれらのクエリを [Saved queries] (保存されたクエリ) タブ。[Saved queries] (保存されたクエリ) タブを使用して、保存したクエリの呼び出し、実行、名前変更、または削除をおこなうことができます。詳細については、「保存されたクエリの使用」を参照してください。

キーボードショートカットと先行入力候補

Athena クエリエディタには、クエリの実行、クエリの書式設定、行操作、検索と置換などの操作のための、多数のキーボードショートカットが用意されています。詳細とショートカットの完全なリストについては、AWS Big Data Blog の「Improve productivity by using keyboard shortcuts in Amazon Athena query editor」を参照してください。

Athena クエリエディタでは、クエリの作成を高速化するための先行入力コード候補がサポートされています。SQL クエリをより正確に、より効率的に記述できるように、次の機能が提供されます。

- 入力すると、キーワード、ローカル変数、スニペット、およびカタログ項目の候補がリアルタイムで表示されます。
- データベース名またはテーブル名の後にドットを入力すると、エディタはテーブルまたは列のリストを表示して選択できるため便利です。
- スニペットの候補にカーソルを合わせると、概要にスニペットの構文と使用法の概要が表示されます。
- コードの読みやすさを向上させるため、キーワードとその強調表示ルールも Trino と Hive の最新構文に合わせて更新されました。

この機能は、デフォルトでご利用になれます。この機能を有効または無効にするには、クエリエディタウィンドウの右下にある [コードエディタの環境設定] (歯車アイコン) を使用します。

他のデータソースへの接続

このチュートリアルでは、Amazon S3 にある CSV 形式のデータソースを使用しました。AWS Glue での Athena の使用については、「AWS Glue を使用した Amazon S3 内のデータソースへの接続」を参照してください。ODBC ドライバー、JDBC ドライバー、外部の Hive メタストア、および Athena データソースコネクタを使用して、Athena をさまざまなデータソースに接続することもできます。詳細については、「データソースへの接続」を参照してください。

データソースへの接続

Amazon Athena を使用して、データセット内のさまざまな場所にさまざまな形式で保存されているデータをクエリすることができます。このデータセットは、CSV、JSON、Avro、Parquet、またはその他の形式である可能性があります。

クエリを実行するために Athena で使用するテーブルとデータベースは、メタデータに基づいています。メタデータは、データセット内の基になるデータに関するデータです。このメタデータがデータセットを記述する方法をスキーマと呼びます。たとえば、テーブル名、テーブル内の列名、各列のデータ型はメタデータとして保存され、基になるデータセットを記述するスキーマです。Athena では、メタデータを整理するためのシステムを、データカタログまたはメタストアと呼んでいます。データセットとそれを記述するデータカタログの組み合わせをデータソースと呼びます。

メタデータと基になるデータセットとの関係は、使用するデータソースの種類によって異なります。MySQL、PostgreSQL、SQL Server などのリレーショナルデータソースは、メタデータ

をデータセットと緊密に統合します。多くの場合これらのシステムでは、データが書き込まれるときにメタデータが書き込まれます。Hive を使用して構築されたものなどの他のデータソースでは、データセットを読み込むときにその場でメタデータを定義できます。データセットは、CSV、JSON、Parquet、または Avro など、さまざまな形式にすることができます。

Athena は AWS Glue Data Catalog をネイティブにサポートします。AWS Glue Data Catalog は、Amazon S3、Amazon Redshift、Amazon DynamoDB などの他のデータセットとデータソースの上に構築されたデータカタログです。Athena は、さまざまなコネクタを使用して他のデータソースに接続することも可能です。

トピック

AWS Glue との統合

AWS Glue は、フルマネージド型の ETL (抽出、変換、ロード) の AWS のサービス です。データの分析と分類は、その主要機能の 1 つです。AWS Glue クローラは、Amazon S3 のデータからデータベースとテーブルのスキーマを自動的に推論し、関連するメタデータを AWS Glue Data Catalog に保存します。

Athena は、Amazon Web Services アカウントの Amazon S3 データに関するテーブルメタデータの保存と取得に AWS Glue Data Catalog を使用します。テーブルメタデータは、Athena クエリエンジンがクエリするデータの検索、読み込み、および処理方法を把握できるようにします。

AWS Glue Data Catalog でデータベースとテーブルのスキーマを作成するには、データソースに対して Athena 内から AWS Glue クローラを実行する、または Athena クエリエディタでデータ定義言語 (DDL) クエリを直接実行することができます。その後、作成したデータベースとテーブルのスキーマを使用して、データをクエリするために Athena でデータ操作言語 (DML) クエリを使用することができます。

AWS Glue Data Catalog は、所有するアカウント以外のアカウントから登録できます。AWS Glue に必要な IAM アクセス許可を設定したら、Athena を使用してクロスアカウントクエリを実行できま

す。詳細については、「AWS Glue データカタログへのクロスアカウントアクセス」を参照してください。

AWS Glue Data Catalog に関する詳細については、「AWS Glue デベロッパーガイド」の「AWS Glue のデータカタログおよびクローラー」を参照してください。

AWS Glue には別料金が適用されます。詳細については、「AWS Glue 料金表」を参照してください。

トピック

- AWS Glue を使用した Amazon S3 内のデータソースへの接続
- 別のアカウントからの AWS Glue Data Catalog の登録
- Athena で AWS Glue を使用するときのベストプラクティス
- AWS CLI を使用して AWS Glue データベースとそのテーブルを再度作成する

AWS Glue を使用した Amazon S3 内のデータソースへの接続

Athena は、テーブルや列の名前などのメタデータを保存するために、AWS Glue Data Catalog を使用して Amazon S3 に保存されたデータに接続できます。接続が確立されると、データベース、テーブル、およびビューが Athena のクエリエディタに表示されます。

AWS Glue が使用するスキーマ情報を定義するには、情報を自動的に取得する AWS Glue クローラを作成するか、手動でテーブルを追加してそこにスキーマ情報を入力します。

AWS Glue クローラの作成

クローラの作成には、まず Athena コンソールを起動し、それと合わせて AWS Glue コンソールを使用します。クローラーを作成する際には、Amazon S3 内にあるクロール対象のデータの場所を指定します。

Athena コンソールから AWS Glue のクローラを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. クエリエディタの [Tables and views] (テーブルとビュー) の横にある [Create] (作成) を選択し、その後 [AWS Glue crawler] を選択します。
3. AWS Glue コンソールの [Add crawler (クローラの追加)] ページで、次の手順に従いクローラを作成します。詳細については、このガイドの「AWS Glue クローラの使用」、ならびに「AWS Glue デベロッパーガイド」の「AWS Glue Data Catalog の入力」を参照してください。

> **Note**
>
> Athena は、AWS Glue クローラに指定した除外パターンを認識しません。例えば、.csv と .json ファイルの両方が含まれる Amazon S3 バケットがある場合、.json ファイルをクローラから除外しても、Athena は両方のファイルのグループをクエリします。これを回避するには、除外するファイルを別の場所に配置します。

フォームを使用したテーブルの追加

以下の手順で、テーブルの追加のために、Athena コンソールの [Create Table From S3 bucket data] (S3 バケットデータからテーブルを作成) フォームを使用する方法を説明します。

フォームによりテーブルを追加してスキーマ情報を入力するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. クエリエディタで、[Tables and views] (テーブルとビュー) の横にある [Create] (作成) をクリックし、次に [S3 bucket data] (S3 バケットデータ) をクリックします。
3. [Create Table From S3 bucket data] (S3 バケットデータからテーブルを作成) フォームで、[Table name] (テーブル名) にテーブルの名前を入力します。
4. [Database configuration] (データベース設定) で既存のデータベースを選択するか、新しいデータベースを作成します。
5. [Location of Input Data Set] (入力データセットの場所) に、処理するデータセットが含まれるフォルダへの Amazon S3 のパスを指定します。パスにはファイル名を含めないでください。Athena で、指定したフォルダ内のすべてのファイルがスキャンされます。データが既に分割されている場合 (例:

   s3://DOC-EXAMPLE-BUCKET/logs/year =2004/month=12/day=11/)、ベースパスのみを入力します (例: s3://DOC-EXAMPLE-BUCKET/logs/)。
6. [Data Format] (データ形式) で、次のオプションから選択します。

   - [Table type] (テーブルタイプ) には、[Apache Hive] (アパッチハイブ)、[Apache Iceberg] (アパッチアイスバーグ)、または [Delta Lake] (デルタレイク) を選択してください。Athena では Apache Hive テーブルタイプがデフォルトとして使用されます。Athena での Apache Iceberg テーブルのクエリについては、「Apache Iceberg テーブルの使用」を参照してください。Athena での Delta Lake テーブルの使用に関する情報は、「Linux Foundation Delta Lake テーブルへのクエリ」を参照してください。

- [File format] (ファイル形式) で、データに適用されているファイル形式またはログ形式を選択します。
  - [Text File with Custom Delimiters (カスタム区切り記号のあるテキストファイル)] オプションでは、[Field terminator (フィールドターミネータ)] (列区切り記号) を指定します。オプションで、配列型の終わりを示す [Collection terminator] (コレクションターミネータ) またはマッピングデータタイプの終わりを示す [Collection terminator] (コレクションターミネータ) を指定できます。
- [SerDe library] (SerDe ライブラリ) – SerDe (シリアライザー/デシリアライザー) ライブラリは特定のデータ形式を解析して、Athena でそのデータ形式のテーブルを作成できるようにします。ほとんどの形式では、デフォルトの SerDe ライブラリが自動的に選択されます。次の形式については、要件に応じてライブラリを選択してください。
  - [Apache Web Logs] (Apache ウェブログ) – [RegexSerDe] または [GrokSerDe] ライブラリのどちらかを選択します。RegexSerDe の場合は、[Regex definition] (Regex の定義) ボックスに正規表現を入力します。GrokSerDe の場合は、`input.format` SerDe プロパティに一連の名前付き正規表現を入力します。名前付き正規表現は、正規表現よりも読みやすく、管理しやすいです。詳細については、「Simple Storage Service (Amazon S3) に保存されている Apache ログのクエリ」を参照してください。
  - [CSV] – カンマ区切り (CSV) データに二重引用符で囲まれた値が含まれていない場合、または `java.sql.Timestamp` 形式を使用している場合は、[LazySimpleSerDe] を選択します。引用符で囲まれた部分がデータに含まれている場合や、`TIMESTAMP` に UNIX の数値形式 (例: `1564610311`) を使用している場合は、[OpenCSVSerDe] を選択します。詳細については、CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDeおよびCSV を処理するための OpenCSVSerDeを参照してください。
  - [JSON] – [OpenX] または [Hive] JSON SerDe ライブラリのどちらかを選択します。どちらの形式でも、各 JSON ドキュメントが、1 行のテキストに存在し、フィールドが改行文字で区切られていないことが想定されます。OpenX SerDe にはいくつかの追加プロパティがあります。これらのプロパティの詳細については、「OpenX JSON SerDe」を参照してください。Hive SerDe については、「Hive JSON SerDe」を参照してください。

  Athena で SerDe ライブラリを使用する方法の詳細については、「サポートされる SerDes とデータ形式」を参照してください。

7. [SerDe properties] (SerDe プロパティ) で、使用している SerDe ライブラリと要件に従って、プロパティと値を追加、編集、または削除します。
   - SerDe プロパティを追加するには、[Add SerDe property] (SerDe プロパティを追加) を選択します。

- [Name] (名前) フィールドに、プロパティの名前を入力します。
- [Value] (値) フィールドに、プロパティの値を入力します。
- SerDe プロパティを削除するには、[Remove] (削除) を選択します。

8. [Table properties] (テーブルプロパティ) で、要件に従ってテーブルプロパティを選択または編集します。

   - [Write compression] (書き込み圧縮) で、圧縮オプションを選択します。書き込み圧縮オプションが使用可能かどうかと、使用可能な圧縮オプションは、データ形式によって異なります。詳細については、「Athena での圧縮のサポート」を参照してください。
   - [Encryption] (暗号化) で、元のデータが Amazon S3 で暗号化されていれば [Encrypted data set] (暗号化されたデータセット) を選択します。このオプションで、`CREATE TABLE` ステートメント内の `has_encrypted_data` テーブルプロパティが true に設定されます。

9. [Column details] (列の詳細) に、テーブルに追加する列の名前とデータ型を入力します。

   - 列を 1 つずつ追加するには、[Add a column (列の追加)] を選択します。
   - すばやく列を追加するには、[Bulk add columns (列を一括追加)] を選択します。テキストボックスに、列のカンマ区切りのリストを *`column_name`*`data_type`, *`column_name`*`data_type`[, ...], の形式で入力し、[Add] (追加) を選択します。

10. (オプション) [Partition details] (パーティションの詳細) に、1 つ以上の列名とデータ型をそれぞれ追加します。パーティションでは、関連するデータが列の値に基づいてまとめられるため、クエリごとにスキャンされるデータ量を減らすことができます。パーティショニングについては、「Athena でのデータのパーティション化」を参照してください。

11. (オプション) [Bucketing] (バケット化) で、グループ化する行を含む 1 つまたは複数の列を指定し、それらの行を複数のバケットに入れることができます。これにより、バケット化された列の値が指定されている場合に、読み込むバケットのみをクエリできます。

    - [Buckets] (バケット) で、一意の値 (プライマリキーなど) が多く、クエリ内のデータのフィルタリングによく使用される列を 1 つまたは複数選択します。
    - [Number of buckets] (バケット数) に、ファイルを最適なサイズにすることができる数値を入力します。詳細については、AWS Big Data ブログの「Amazon Athena のパフォーマンスチューニング Tips トップ 10」を参照してください。
    - バケット列を指定するには、`CREATE TABLE` ステートメントで次の構文を使用します。

```
CLUSTERED BY (bucketed_columns) INTO number_of_buckets BUCKETS
```

> **Note**
>
> [Bucketing] (バケット化) オプションは、[Iceberg] テーブルタイプでは使用できません。

12. [Preview table query] (テーブルクエリのプレビュー) ボックスに、フォームで入力した情報を基に生成された `CREATE TABLE` ステートメントが表示されます。プレビューでのステートメントを、直接編集することはできません。ステートメントを変更するには、プレビューの上にあるフォームのフィールドを変更します。あるいは、フォームを使用する代わりに、クエリエディタでステートメントを直接作成します。
13. [Create table] (テーブルの作成) をクリックして、そこで生成されたステートメントをクエリエディタ内で実行し、テーブルを作成します。

別のアカウントからの AWS Glue Data Catalog の登録

Athena のクロスアカウント AWS Glue カタログ機能を使用して、所有するアカウント以外のアカウントから AWS Glue カタログを登録できます。AWS Glue に必要な IAM アクセス許可を設定して、カタログを Athena `DataCatalog` リソースとして登録したら、Athena を使用してクロスアカウントクエリを実行できます。必要な許可の設定については、「AWS Glue データカタログへのクロスアカウントアクセス」を参照してください。

以下の手順では、Athena コンソールを使用して、所有する Amazon Web Services アカウント以外のアカウントの AWS Glue Data Catalog をデータソースとして設定する方法を説明します。

別のアカウントから AWS Glue Data Catalog を登録するには

1. 「AWS Glue データカタログへのクロスアカウントアクセス」のステップに従って、他のアカウントのデータカタログをクエリする許可があることを確認します。
2. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
3. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

4. [Data sources] (データソース) を選択します。
5. コンソールの右上で、[Create data source] (データソースの作成) を選択します。
6. [Choose a data source] (データソースを選択) ページの [Data sources] (データソース) で、[S3 - AWS Glue Data Catalog] (S3 – AWS Glue Data Catalog データカタログ) を選択してから、[Next] (次へ) を選択します。
7. [Enter data source details] (データソースの詳細を入力) ページの [AWS Glue Data Catalog] (AWS Glue Data Catalog データカタログ) セクションにある [Choose an AWS Glue Data Catalog] (AWS Glue Data Catalog データカタログを選択) で、[AWS Glue Data Catalog in another account] (別のアカウントの AWS Glue Data Catalog データカタログ) を選択します。
8. [Data source details] (データソースの詳細) に、以下の情報を入力します。
   - [Data source name] (データソース名) – 他のアカウントにあるデータカタログを参照するために SQL クエリで使用する名前を入力します。
   - Description (説明) – (オプション) 他のアカウントにあるデータカタログの説明を入力します。
   - カタログ ID – データカタログが属する Amazon Web Services アカウントの 12 桁のアカウント ID を入力します。Amazon Web Services アカウント ID はカタログ ID です。
9. (オプション) [Tags] (タグ) では、データソースと関連付けるキーと値のペアを入力します。タグの詳細については、Athena リソースへのタグ付けを参照してください。
10. [Next] を選択します。
11. [Review and create] (確認と作成) ページで入力した情報を確認してから、[Create data source] (データソースの作成) を選択します。[Data source details] (データソースの詳細) ページに、登録したデータカタログのデータベースとタグがリストされます。

12. [Data sources] (データソース) を選択します。登録したデータカタログは、[Data source name] (データソース名) 列にリストされています。
13. データカタログに関する情報を表示または編集するには、カタログを選択してから、[Actions] (アクション)、[Edit] (編集) の順に選択します。
14. 新しいデータカタログを削除するには、カタログを選択してから、[Actions] (アクション)、[Delete] (削除する) の順に選択します。

詳細については、AWS Big Data Blog の「Query cross-account AWS Glue Data Catalogs using Amazon Athena」を参照してください。

Athena で AWS Glue を使用するときのベストプラクティス

Athena で AWS Glue Data Catalog を使用するときは、AWS Glue を使用して Athena でクエリされるデータベースとテーブル (スキーマ) を作成するか、Athena を使用してスキーマを作成してから AWS Glue や関連サービスでそれらを使用することができます。このトピックでは、各方法を使用する際の考慮事項とベストプラクティスを示します。

Athena はその内部で、DML ステートメントの処理に Trino を使用し、スキーマを作成および変更する DDL ステートメントの処理には Hive を使用します。これらのテクノロジーを使用する場合、いくつかの規則に従うと、Athena と AWS Glue を効率よく連携できます。

このトピックの内容

- Athena における ETL 用の AWS Glue ジョブの使用
  - Athena を使用した AWS Glue ETL ジョブ用のテーブルの作成
  - ETL ジョブを使用したクエリパフォーマンスの最適化
  - ORC への変換に伴う SMALLINT データ型と TINYINT データ型の INT への変換
  - ETL 用の AWS Glue ジョブの自動化

データベース、テーブル、および列の名前

AWS Glue で作成したスキーマを Athena でクエリする場合は、以下の点を考慮します。

- AWS Glue 内のデータベース名、テーブル名、および列名で使用できる文字は、UTF-8 文字列である必要があります。文字列は、1 バイト以上、255 バイト未満の長さにする必要があります。使用できる文字にはスペースが含まれ、以下にある単一行の文字列パターンで定義されます。

  ```
  [\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\t]*
  ```

- 現在、AWS Glue regex パターンでは、名前の先頭にスペースを追加することができます。このような先頭のスペースは検出が難しく、作成後にユーザビリティ問題を引き起こす可能性があるため、先頭にスペースがあるオブジェクト名は作成しないようにしてください。
- AWS Glue データベースを作成するために、AWS::Glue::Database の AWS CloudFormation テンプレートを、データベース名を指定せずに使用した場合は、AWS Glue が自動的にデータベース名を生成します。この形式は *resource_name-random_string* となり、Athena との互換性がなくなります。
- AWS Glue Catalog Manager では、列の名前を変更できますが、テーブル名やデータベース名は変更できません。この制限を回避するには、古いデータベースの定義を使用して、新しい名前のデータベースを作成する必要があります。次に、古いデータベースのテーブルの定義を使用して、新しいデータベースにテーブルを再度作成します。これには、AWS CLI または AWS Glue SDK を使用できます。この手順については、「AWS CLI を使用して AWS Glue データベースとそのテーブルを再度作成する」を参照してください。

AWS Glue 内のデータベースとテーブルの詳細については、「AWS Glue デベロッパーガイド」の「データベース」と「テーブル」を参照してください。

AWS Glue クローラの使用

AWS Glue クローラを使用すると、データセットのスキーマを検出して AWS Glue Data Catalog に登録できます。クローラは、データを参照しスキーマの判定を行います。さらに、パーティションを

検出して登録することもできます。詳細については、「AWS Glue デベロッパーガイド」の「クローラの定義」を参照してください。正常にクロールされたデータのテーブルは Athena からクエリできます。

> **Note**
>
> Athena は、AWS Glue クローラに指定した除外パターンを認識しません。例えば、`.csv` と `.json` ファイルの両方が含まれる Amazon S3 バケットがある場合、`.json` ファイルをクローラから除外しても、Athena は両方のファイルのグループをクエリします。これを回避するには、除外するファイルを別の場所に配置します。

AWS Glue Data Catalog と Amazon S3 を同期させるためのクローラのスケジュール設定

AWS Glue クローラは、スケジュールに従って実行するか、オンデマンドで実行するように設定できます。詳細については、「AWS Glue デベロッパーガイド」の「ジョブとクローラの時間ベースのスケジュール」を参照してください。

パーティションテーブルのデータが定時に着信する場合は、スケジュールに従って実行するように AWS Glue クローラを設定し、テーブルのパーティションを検出して更新できます。これにより、時間と費用がかかる可能性がある `MSCK REPAIR` コマンドを実行したり、`ALTER TABLE ADD PARTITION` コマンドの手動で実行したりする必要がなくなります。詳細については、「AWS Glue デベロッパーガイド」の「テーブルパーティション」を参照してください。

クローラでの複数のデータソースの使用

AWS Glue クローラが Amazon S3 をスキャンして複数のディレクトリを検出すると、ヒューリスティックを使用してテーブルのルートがディレクトリ構造内のどこにあり、どのディレクトリがテーブルのパーティションであるかを判断します。複数のディレクトリで同様のスキーマが検出されると、クローラは、これらを個別のテーブルではなくパーティションとみなす場合があります。クローラで個別のテーブルを検出しやすくするには、1 つの方法として各テーブルのルートディレクトリをクローラのデータストアとして追加します。

以下の Amazon S3 のパーティションは、その一例です。

```
s3://DOC-EXAMPLE-BUCKET/folder1/table1/partition1/file.txt
s3://DOC-EXAMPLE-BUCKET/folder1/table1/partition2/file.txt
s3://DOC-EXAMPLE-BUCKET/folder1/table1/partition3/file.txt
s3://DOC-EXAMPLE-BUCKET/folder1/table2/partition4/file.txt
```

```
s3://DOC-EXAMPLE-BUCKET/folder1/table2/partition5/file.txt
```

table1 と table2 のスキーマが類似し、AWS Glue のデータソースが s3://DOC-EXAMPLE-BUCKET/folder1/ に対して 1 つのみ設定されている場合、クローラは 1 つのテーブルを 2 つのパーティション列で作成することがあります。1 つのパーティション列に table1 と table2 が入り、別のパーティション列に partition1〜partition5 が入ります。

AWS Glue クローラで 2 つのテーブルを別個に作成するには、クローラに 2 つのデータソース (s3://DOC-EXAMPLE-BUCKET/folder1/table1/ と s3://DOC-EXAMPLE-BUCKET/folder1/table2) を設定します。以下に手順を示します。

AWS Glue で既存のクローラーに S3 データストアを追加するには

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[Crawlers (クローラ)] を選択します。
3. クローラーへのリンクを選択してから、[編集] を選択します。
4. [ステップ 2: データソースと分類子を選択する] で、[編集] を選択します。
5. [Data sources] (データソース) で、[Add a data source] (データソースを追加) を選択します。
6. [Add data source] (データソースの追加) ダイアログボックスの [S3 path] (S3 パス) で、[Browse] (ブラウズ) を選択します。
7. 使用したいバケットを選択し、[Choose] (選択) を選択します。

   追加したデータソースは、[Data sources] (データソース) リストに表示されます。
8. [Next] を選択します。
9. [Configure security settings] (セキュリティ設定の構成) ページで、クローラーの IAM ロールを選択してから、[Next] (次へ) を選択します。
10. S3 パスの末尾がスラッシュで終わっていることを確認し、[Add an S3 data source] (S3 データソースの追加) を選択します。
11. [Set output and scheduling] (出力とスケジュールの設定) ページの、[Output configuration] (出力の設定) で、ターゲットデータベースを選択します。
12. [Next] を選択します。
13. [Review and update] (確認と更新) ページで、選択した内容を確認します。ステップを編集するには、[Edit] (編集) を選択します。

14[Update] (更新) を選択します。

"HIVE_PARTITION_SCHEMA_MISMATCH" を避けるためのパーティションスキーマの同期

AWS Glue データカタログでは、パーティション列があるテーブルごとにテーブルレベルでスキーマが保存され、さらにテーブル内のパーティション別にスキーマが保存されます。パーティションのスキーマは、AWS Glue クローラがパーティション内で読み取ったデータのサンプルに基づいて作成されます。詳細については、「クローラでの複数のデータソースの使用」を参照してください。

Athena がクエリを実行するときは、テーブルのスキーマと、クエリに必要なパーティションのスキーマを検証します。この検証では、列のデータ型を順に比較し、重複する列のデータ型が一致することを確認します。これにより、テーブルの中間で列が追加/削除されるなどの予期しないオペレーションが防止されます。パーティションのスキーマがテーブルのスキーマと異なることを Athena が検知した場合、Athena はクエリを処理できない可能性があり、`HIVE_PARTITION_SCHEMA_MISMATCH` で失敗します。

この問題を解決するいくつかの方法があります。まず、データを誤って追加した場合は、スキーマの差異を生じたデータファイルを削除し、パーティションを削除して、データを最クロールできます。2 つ目は、個々のパーティションをドロップしてから Athena で `MSCK REPAIR` を実行して、テーブルのスキーマを使用してパーティションを再度作成することです。この 2 番目のオプションは、適用するスキーマで引き続きデータを正しく読み取れることが確かな場合にのみ使用します。

テーブルメタデータの更新

AWS Glue クローラは、クロール後に、Apache Hive、Presto、Spark などの他の外部テクノロジーに準拠するために特定のテーブルメタデータを自動的に割り当てます。時折、このクローラが割り当てるメタデータプロパティが正しくないことがあります。正しくないプロパティは、Athena でテーブルをクエリする前に、AWS Glue で手動で修正してください。詳細については、「AWS Glue デベロッパーガイド」の「テーブルの詳細の表示と編集」を参照してください。

CSV ファイルの各データフィールドが引用符で囲まれている場合、AWS Glue は `serializationLib` プロパティを誤解してメタデータの割り当てを間違える場合があります。詳細については、「引用符で囲まれた CSV データ」を参照してください。

CSV ファイルの使用

CSV ファイルで各列向けのデータ値が引用符で囲まれていたり、CSV ファイルに分析対象のデータではないヘッダー値が含まれていたりする場合があります。AWS Glue でこれらのファイルからスキーマを作成する場合は、このセクションのガイダンスに従ってください。

引用符で囲まれた CSV データ

以下の例にあるように、データフィールドが二重引用符で囲まれた CSV ファイルがある場合があります。

```
"John","Doe","123-555-1231","John said \"hello\""
"Jane","Doe","123-555-9876","Jane said \"hello\""
```

引用符で囲まれた値を持つ CSV ファイルから作成されたテーブルに対して Athena でクエリを実行するには、AWS Glue で、OpenCSVSerDe を使用するようにテーブルプロパティを変更する必要があります。OpenCSV SerDe の詳細については、「CSV を処理するための OpenCSVSerDe」を参照してください。

AWS Glue コンソールでテーブルプロパティを編集するには

1. AWS Glue コンソールのナビゲーションペインで、[テーブル] を選択します。
2. 編集するテーブルのリンクを選択した後、[Action] (アクション)、[Edit table] (テーブルの編集) の順に選択します。
3. [Edit table] (テーブルを編集) ページで、以下の変更を行います。
   - [Serialization lib] (シリアル化ライブラリ) には `org.apache.hadoop.hive.serde2.OpenCSVSerde` を入力します。
   - [Serde parameters] (Serde パラメータ) には、`escapeChar`、`quoteChar`、および `separatorChar` の各キーに以下の値を入力します。
     - `escapeChar` には、バックスラッシュ (****) を入力します。
     - `quoteChar` には、二重引用符 (**"**) を入力します。
     - `separatorChar` には、カンマ (**,**) を入力します。
4. [Save] を選択します。

詳細については、「AWS Glue デベロッパーガイド」の「テーブルの詳細の表示と編集」を参照してください。

AWS Glue テーブルプロパティのプログラム的な更新

以下の JSON 例にあるように、AWS Glue の UpdateTable API オペレーション、または update-table CLI コマンドを使用して、テーブル定義内の `SerDeInfo` ブロックを変更することができます。

```
"SerDeInfo": {
   "name": "",
   "serializationLib": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
   "parameters": {
      "separatorChar": ","
      "quoteChar": "\""
      "escapeChar": "\\"
      }
},
```

ヘッダー付きの CSV ファイル

CREATE TABLE ステートメントを使用して Athena でテーブルを定義するときは、以下の例にあるように、skip.header.line.count テーブルプロパティを使用して CSV データ内のヘッダーを無視することができます。

```
...
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/csvdata_folder/';
TBLPROPERTIES ("skip.header.line.count"="1")
```

または、CSV のヘッダーを事前に削除して、ヘッダー情報が Athena のクエリ結果に含まれないようにすることもできます。そのためには、1 つの方法として AWS Glue ジョブを使用して ETL (抽出、変換、ロード) スクリプトを実行します。必要なスクリプトは、AWS Glue で PySpark Python ダイアレクトの拡張言語を使用して作成できます。詳細については、AWS Glue デベロッパーガイドの「AWS Glue でのジョブの作成」を参照してください。

次の例に示す AWS Glue スクリプトの関数では、from_options を使用して動的フレームを作成し、writeHeader フォーマットオプションを false に設定することで、ヘッダー情報を削除しています。

```
glueContext.write_dynamic_frame.from_options(frame = applymapping1, connection_type =
 "s3", connection_options = {"path": "s3://DOC-EXAMPLE-BUCKET/MYTABLEDATA/"}, format =
 "csv", format_options = {"writeHeader": False}, transformation_ctx = "datasink2")
```

AWS Glue パーティションのインデックス作成とフィルタリング

Athena は、パーティション分割されたテーブルに対しクエリを実行する際、クエリに関連するサブセットに使用可能なテーブルパーティションの取得とフィルタリングを行います。新しいデータ

とパーティションが追加されると、パーティションの処理に時間がかかり、クエリの実行時間が長くなる可能性があります。時間の経過とともに多数のテーブルが増加するパーティションの場合は、AWS Glue パーティションのインデックス作成とフィルタリングを使用することを検討してください。パーティションのインデックス作成により、Athena によるパーティション処理が最適化され、高度にパーティション化されたテーブルでのクエリパフォーマンスを向上させることができます。テーブルのプロパティでのパーティションフィルタリングの設定は、以下 2 つのステップで行います。

1. AWS Glue でのパーティションインデックスの作成。
2. テーブルのパーティションフィルタリングの有効化。

パーティションインデックスの作成

AWS Glue でのパーティションインデックス作成のステップについては、「AWS Glue デベロッパーガイド」の「パーティションインデックスの使用」を参照してください。AWS Glue でのパーティションインデックスの制限については、同じページの「パーティションインデックスについて」セクションを参照してください。

パーティションフィルタリングの有効化

テーブルのパーティションフィルタリングを有効にするには、AWS Glue で新しくテーブルのプロパティを設定する必要があります。AWS Glue でテーブルプロパティを設定する方法については、「パーティション射影のセットアップ」ページを参照してください。AWS Glue でテーブルの詳細設定を編集する際、[Table properties] (テーブルのプロパティ) セクションに、以下のキーと値のペアを追加します。

- [Key] (キー) に `partition_filtering.enabled` を追加します。
- [Value] (値) に `true` を追加します。

値 `partition_filtering.enabled` に `false` を設定することで、このテーブルのパーティションフィルタリングをいつでも無効にすることができます。

上記の各ステップを完了したら、Athena コンソールに戻ってデータをクエリできます。

パーティションのインデックス作成のフィルタリングの使用の詳細については、AWS Big Data Blog の「Improve Amazon Athena query performance using AWS Glue Data Catalog partition indexes」を参照してください。

地理空間データの使用

AWS Glue は、Well-known Text (WKT)、Well-Known Binary (WKB)、またはその他の PostGIS データ型をネイティブにサポートしていません。AWS Glue 分類子は地理空間データを解析し、CSV 用の `varchar` など、形式でサポートされているデータ型を使用して分類します。他の AWS Glue テーブルと同様に、Athena がこれらのデータ型をそのまま解析できるよう、地理空間データから作成されたテーブルのプロパティ更新が必要になる場合があります。詳細については、「AWS Glue クローラの使用」および「CSV ファイルの使用」を参照してください。Athena は、AWS Glue テーブルの一部の地理空間データ型をそのまま解析できない場合があります。Athena での地理空間データの操作の詳細については、「地理空間データのクエリ」を参照してください。

Athena における ETL 用の AWS Glue ジョブの使用

AWS Glue ジョブは ETL オペレーションを実行します。AWS Glue ジョブは、ソースからデータを抽出し、そのデータを変換してターゲット内にロードするためのスクリプトを実行します。詳細については、AWS デベロッパーガイドの「AWS Glue Glue でのジョブの作成」を参照してください。

Athena を使用した AWS Glue ETL ジョブ用のテーブルの作成

Athena 内で作成するテーブルには、データの形式を識別する `classification` と呼ばれるテーブルプロパティが追加されている必要があります。これにより、AWS Glue はテーブルを ETL ジョブに使用できます。分類値は `avro`、`csv`、`json`、`orc`、`parquet` または `xml` です。Athena での `CREATE TABLE` ステートメントの例は、以下のとおりです。

```
CREATE EXTERNAL TABLE sampleTable (
  column1 INT,
  column2 INT
  ) STORED AS PARQUET
  TBLPROPERTIES (
  'classification'='parquet')
```

このテーブルプロパティは、テーブルの作成時に追加しなかった場合、AWS Glue コンソールを使用して追加できます。

AWS Glue コンソールを使用して分類テーブルプロパティを追加するには

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. コンソールのナビゲーションペインで、[Tables] (テーブル) を選択します。

3. 編集するテーブルのリンクを選択した後、[Action] (アクション)、[Edit table] (テーブルの編集) の順に選択します。
4. [Table properties] (テーブルプロパティ) セクションまで、下にスクロールします。
5. 追加] を選択します。
6. [Key] (キー) に「**classification**」と入力します。
7. [Value] (値) に、データタイプを入力します (例:**json**)。
8. [Save] を選択します。

   [Table details] (テーブルの詳細) セクションで、入力したデータタイプがテーブルの [Classification] (分類) フィールドに表示されます。

詳細については、「AWS Glue デベロッパーガイド」の「テーブルの使用」を参照してください。

ETL ジョブを使用したクエリパフォーマンスの最適化

AWS Glue ジョブは、Athena でのクエリパフォーマンスが最適化されるような形式にデータを変換するのに役立ちます。データの形式は、Athena でのクエリパフォーマンスとクエリコストに大きな影響を及ぼします。

Parquet 形式と ORC データ形式を使用することをお勧めします。AWS Glue は、両方のデータ形式に対応しており、Athena の最適な形式にデータを簡単かつ高速に変換できます。これらの形式、およびパフォーマンスを改善するためのその他の方法に関する詳細については、「Amazon Athena のパフォーマンスチューニング Tips トップ 10」を参照してください。

ORC への変換に伴う SMALLINT データ型と TINYINT データ型の INT への変換

AWS Glue ETL ジョブで生成される `SMALLINT` データ型と `TINYINT` データ型を Athena で読み取れない問題を減らすために、ウィザードの使用時または ETL ジョブ用のスクリプトの作成時に `SMALLINT` と `TINYINT` を `INT` に変換します。

ETL 用の AWS Glue ジョブの自動化

AWS Glue ETL ジョブは、トリガーに基づいて自動的に実行するように設定できます。この機能は、AWS 外からのデータが、Athena でのクエリのために最適とは言えない形式で Amazon S3 バケットにプッシュされている場合に適しています。詳細については、「AWS Glue デベロッパーガイド」の「AWS Glue ジョブのトリガー」を参照してください。

AWS CLI を使用して AWS Glue データベースとそのテーブルを再度作成する

AWS Glue データベースの名前を直接変更することはできませんが、その定義をコピーして定義を変更し、その定義を使用してデータベースを別の名前で再度作成することができます。同様に、古いデータベースのテーブルの定義をコピーして、その定義を変更し、変更した定義を使用して新しいデータベースにテーブルを再度作成することができます。

> **Note**
>
> ここで示した方法では、テーブルのパーティショニングはコピーされません。

次の Windows での手順では、お客様の AWS CLI に JSON 出力が設定されていることを前提としています。AWS CLI のデフォルトの出力形式を変更するには、`aws configure` を実行します。

AWS CLI を使用して AWS Glue データベースをコピーする方法

1. コマンドプロンプトで次の AWS CLI コマンドを実行し、コピーする AWS Glue データベースの定義を取得します。

   ```
   aws glue get-database --name database_name
   ```

   `get-database` コマンドの詳細については、「get-database」を参照してください。
2. JSON 出力を、新しいデータベースの名前が付けられたファイル (例: *new_database_name*`.json`) としてデスクトップに保存します。
3. テキストエディタで *new_database_name*`.json` ファイルを開きます。
4. JSON ファイルで以下の手順を実行します。

   a. 外側の `{ "Database":` エントリと、ファイルの最後にある対応する閉じ括弧 `}` を削除します。

   b. `Name` エントリを新しいデータベース名に変更します。

   c. `CatalogId` フィールドを削除します。
5. ファイルを保存します。
6. コマンドプロンプトで次の AWS CLI コマンドを実行し、変更したデータベース定義ファイルを使用して、新しい名前でデータベースを作成します。

   ```
   aws glue create-database --database-input "file://~/Desktop\new_database_name.json"
   ```

create-database コマンドの詳細については、「create-database」を参照してください。ファイルから AWS CLI パラメータを読み込む方法の詳細については、「AWS Command Line Interface ユーザーガイド」の「ファイルから AWS CLI パラメータを読み込む」を参照してください。

7. 新しいデータベースが AWS Glue に作成されたことを確認するには、次のコマンドを実行します。

```
aws glue get-database --name new_database_name
```

これで、新しいデータベースにコピーするテーブルの定義を取得し、定義を変更して、変更した定義を使用して新しいデータベースにテーブルを再度作成する準備ができました。この手順でテーブル名は変更されません。

AWS CLI を使用して AWS Glue テーブルをコピーする方法

1. コマンドプロンプトで、次の AWS CLI コマンドを実行します。

```
aws glue get-table --database-name database_name --name table_name
```

get-table コマンドの詳細については、「get-table」を参照してください。

2. JSON 出力を、テーブルの名前が付けられたファイル (例: *table_name*.json) として Windows デスクトップに保存します。
3. テキストエディタで ファイルを開きます。
4. JSON ファイルで、外部 {"Table": エントリとファイルの最後にあるそれに対応する閉じ括弧 } を削除します。
5. JSON ファイルで、次のエントリとその値を削除します。

   - DatabaseName — create-table CLI コマンドは --database-name パラメータを使用するため、このエントリは必要ありません。
   - CreateTime
   - UpdateTime
   - CreatedBy
   - IsRegisteredWithLakeFormation
   - CatalogId

- VersionId

6. テーブル定義ファイルを保存します。
7. コマンドプロンプトで次の AWS CLI コマンドを実行し、新しいデータベースにテーブルを再度作成します。

```
aws glue create-table --database-name new_database_name --table-input "file://~/
Desktop\table_name.json"
```

create-table コマンドの詳細については、「create-table」を参照してください。

これで、テーブルが AWS Glue の新しいデータベースに表示され、Athena からクエリできるようになりました。

8. 手順を繰り返して、追加する各テーブルを AWS Glue の新しいデータベースにコピーします。

外部 Hive メタストア用の Athena データコネクタの使用

Apache Hive メタストアを使用する Amazon S3 内のデータセットをクエリするには、外部 Hive メタストア用の Amazon Athena データコネクタを使用できます。AWS Glue Data Catalog へのメタデータの移行は不要です。Athena マネジメントコンソールでプライベート VPC 内の Hive メタストアと通信する Lambda 関数を設定してから、それをメタストアに接続します。Lambda から Hive メタストアへの接続はプライベート Amazon VPC チャンネルによって保護されており、パブリックインターネットを使用しません。独自の Lambda 関数コードを提供することも、外部 Hive メタストア用の Athena データコネクタのデフォルト実装を使用することもできます。

トピック

- 機能の概要
- ワークフロー
- 考慮事項と制約事項
- Apache Hive メタストアへの Athena の接続
- AWS Serverless Application Repository を使用した Hive データソースコネクタのデプロイ
- 既存の IAM 実行ロールを使用した Hive メタストアへの Athena の接続
- デプロイ済みの Hive メタストアコネクタを使用するように Athena を設定する
- 外部 Hive メタストアクエリでのデフォルトのデータソース名の使用
- Hive ビューの使用

- Hive メタストアで AWS CLI を使用する
- リファレンス実装

機能の概要

外部 Hive メタストア用の Athena データコネクタを使用することで、以下のタスクを実行できます。

- Athena コンソールを使用して、カスタムカタログを登録し、それらを使用してクエリを実行する。
- 異なる外部 Hive メタストアの Lambda 関数を定義して、Athena クエリでそれらを結合する。
- AWS Glue Data Catalog と外部 Hive メタストアを同じ Athena クエリで使用する。
- クエリ実行コンテキストのカタログを現在のデフォルトカタログとして指定する。これにより、クエリのデータベース名にカタログ名のプレフィックスを付ける必要がなくなります。構文 *`catalog`*`.`*`database`*`.`*`table`* を使用する代わりに、*`database`*`.`*`table`* を使用できます。
- さまざまなツールを使用して、外部 Hive メタストアを参照するクエリを実行する。これには、Athena コンソール、AWS CLI、AWS SDK、Athena API、更新された Athena JDBC および ODBC ドライバーを使用できます。更新されたドライバーは、カスタムカタログをサポートしています。

API サポート

外部 Hive メタデータ用の Athena データコネクタには、カタログ登録 API オペレーションとメタデータ API オペレーションのサポートが含まれています。

- カタログ登録 – 外部 Hive メタストアとフェデレーティッドデータソース用のカスタムカタログを登録します。
- メタデータ – メタデータ API を使用して、AWS Glue と、Athena に登録した任意のカタログのデータベースとテーブル情報を提供します。
- Athena JAVA SDK クライアント – 更新された Athena Java SDK クライアントのカタログ登録 API、メタデータ API、および `StartQueryExecution` オペレーションでのカタログのサポートを使用します。

リファレンス実装

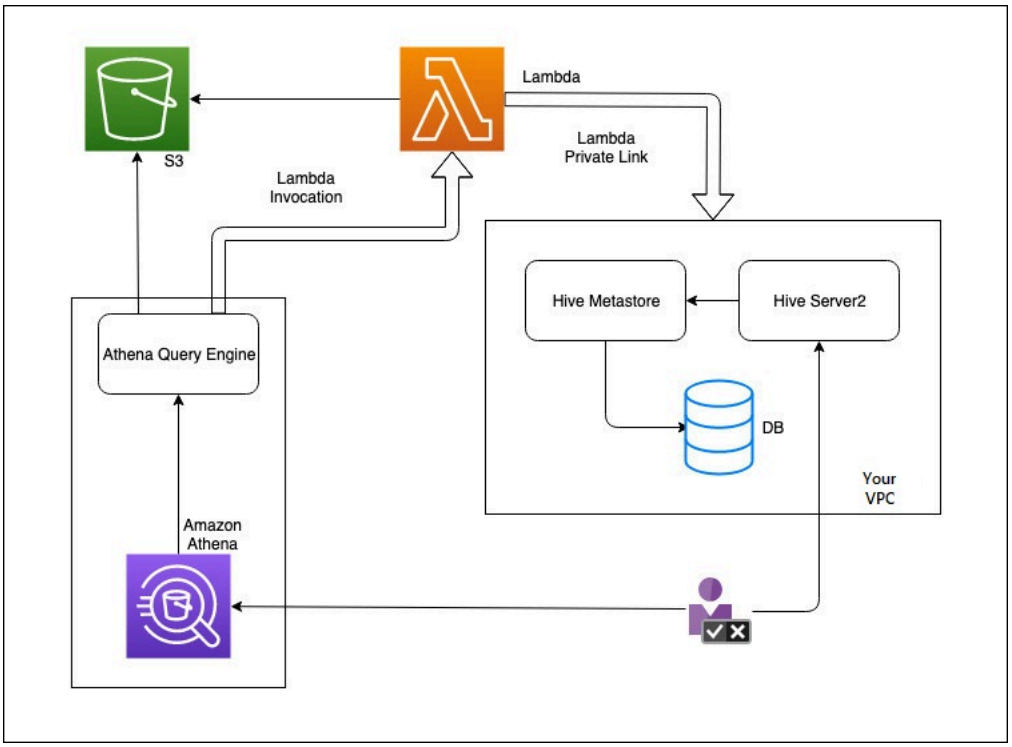
Athena は、外部 Hive メタストアに接続する Lambda 関数のリファレンス実装を提供します。リファレンス実装は、Athena Hive メタストアのオープンソースプロジェクトとして GitHub で提供されています。

リファレンス実装は、AWS SAM (SAR) で次の 2 つの AWS Serverless Application Repository アプリケーションとして利用できます。これらのアプリケーションのいずれかを SAR で使用して、独自の Lambda 関数を作成できます。

- **`AthenaHiveMetastoreFunction`** – Uber Lambda 関数 `.jar` ファイルです。「uber」JAR (fat JAR、または依存関係が含まれる JAR としても知られています) は、Java プログラムとその依存関係の両方を単一のファイルに収めた `.jar` ファイルです。
- **`AthenaHiveMetastoreFunctionWithLayer`** – Lambda レイヤーと thin Lambda 関数 `.jar` ファイルです。

ワークフロー

以下の図は、Athena が外部 Hive メタストアとどのようにやり取りするかを説明しています。



このワークフローでは、データベースに接続された Hive メタストアが VPC 内にあります。Hive Server2 を使用して、Hive CLI を使用して Hive メタストアを管理します。

Athena から外部 Hive メタストアを使用するワークフローには、以下のステップが含まれます。

1. VPC 内の Hive メタストアに Athena を接続する Lambda 関数を作成します。
2. Hive メタストアの一意のカタログ名をアカウントに登録し、対応する関数名を登録します。
3. カタログ名を使用する Athena DML または DDL クエリを実行すると、Athena クエリエンジンがそのカタログ名に関連付けられた Lambda 関数名を呼び出します。
4. AWS PrivateLink を使用すると、Lambda 関数が VPC 内の外部 Hive メタストアと通信し、メタデータリクエストに対する応答を受け取ります。Athena は、デフォルトの AWS Glue Data Catalogからのメタデータを使用する場合と同様に、外部 Hive メタストアからのメタデータを使用します。

考慮事項と制約事項

外部 Hive メタストア用の Athena データコネクタを使用するときは、以下の点を考慮してください。

- CTAS を使用して、外部の Hive メタストアにテーブルを作成できます。
- INSERT INTO を使用して、外部 Hive メタストアにデータを挿入できます。
- 外部 Hive メタストアに対する DDL サポートは、以下のステートメントに限定されています。
  - ALTER DATABASE SET DBPROPERTIES
  - ALTER TABLE ADD COLUMNS
  - ALTER TABLE ADD PARTITION
  - ALTER TABLE DROP PARTITION
  - ALTER TABLE RENAME PARTITION
  - テーブルの変更列の置き換え
  - ALTER TABLE SET LOCATION
  - ALTER TABLE SET TBLPROPERTIES
  - CREATE DATABASE
  - CREATE TABLE
  - CREATE TABLE AS
  - DESCRIBE TABLE
  - DROP DATABASE
  - DROP TABLE

  - SHOW COLUMNS
  - SHOW CREATE TABLE
  - SHOW PARTITIONS
  - SHOW SCHEMAS
  - SHOW TABLES
  - SHOW TBLPROPERTIES
- 登録できるカタログの最大数は 1,000 です。
- Hive メタストアの Kerberos 認証はサポートされていません。
- 外部 Hive メタストア、またはフェデレーティッドクエリで JDBC ドライバーを使用するには、JDBC 接続文字列に `MetadataRetrievalMethod=ProxyAPI` を含めます。JDBC ドライバーの詳細については、「JDBC を使用した Amazon Athena への接続」を参照してください。
- Hive の非表示列 である `$path`、`$bucket`、`$file_size`、`$file_modified_time`、`$partition`、`$row_id` はきめ細かいアクセス制御フィルタリングには使用できません。
- *`example_table`*`$partitions` や *`example_table`*`$properties` などの Hive 非表示システムテーブルでは、きめ細かなアクセス制御はサポートされていません。

アクセス許可

構築済みのデータソースコネクタは、正しく機能するために、次のリソースへのアクセスが必要になる場合があります。使用するコネクタの情報をチェックして、VPC が正しく設定されていることを確認します。Athena でクエリを実行してデータソースコネクタを作成するために必要な IAM 許可については、「外部 Hive メタストア用の Athena データコネクタへのアクセスを許可する」および「外部 Hive メタストアへの Lambda 関数アクセスを許可する」を参照してください。

- Simple Storage Service (Amazon S3) – データコネクタは、Athena のクエリ結果が保存される Amazon S3 内の場所にクエリ結果を書き込むほか、Amazon S3 のスピルバケットにも書き込みます。この Amazon S3 の場所に対する接続と許可が必要です。詳細については、このトピックで後述する「Amazon S3 のスピルの場所」を参照してください。
- Athena – クエリのステータスをチェックして、オーバースキャンを防止するためのアクセス権が必要です。
- AWS Glue – コネクタが補足メタデータまたはプライマリメタデータに AWS Glue を使用する場合、アクセス権が必要です。
- AWS Key Management Service

- ポリシー – Hive メタストア、Athena Query Federation、および UDF には、AWS 管理ポリシー: AmazonAthenaFullAccess 以外にポリシーが必要です。詳細については、「Athena でのアイデンティティとアクセス権の管理」を参照してください。

Amazon S3 のスピルの場所

Lambda 関数のレスポンスサイズに対する制限のため、しきい値を超えるサイズのレスポンスは、Lambda 関数の作成時に指定する Amazon S3 の場所にスピルされます。Athena は、これらのレスポンスを Amazon S3 から直接読み込みます。

> **Note**
>
> Athena は Amazon S3 上のレスポンスファイルを削除しません。レスポンスファイルを自動的に削除するように保持ポリシーをセットアップすることが推奨されます。

Apache Hive メタストアへの Athena の接続

Athena を Apache Hive メタストアに接続するには、Lambda 関数を作成して設定する必要があります。基本的な実装では、Athena マネジメントコンソールから必要なステップのすべてを実行できます。

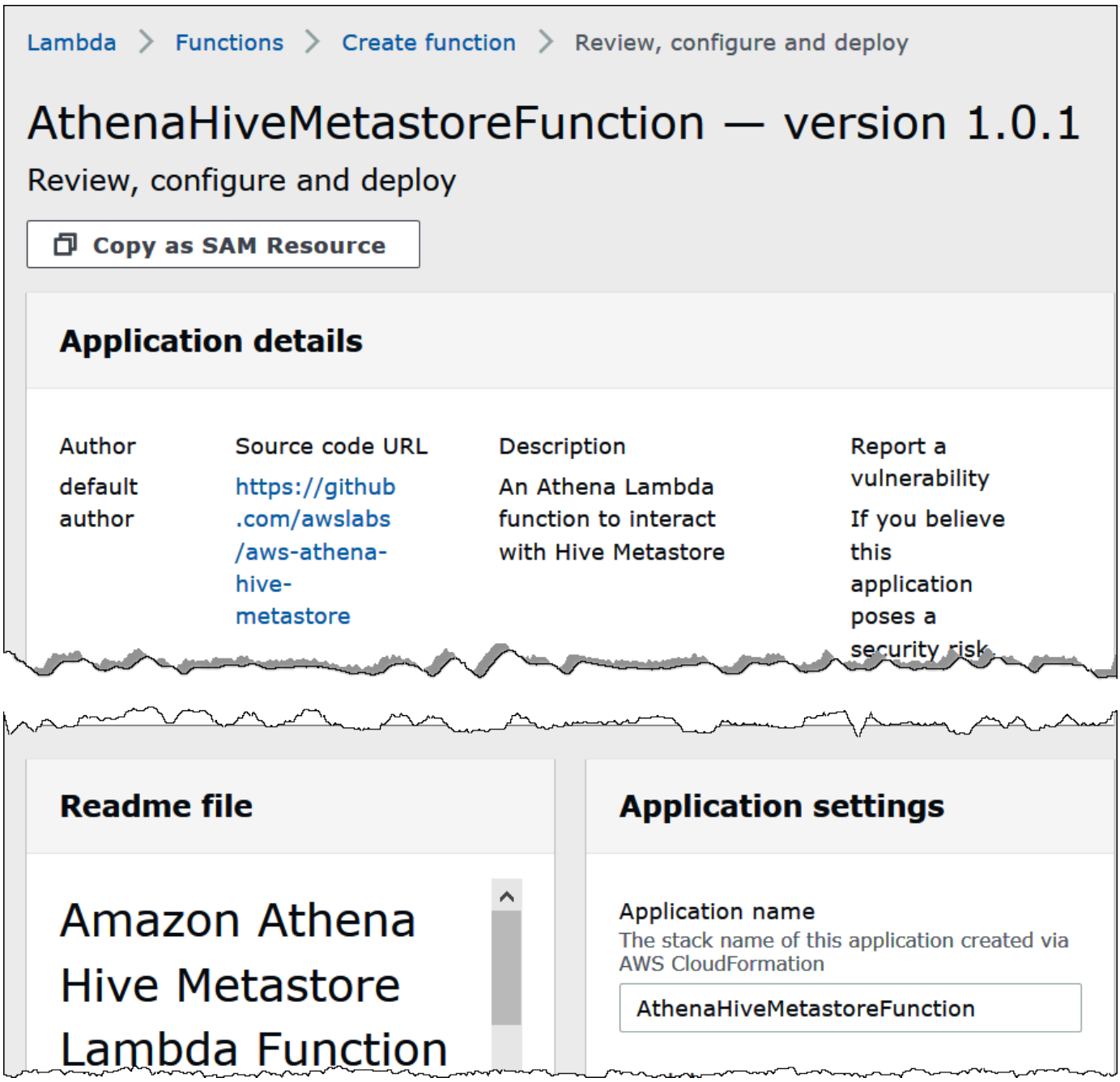
> **Note**
>
> 以下の手順には、Lambda 関数にカスタム IAM ロールを作成するための許可が必要です。カスタムロールを作成するアクセス許可がない場合は、Athena のリファレンス実装を使用して Lambda 関数を個別に作成してから、AWS Lambda コンソールを使用してこの関数に既存の IAM ロールを選択することができます。詳細については、「」を参照してください既存の IAM 実行ロールを使用した Hive メタストアへの Athena の接続

Athena を Hive メタストアに接続する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. [Data sources] (データソース) を選択します。
4. コンソールの右上で、[Create data source] (データソースの作成) を選択します。
5. [Choose a data source] (データソースの選択) ページの [Data sources] (データソース) で、[S3 - Apache Hive metastore] (S3 – Apache Hive メタストア) を選択します。
6. [Next] を選択します。
7. [Data source details] (データソースの詳細) セクションの [Data source name] (データソース名) に、Athena からデータソースをクエリする際に SQL ステートメントで使用する名前を入力します。名前は最大 127 文字で、アカウント内で一意である必要があります。この設定は、作成後に変更することはできません。有効な文字は、a ~ z、A ~ Z、0 ~ 9、_ (アンダースコア)、@ (at マーク)、および - (ハイフン) です。`awsdatacatalog`、`hive`、`jmx`、および `system` の名前は Athena によって予約されており、データソース名には使用できません。
8. [Lambda 関数] で [Lambda 関数の作成] を選択してから、[AWS Lambda で新しい関数を作成] を選択します。

   AWS Lambda コンソールに [AthenaHiveMetastoreFunction] ページが開きます。このページには、コネクタに関する詳細情報が表示されます。

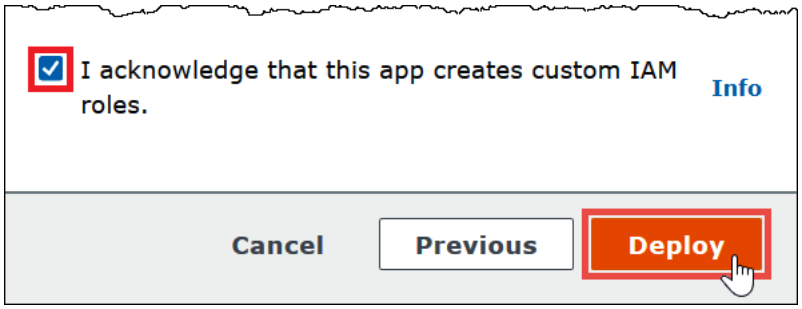


9. [Application settings] (アプリケーションの設定) で Lambda 関数のパラメータを入力します。

   - [LambdaFuncName] – 関数の名前を指定します。たとえば、myHiveMetastore です。
   - [SpillLocation] (スピル場所) - Lambda 関数のレスポンスサイズが 4 MB を超える場合にスピルオーバーメタデータを保持するための、このアカウント内の Amazon S3 の場所を指定します。
   - [HMSUris] – ポート 9083 で Thrift プロトコルを使用する Hive メタストアホストの URI を入力します。`thrift://<host_name>:9083` 構文を使用してください。

- [LambdaMemory] - 128 MB から 3,008 MB までの値を指定します。Lambda 関数には、設定するメモリ量に比例した CPU サイクルが割り当てられます。デフォルトは 1024 です。
- [LambdaTimeout] – Lambda 呼び出しの最大許容実行時間を 1 ~ 900 (900 秒は 15 分) の秒数で指定します。デフォルトは 300 秒 (5 分) です。
- [VPCSecurityGroupIds] – Hive メタストアの VPC セキュリティグループ ID のカンマ区切りリストを入力します。
- [VPCSubnetIds] – Hive メタストアの VPC サブネット ID のカンマ区切りリストを入力します。

10. [I acknowledge that this app creates custom IAM roles and resource policies] (このアプリがカスタム IAM ロールとリソースポリシーを作成することを承認します) を選択した後、[Deploy] (デプロイ) を選択します。



デプロイが完了すると、Lambda アプリケーションのリストにこの関数が表示されます。Hive メタストア関数がアカウントにデプロイされたところで、それを使用するように Athena を設定できます。

11. Athena コンソールの [Connect data sources] (データソースを接続) ページに戻ります。
12. [Lambda function] セクションで、Lambda 関数の検索ボックスの横にある更新アイコンをクリックします。利用できる関数のリストを更新すると、新しく作成した関数がリストに表示されます。
13. Lambda コンソールで作成した関数の名前を選択します。Lambda 関数の ARN が表示されます。
14. (オプション) [Tags] (タグ) でキーと値のペアを追加して、このデータソースに関連付けます。タグの詳細については、Athena リソースへのタグ付けを参照してください。
15. [Next] を選択します。
16. [Review and create] (確認と作成) ページで、データソースの詳細について確認し、[Add data source] (データソースの追加) を選択します。
17. データソースのページの [データソースの詳細] セクションには、新しいコネクタに関する情報が表示されます。

これで、指定した [Data source name] (データソース名) を使用して、Athena の SQL クエリで Hive メタストアを参照できるようになります。SQL クエリで、次の構文例を使用し、hms-catalog-1 を以前に指定したカタログ名に置き換えます。

```
SELECT * FROM hms-catalog-1.CustomerData.customers
```

18. 作成したデータソースを表示、編集、または削除する方法については、「データソースの管理」を参照してください。

AWS Serverless Application Repository を使用した Hive データソースコネクタのデプロイ

Hive 用の Athena データソースコネクタをデプロイする際は、Athena コンソールから始める代わりに AWS Serverless Application Repository を使用できます。AWS Serverless Application Repository により、使用する対象のコネクタを検索し、コネクタに必要なパラメータを指定した上で、そのコネクタをアカウントにデプロイします。コネクタのデプロイが完了したら、Athena コンソールを使用して Athena がデータソースを使用できるようにします。

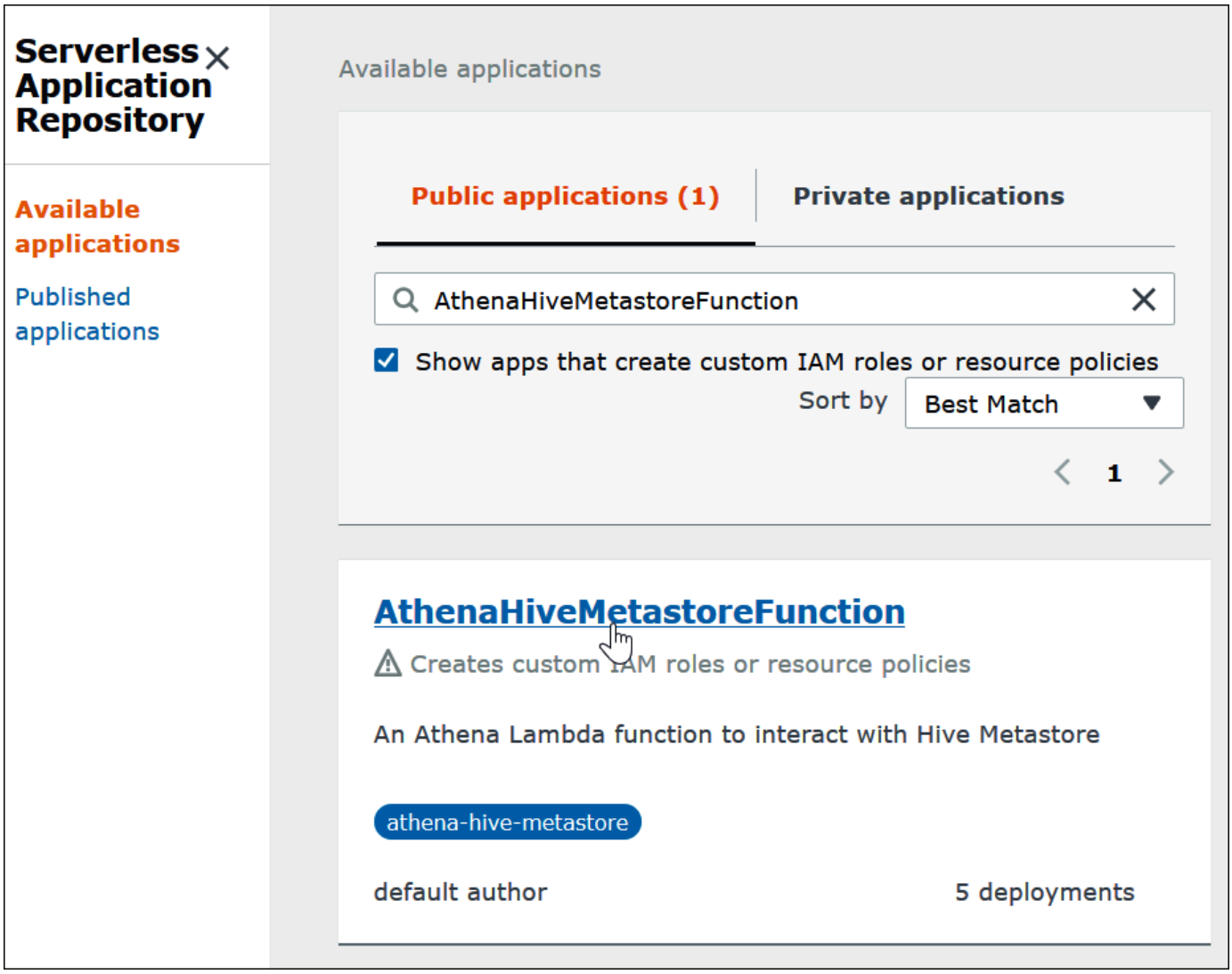
AWS Serverless Application Repository を使用して Hive のデータソースコネクタをアカウントにデプロイするには

1. AWS Management Console にサインインし、サーバーレスアプリケーションリポジトリを開きます。
2. ナビゲーションペインで、[Available applications] (利用可能なアプリケーション) を選択します。
3. [Show apps that create custom roles or resource policies] (カスタム IAM ロールまたはリソースポリシーを作成するアプリを表示する) オプションを選択します。
4. 検索ボックスに「**Hive**」と入力します。表示されるコネクタには、次の 2 つがあります。

   - AthenaHiveMetastoreFunction – Uber Lambda 関数の .jar ファイル。
   - AthenaHiveMetastoreFunctionWithLayer – Lambda レイヤーと thin Lambda 関数の .jar ファイル。

   2 つのアプリケーションの機能は同じで、実装のみが異なります。どちらか 1 つを使用して、Athena を Hive メタストアに接続する Lambda 関数を作成できます。

5. 使用するコネクタの名前を選択します。このチュートリアルでは AthenaHiveMetastoreFunction を使用します。



6. [Application settings] (アプリケーションの設定) で Lambda 関数のパラメータを入力します。

   - [LambdaFuncName] – 関数の名前を指定します。たとえば、myHiveMetastore です。
   - [SpillLocation] (スピル場所) - Lambda 関数のレスポンスサイズが 4 MB を超える場合にスピルオーバーメタデータを保持するための、このアカウント内の Amazon S3 の場所を指定します。
   - [HMSUris] – ポート 9083 で Thrift プロトコルを使用する Hive メタストアホストの URI を入力します。`thrift://<host_name>:9083` 構文を使用してください。
   - [LambdaMemory] - 128 MB から 3,008 MB までの値を指定します。Lambda 関数には、設定するメモリ量に比例した CPU サイクルが割り当てられます。デフォルトは 1024 です。

- [LambdaTimeout] – Lambda 呼び出しの最大許容実行時間を 1 ~ 900 (900 秒は 15 分) の秒数で指定します。デフォルトは 300 秒 (5 分) です。
- [VPCSecurityGroupIds] – Hive メタストアの VPC セキュリティグループ ID のカンマ区切りリストを入力します。
- [VPCSubnetIds] – Hive メタストアの VPC サブネット ID のカンマ区切りリストを入力します。

7. [Application details] (アプリケーションの詳細)ページの右下で [I acknowledge that this app creates custom IAM roles] (このアプリがカスタム IAM ロールを作成することを承認します) を選択してから、[Deploy] (デプロイ) をクリックします。

この時点で、Lambda 関数を使用して Hive メタストアに接続するように Athena を設定できます。この手順については、「デプロイ済みの Hive メタストアコネクタを使用するように Athena を設定する」を参照してください。

既存の IAM 実行ロールを使用した Hive メタストアへの Athena の接続

既存の IAM ロールを使用する Lambda 関数を使用して外部 Hive メタストアを Athena に接続するには、外部 Hive メタストア用の Athena コネクタのリファレンス実装を使用できます。

以下は、3 つの主なステップです。

1. クローンと構築 – Athena リファレンス実装をクローンし、Lambda 関数コードが含まれる JAR ファイルを構築します。
2. AWS Lambda コンソール – AWS Lambda コンソールで Lambda 関数を作成し、それに既存の IAM 実行ロールを割り当てて、生成した関数コードをアップロードします。
3. Amazon Athena コンソール – Amazon Athena コンソールで、Athena クエリで外部 Hive メタストアを参照する際に使用するデータソース名を作成します。

カスタム IAM ロールを作成するアクセス許可がすでにある場合は、Lambda 関数の作成と設定に Athena コンソールと AWS Serverless Application Repository を使用する、よりシンプルなワークフローを使用できます。詳細については、「Apache Hive メタストアへの Athena の接続」を参照してください。

前提条件

- Git がシステムにインストールされている。

- Apache Maven がインストールされている。
- Lambda 関数に割り当てることができる IAM 実行ロールがある。詳細については、「」を参照してください外部 Hive メタストアへの Lambda 関数アクセスを許可する

Lambda 関数をクローンして構築する

Athena リファレンス実装の関数コードは、GitHub (awslabs/aws-athena-hive-metastore) にある Maven プロジェクトです。プロジェクトの詳細については、GitHub の対応する README ファイル、または本書の「リファレンス実装」トピックを参照してください。

Lambda 関数コードをクローンして構築する

1. 以下のコマンドを入力して、Athena リファレンス実装をクローンします。

   ```
   git clone https://github.com/awslabs/aws-athena-hive-metastore
   ```

2. 以下のコマンドを実行して、Lambda 関数の `.jar` ファイルを構築します。

   ```
   mvn clean install
   ```

   プロジェクトが正常に構築されると、以下の `.jar` ファイルがプロジェクトのターゲットフォルダに作成されます。

   ```
   hms-lambda-func-1.0-SNAPSHOT-withdep.jar
   ```

   次のセクションでは、AWS Lambda コンソールを使用してこのファイルを Amazon Web Services アカウントにアップロードします。

AWS Lambda コンソールで Lambda 関数を作成して設定する

このセクションでは、AWS Lambda コンソールを使用して、既存の IAM 実行ロールを使用する関数を作成します。関数用の VPC を設定したら、関数コードをアップロードして、関数の環境変数を設定します。

Lambda 関数を作成する

このステップでは、既存の IAM ロールを使用する関数を AWS Lambda コンソールで作成します。

既存の IAM ロールを使用する Lambda 関数を作成する

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. ナビゲーションペインで、[Functions] (関数) を選択します。
3. [関数の作成] を選択します。
4. Author from scratch (製作者を最初から) を選択します。
5. [Function name] (関数名) には、Lambda 関数の名前 (**`EHMSBasedLambda`** など) を入力します。
6. [Runtime] (ランタイム) には [Java 8] を選択します。
7. [Permissions] (許可) で、[Change default execution role] (デフォルトの実行ロールの変更) を展開します。
8. [Execution role (実行ロール)] で、[Use an existing role (既存のロールを使用する)] を選択します。
9. [Existing role] (既存のロール) には、Lambda 関数が Athena 向けに使用する IAM 実行ロールを選択します (この例では `AthenaLambdaExecutionRole` と呼ばれるロールを使用しています)。
10. [Advanced settings (詳細設定)] を展開します。
11. [Enable Network] (ネットワークの有効化) を選択します。
12. [VPC] で、関数がアクセスする VPC を選択します。
13. [Subnets] (サブネット) には、Lambda が使用する VPC サブネットを選択します。
14. [Security groups] (セキュリティグループ) には、Lambda が使用する VPC セキュリティグループを選択します。
15. [Create function (関数の作成)] を選択します。AWS Lambda コンソールが関数の設定ページを開き、関数の作成を開始します。

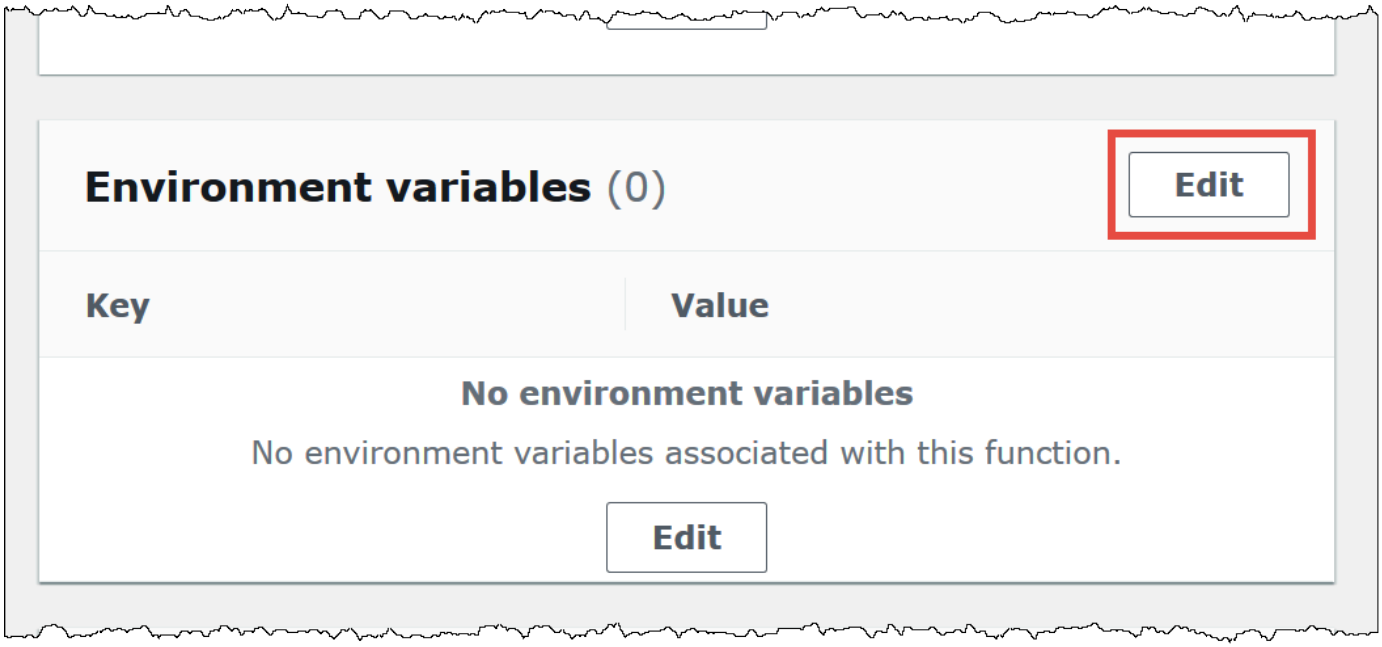
コードをアップロードして Lambda 関数を設定する

コンソールから関数が正常に作成されたことが通知されると、関数コードをアップロードして、その環境変数を設定する準備が整います。

Lambda 関数コードをアップロードしてその環境変数を設定する

1. Lambda コンソールで、指定した関数ページの [Code] (コード) タブが開かれていることを確認します。

2. [Code source] (コードソース) で、[Upload from] (アップロード元) をクリックした後、[.zip or .jar file] (.zip または.jar ファイル) を選択します。
3. 先ほど生成した `hms-lambda-func-1.0-SNAPSHOT-withdep.jar` ファイルをアップロードします。
4. [Lambda 関数] ページで、[設定] タブを選択します。
5. 左側のペインで、[Environment variables] (環境変数) をクリックします。
6. [環境変数] セクションで、[編集] を選択します。

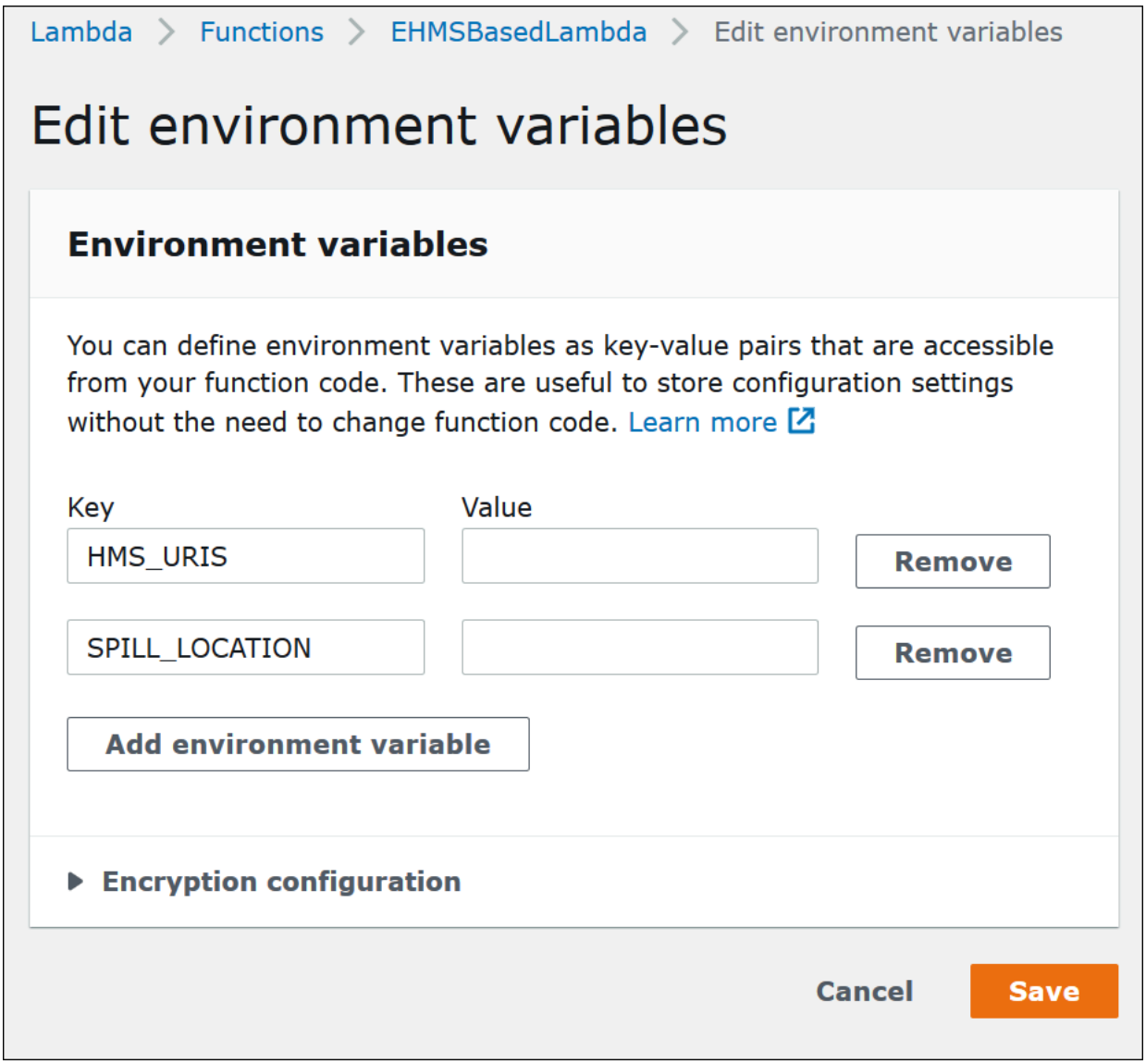


7. [Edit environment variables] (環境変数の編集) ページで、[Add environment variable] (環境変数の追加) オプションを使用し、以下の環境変数のキーと値を追加します。

   - HMS_URIS – 以下の構文を使用して、ポート 9083 で Thrift プロトコルを使用する Hive メタストアホストの URI を入力します。

     ```
     thrift://<host_name>:9083
     ```

   - SPILL_LOCATION - Lambda 関数のレスポンスサイズが 4 MB を超える場合、スピルオーバーメタデータを保持するために Amazon Web Services アカウントの Amazon S3 ロケーションを指定します。



8. [Save] を選択します。

これで、Lambda 関数を使用して Hive メタストアに接続するように、Athena を設定するための準備ができました。この手順については、「デプロイ済みの Hive メタストアコネクタを使用するように Athena を設定する」を参照してください。

デプロイ済みの Hive メタストアコネクタを使用するように Athena を設定する

アカウントに `AthenaHiveMetastoreFunction` のような Lambda データソースコネクタをデプロイした後、その使用を Athena に設定します。これを行うには、Athena クエリで使用する外部 Hive メタストアを参照するためのデータソース名を作成します。

既存の Lambda 関数を使用して Athena を Hive メタストアに接続するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. [Data sources] (データソース) を選択します。
4. [Data sources] (データソース) タブで [Connect data source] (データソースを接続する) を選択します。
5. [Choose a data source] (データソースの選択) ページの [Data sources] (データソース) で、[S3 - Apache Hive metastore] (S3 – Apache Hive メタストア) を選択します。
6. [Next] を選択します。
7. [Data source details] (データソースの詳細) セクションの [Data source name] (データソース名) に、Athena からデータソースをクエリする際に SQL ステートメントで使用する名前 (例えば `MyHiveMetastore`) を入力します。名前は最大 127 文字で、アカウント内で一意である必要があります。この設定は、作成後に変更することはできません。有効な文字は、a ~ z、A ~ Z、0 ~ 9、_ (アンダースコア)、@ (at マーク)、および - (ハイフン) です。`awsdatacatalog`、`hive`、`jmx`、および `system` の名前は Athena によって予約されており、データソース名には使用できません。
8. [接続の詳細] セクションで、[Lambda 関数を選択または入力] ボックスを使って、先ほど作成した関数の名前を選択します。Lambda 関数の ARN が表示されます。
9. (オプション) [Tags] (タグ) でキーと値のペアを追加して、このデータソースに関連付けます。タグの詳細については、Athena リソースへのタグ付けを参照してください。

10. [Next] を選択します。
11. [Review and create] (確認と作成) ページで、データソースの詳細について確認し、[Add data source] (データソースの追加) を選択します。
12. データソースのページの [データソースの詳細] セクションには、新しいコネクタに関する情報が表示されます。

    これで、指定した [Data source name] (データソース名) を使用して、Athena の SQL クエリで Hive メタストアを参照できるようになります。

    SQL クエリで次の構文例を使用する際には、`ehms-catalog` の部分を先に指定したデータソース名に置き換えます。

    ```
    SELECT * FROM ehms-catalog.CustomerData.customers
    ```

13. 作成したデータソースを表示、編集、または削除するには、「データソースの管理」を参照してください 。

外部 Hive メタストアクエリでのデフォルトのデータソース名の使用

外部 Hive メタストアで DML クエリと DDL クエリを実行するとき、クエリエディタでカタログ名が選択されている場合は、カタログ名を省略することでクエリ構文を簡略化できます。この機能には一定の制限が適用されます。

DML ステートメント

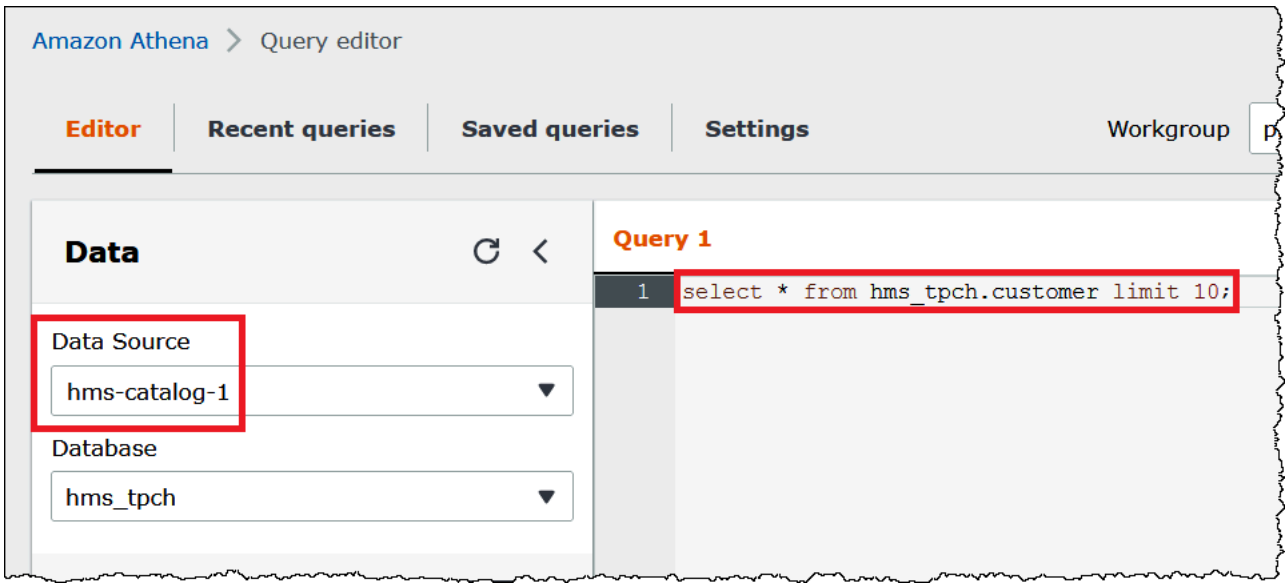
登録済みカタログでクエリを実行するには

1. 次の例のように、構文 `[[data_source_name].database_name].table_name` を使用してデータベースの前にデータソース名を配置できます。

    ```
    select * from  "hms-catalog-1".hms_tpch.customer limit 10;
    ```

2. 使用するデータソースが、クエリエディタ内ですでに選択されている場合は、次の例のようにクエリからカタログ名を省略できます。

    ```
    select * from hms_tpch.customer limit 10:
    ```



3. クエリで複数のデータソースを使用する場合、デフォルトのデータソース名のみ省略が可能で、デフォルト以外のすべてのデータソースについては、フルネームを指定する必要があります。

   例えば、クエリエディタでデフォルトのデータソースとして AwsDataCatalog が選択されているとします。次に抜粋した、クエリの FROM ステートメントでは、最初の 2 つのデータソース名については完全修飾名で指定していますが、3 番目のデータソースの名前は、AWS Glue データカタログに含まれているため省略しています。

```
...
FROM ehms01.hms_tpch.customer,
         "hms-catalog-1".hms_tpch.orders,
         hms_tpch.lineitem
...
```

DDL ステートメント

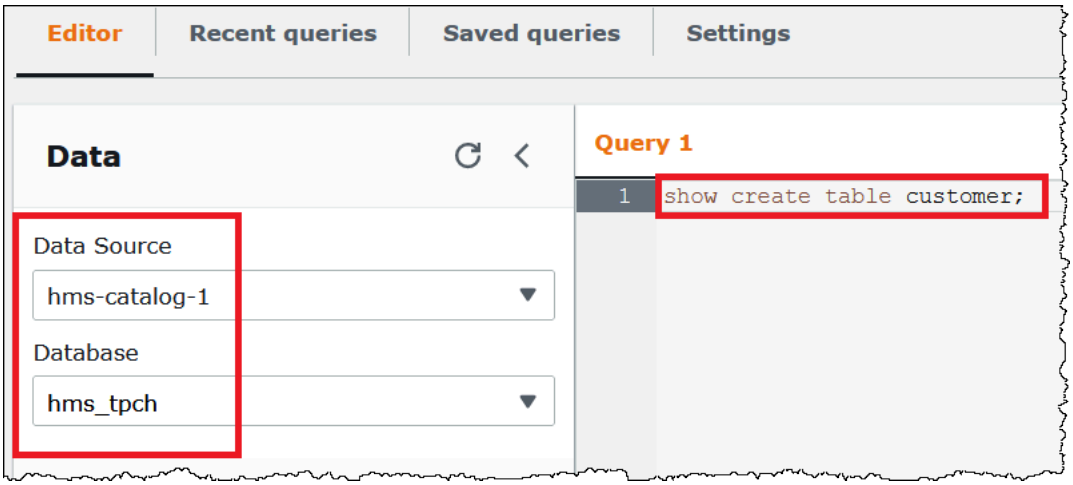
以下の Athena DDL ステートメントは、カタログ名のプレフィックスをサポートします。他の DDL ステートメントのカタログ名プレフィックスは、構文エラーの原因となります。

```
SHOW TABLES [IN [catalog_name.]database_name] ['regular_expression']

SHOW TBLPROPERTIES [[catalog_name.]database_name.]table_name [('property_name')]

SHOW COLUMNS IN [[catalog_name.]database_name.]table_name
```

```
SHOW PARTITIONS [[catalog_name.]database_name.]table_name

SHOW CREATE TABLE [[catalog_name.][database_name.]table_name

DESCRIBE [EXTENDED | FORMATTED] [[catalog_name.][database_name.]table_name [PARTITION
 partition_spec] [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

クエリエディタでデータソースとデータベースが選択されている場合には、DML のステートメントと同様に、クエリからデータソースとデータベースのプレフィックスを省略できます。

次の図では、クエリエディタでデータソース hms-catalog-1 とデータベース hms_tpch が選択されています。クエリ自体から、プレフィックス hms-catalog-1 と データベース名 hms_tpch が省略されていても、show create table customer ステートメントは成功します。



JDBC 接続文字列でのデフォルトデータソースの指定

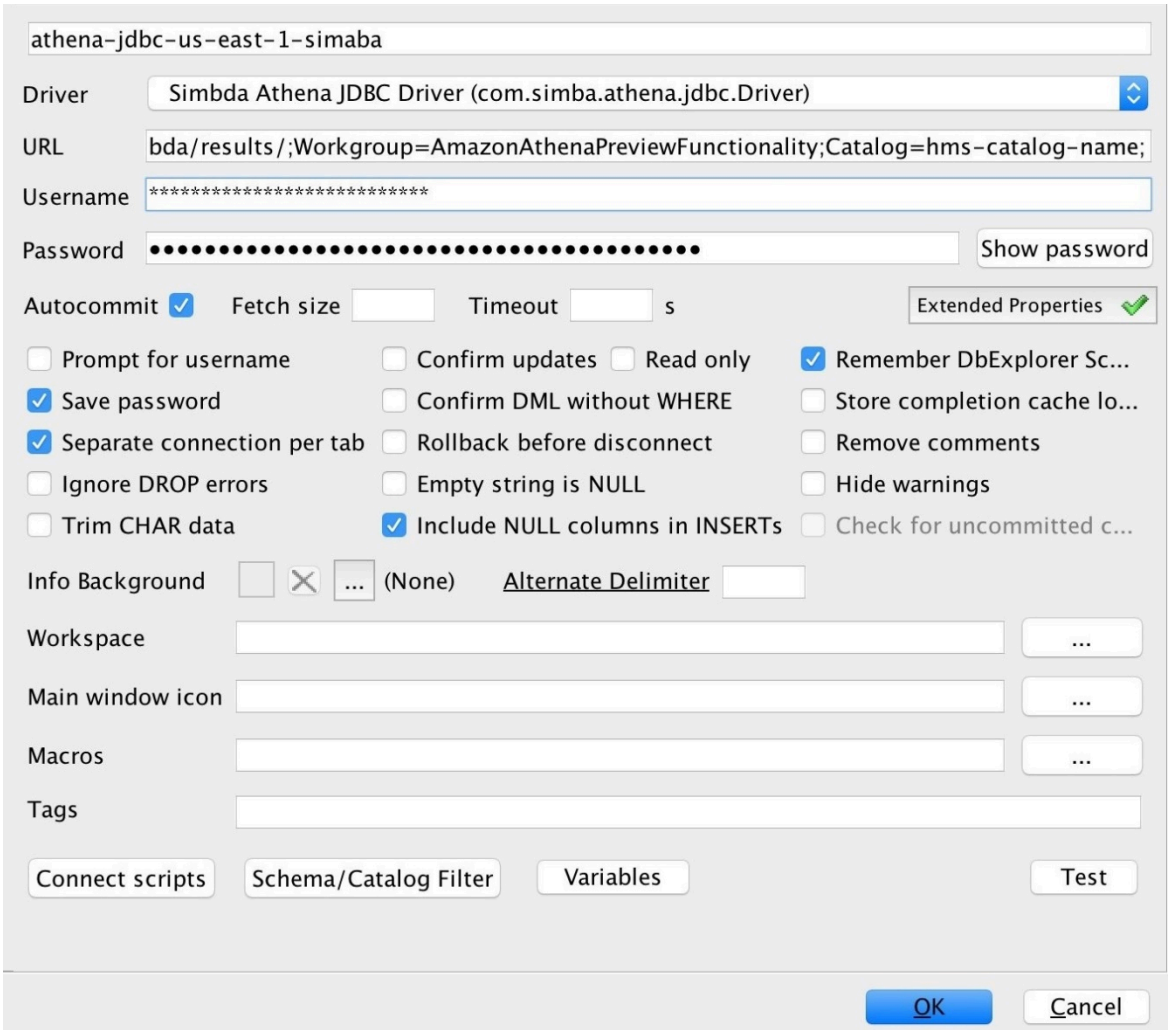
Athena JDBC ドライバーを使用して Athena を外部 Hive メタストアに接続する場合、SQL Workbench などの SQL エディタで、Catalog パラメータを使用して接続文字列にデフォルトのデータソース名を指定します。

> **Note**
>
> 最新の Athena JDBC ドライバーをダウンロードするには、「Athena での JDBC ドライバーの使用」を参照してください。

次の接続文字列は、デフォルトのデータソースに *hms-catalog-name* を指定しています。

```
    jdbc:awsathena://AwsRegion=us-east-1;S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/
lambda/results/;Workgroup=AmazonAthenaPreviewFunctionality;Catalog=hms-catalog-name;
```

以下の画像は、SQL Workbench で設定されている、JDBC 接続 URL の例を示しています。



Hive ビューの使用

Athena を使用して、外部 Apache Hive メタストア内の既存のビューをクエリできます。Athena は、元のビューを変更したり、翻訳を保存したりすることなく、ランタイムにその場でビューを翻訳します。

例えば、Athena ではサポートされていない構文 (`LATERAL VIEW explode()`) を使用する次のような Hive ビューがあるとします。

```
CREATE VIEW team_view AS
```

```
SELECT team, score
FROM matches
LATERAL VIEW explode(scores) m AS score
```

Athena は、Hive ビューのクエリ文字列を、Athena が実行できる次のようなステートメントに翻訳します。

```
SELECT team, score
FROM matches
CROSS JOIN UNNEST(scores) AS m (score)
```

外部の Hive メタストアを Athena に接続する方法の詳細については、「外部 Hive メタストア用の Athena データコネクタの使用」(Athena データコネクタを外部の Hive メタストアに使用する) を参照してください。

考慮事項と制約事項

Athena から Hive ビューをクエリする際は、以下の点を考慮してください。

- Athena では、Hive ビューの作成をサポートしていません。外部 Hive メタストアに Hive ビューを作成し、そこに Athena からクエリを実行することができます。
- Athena では、Hive ビューのカスタム UDF をサポートしていません。
- Athena コンソールの既知の問題により、Hive ビューは、ビューのリストではなくテーブルのリストに表示されます。
- 翻訳プロセスは自動的に実行されますが、ある特定の Hive 関数は、Hive ビューでサポートされていないか、特別な処理が必要になります。詳細については、以下の セクションを参照してください。

Hive 関数のサポート制限

このセクションでは、Athena が Hive ビューでサポートしていない Hive 関数、および特別な処理が必要な Hive 関数について取り上げます。現在、Athena は主に Hive 2.2.0 の関数をサポートしているため、上位バージョン (Hive 4.0.0 など) でのみ有効な関数を使用することはできません。Hive 関数の全リストについては、Hive の「LanguageManual UDF」(言語マニュアル UDF) を参照してください。

集計関数

特別な処理を必要とする集計関数

Hive ビューの次の集計関数では、特別な処理が必要です。

- 平均 — `avg(INT i)` の代わりに `avg(CAST(i AS DOUBLE))` を使用します。

サポートされていない集計関数

Athena の Hive ビューでは、次の Hive 集計関数はサポートされていません。

```
covar_pop
histogram_numeric
ntile
percentile
percentile_approx
```

Athena の Hive ビューでは、`regr_count`、`regr_r2`、`regr_sxx` などの回帰関数はサポートされていません。

サポートされていない日付関数

Athena の Hive ビューでは、次の Hive 日付関数はサポートされていません。

```
date_format(date/timestamp/string ts, string fmt)
day(string date)
dayofmonth(date)
extract(field FROM source)
hour(string date)
minute(string date)
month(string date)
quarter(date/timestamp/string)
second(string date)
weekofyear(string date)
year(string date)
```

サポートされていないマスク関数

Athena の Hive ビューでは、`mask()` や `mask_first_n()` などの Hive マスク関数はサポートされていません。

その他の関数

特別な処理を必要とするその他の関数

Hive ビューの次の種々の関数では、特別な処理が必要です。

- md5 – Athena では `md5(binary)` はサポートされますが、`md5(varchar)` はサポートされません。
- Explode – 次の構文で使用される場合は、Athena で `explode` がサポートされます。

```
LATERAL VIEW [OUTER] EXPLODE(<argument>)
```

- Posexplode – 次の構文で使用される場合は、Athena で `posexplode` がサポートされます。

```
LATERAL VIEW [OUTER] POSEXPLODE(<argument>)
```

  `(pos, val)` 出力で、Athena は pos 列を `BIGINT` として扱います。このため、古いビューを回避するには、pos 列を `BIGINT` にキャストする必要がある場合があります。以下に示しているのはこの手法の例です。

```
SELECT CAST(c AS BIGINT) AS c_bigint, d
FROM table LATERAL VIEW POSEXPLODE(<argument>) t AS c, d
```

サポートされていないその他の関数

Athena の Hive ビューでは、次の Hive 関数はサポートされていません。

```
aes_decrypt
aes_encrypt
current_database
current_user
inline
java_method
logged_in_user
reflect
sha/sha1/sha2
stack
version
```

演算子

特別な処理を必要とする演算子

Hive ビューの次の演算子では、特別な処理が必要です。

- Mod 演算子 (%) – DOUBLE 型は暗黙的に DECIMAL(x,y) にキャストされるため、次の構文は View is stale (ビューが古くなっています) エラーメッセージの原因となる可能性があります。

  ```
  a_double % 1.0 AS column
  ```

  この問題を回避するには、次の例のように CAST を使用します。

  ```
  CAST(a_double % 1.0 as DOUBLE) AS column
  ```

- 除算演算子 (/) – Hive では、int を int で除算すると double になります。Athena では、同じ演算で、切り捨てられた int が生成されます。

サポートされていない演算子

Athena の Hive ビューでは、次の演算子はサポートされていません。

~A – ビット単位の NOT

A ^ b – ビット単位の XOR

A & b – ビット単位の AND

A | b – ビット単位の OR

A <=> b – Null 以外のオペランドの等価 (=) 演算子と同じ結果を返します。両方とも NULL の場合は TRUE を返し、いずれかが NULL の場合は FALSE を返します。

文字列関数

特別な処理を必要とする文字列関数

Hive ビューの次の Hive 文字列関数では、特別な処理が必要です。

- chr(bigint|double a) – Hive では負の引数を使用できますが、Athena では使用できません。
- instr(string str, string substr) – instr 関数の Athena マッピングでは、INT ではなく BIGINT が返されるため、次の構文を使用します。

```
CAST(instr(string str, string substr) as INT)
```

このステップがない場合、ビューは古いものと見なされます。

- length(string a) – `length` 関数の Athena マッピングでは、`INT` ではなく `BIGINT` が返されるため、ビューが古いものと見なされないように、次の構文を使用します。

```
CAST(length(string str) as INT)
```

サポートされていない文字列関数

Athena の Hive ビューでは、次の Hive 文字列関数はサポートされていません。

```
ascii(string str)
character_length(string str)
decode(binary bin, string charset)
encode(string src, string charset)
elt(N int,str1 string,str2 string,str3 string,...)
field(val T,val1 T,val2 T,val3 T,...)
find_in_set(string str, string strList)
initcap(string A)
levenshtein(string A, string B)
locate(string substr, string str[, int pos])
octet_length(string str)
parse_url(string urlString, string partToExtract [, string keyToExtract])
printf(String format, Obj... args)
quote(String text)
regexp_extract(string subject, string pattern, int index)
repeat(string str, int n)
sentences(string str, string lang, string locale)
soundex(string A)
space(int n)
str_to_map(text[, delimiter1, delimiter2])
substring_index(string A, string delim, int count)
```

サポートされていない XPath 関数

Athena の Hive ビューでは、`xpath`、`xpath_short`、`xpath_int` などの Hive XPath 関数はサポートされていません。

トラブルシューティング

Athena で Hive ビューを使用すると、次の問題が発生する場合があります。

- View *<view name>* is stale (ビュー <ビュー名> が古くなっています) – このメッセージは通常、Hive と Athena のビューで型の不一致が存在することを示します。Hive の「LanguageManual UDF」(言語マニュアル UDF) ドキュメントと Presto の「Functions and Operators」(関数と演算子) ドキュメントの同じ関数が異なるシグネチャを持つ場合は、不一致のデータ型をキャストしてみてください。
- Function not registered (関数が登録されていません) – 現在、Athena では、この関数をサポートしていません。詳細については、このドキュメントの前述の情報を参照してください。

Hive メタストアで AWS CLI を使用する

`aws athena` CLI コマンドを使用して、Athena で使用する Hive メタストアデータカタログを管理できます。Athena で使用するカタログを 1 つ、または複数定義したら、`aws athena` DDL コマンドと DML コマンドでこれらのカタログを参照できます。

AWS CLI を使用して Hive メタストアカタログを管理する

カタログの登録: Create-data-catalog

データカタログを登録するには、`create-data-catalog` コマンドを使用します。`name` パラメータを使用して、カタログに使用する名前を指定します。Lambda 関数の ARN を `parameters` 引数の `metadata-function` オプションに渡します。新しいカタログのタグを作成するには、`tags` パラメータを、スペースで区切った 1 つ以上の `Key=`*`key`*`,Value=`*`value`* 引数ペアとともに使用します。

次の例では、`hms-catalog-1` という名前の Hive メタストアカタログを登録します。コマンドは、読みやすい形式にしてあります。

```
$ aws athena create-data-catalog
 --name "hms-catalog-1"
 --type "HIVE"
 --description "Hive Catalog 1"
 --parameters "metadata-function=arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3,sdk-version=1.0"
 --tags Key=MyKey,Value=MyValue
 --region us-east-1
```

カタログの詳細の表示: Get-data-catalog

カタログの詳細を表示するには、次の例のように、カタログの名前を `get-data-catalog` コマンドに渡します。

```
$ aws athena get-data-catalog --name "hms-catalog-1" --region us-east-1
```

次のサンプル結果は JSON 形式です。

```
{
    "DataCatalog": {
        "Name": "hms-catalog-1",
        "Description": "Hive Catalog 1",
        "Type": "HIVE",
        "Parameters": {
            "metadata-function": "arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3",
            "sdk-version": "1.0"
        }
    }
}
```

登録済みカタログのリスト化: List-data-catalogs

登録されたカタログをリスト化するには、`list-data-catalogs` コマンドを使用し、必要に応じて次の例のようにリージョンを指定します。リストされているカタログには、常に AWS Glue が含まれます。

```
$ aws athena list-data-catalogs --region us-east-1
```

次のサンプル結果は JSON 形式です。

```
{
    "DataCatalogs": [
        {
            "CatalogName": "AwsDataCatalog",
            "Type": "GLUE"
        },
        {
            "CatalogName": "hms-catalog-1",
```

```
            "Type": "HIVE",
            "Parameters": {
                "metadata-function": "arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3",
                "sdk-version": "1.0"
            }
        }
    ]
}
```

カタログの更新: Update-data-catalog

データカタログを更新するには、次の例のように update-data-catalog コマンドを使用します。コマンドは、読みやすい形式にしてあります。

```
$ aws athena update-data-catalog
 --name "hms-catalog-1"
 --type "HIVE"
 --description "My New Hive Catalog Description"
 --parameters "metadata-function=arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new,sdk-version=1.0"
 --region us-east-1
```

カタログの削除: Delete-data-catalog

データカタログを削除するには、次の例のように delete-data-catalog コマンドを使用します。

```
$ aws athena delete-data-catalog --name "hms-catalog-1" --region us-east-1
```

データベース詳細の表示: Get-database

データベースの詳細を表示するには、次の例のように、カタログとデータベースの名前を get-database コマンドに渡します。

```
$ aws athena get-database --catalog-name hms-catalog-1 --database-name mydb
```

次のサンプル結果は JSON 形式です。

```
{
    "Database": {
```

```
        "Name": "mydb",
        "Description": "My database",
        "Parameters": {
            "CreatedBy": "Athena",
            "EXTERNAL": "TRUE"
        }
    }
}
```

カタログ内のデータベースのリスト化: List-databases

カタログのデータベースをリスト化するには、`list-databases` コマンドを使用し、必要に応じて次の例のようにリージョンを指定します。

```
$ aws athena list-databases --catalog-name AwsDataCatalog --region us-west-2
```

次のサンプル結果は JSON 形式です。

```
{
    "DatabaseList": [
        {
            "Name": "default"
        },
        {
            "Name": "mycrawlerdatabase"
        },
        {
            "Name": "mydatabase"
        },
        {
            "Name": "sampledb",
            "Description": "Sample database",
            "Parameters": {
                "CreatedBy": "Athena",
                "EXTERNAL": "TRUE"
            }
        },
        {
            "Name": "tpch100"
        }
    ]
}
```

テーブルの詳細の表示: Get-table-metadata

カラム名やデータ型を含むテーブルのメタデータを表示するには、次の例のように、カタログ、データベース、およびテーブル名を `get-table-metadata` コマンドに渡します。

```
$ aws athena get-table-metadata --catalog-name AwsDataCatalog --database-name mydb --
table-name cityuseragent
```

次のサンプル結果は JSON 形式です。

```
{
    "TableMetadata": {
        "Name": "cityuseragent",
            "CreateTime": 1586451276.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "city",
                    "Type": "string"
                },
                {
                    "Name": "useragent1",
                    "Type": "string"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "COLUMN_STATS_ACCURATE": "false",
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "last_modified_by": "hadoop",
                "last_modified_time": "1586454879",
                "location": "s3://DOC-EXAMPLE-BUCKET/",
                "numFiles": "1",
                "numRows": "-1",
                "outputformat":
 "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "rawDataSize": "-1",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
 "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "totalSize": "61"
```

```
            }
        }
}
```

データベース内のすべてのテーブルのメタデータの表示: List-table-metadata

データベース内のすべてのテーブルのメタデータを表示するには、カタログの名前とデータベース名を `list-table-metadata` コマンドに渡します。`list-table-metadata` コマンドは、テーブル名を指定しない点を除いて、`get-table-metadata` コマンドに似ています。結果件数を制限するには、次の例のように `--max-results` オプションを使用します。

```
$ aws athena list-table-metadata --catalog-name AwsDataCatalog --database-name sampledb
 --region us-east-1 --max-results 2
```

次のサンプル結果は JSON 形式です。

```
{
    "TableMetadataList": [
        {
            "Name": "cityuseragent",
            "CreateTime": 1586451276.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "city",
                    "Type": "string"
                },
                {
                    "Name": "useragent1",
                    "Type": "string"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "COLUMN_STATS_ACCURATE": "false",
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "last_modified_by": "hadoop",
                "last_modified_time": "1586454879",
                "location": "s3://DOC-EXAMPLE-BUCKET/",
                "numFiles": "1",
```

```
                "numRows": "-1",
                "outputformat":
"org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "rawDataSize": "-1",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "totalSize": "61"
            }
        },
        {
            "Name": "clearinghouse_data",
            "CreateTime": 1589255544.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "location",
                    "Type": "string"
                },
                {
                    "Name": "stock_count",
                    "Type": "int"
                },
                {
                    "Name": "quantity_shipped",
                    "Type": "int"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "location": "s3://DOC-EXAMPLE-BUCKET/",
                "outputformat":
"org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "transient_lastDdlTime": "1589255544"
            }
        }
    ],
```

```
    "NextToken":
 "eyJsYXN0RXZhbHVhdGVkS2V5Ijp7IkhBU0hfS0VZIjp7InMiOiJ0Ljk0YWZjYjk1MjJjNTQ1YmU4Y2I5OWE5NTg0MjFjY
}
```

DDL および DML ステートメントの実行

AWS CLI を使用して DDL および DML 文を実行する場合、次の 2 つの方法のいずれかで Hive メタストアカタログの名前を渡すことができます。

- それをサポートするステートメントに直接渡す。
- `--query-execution-context` `Catalog` パラメータに渡す。

DDL ステートメント

次の例では、`show create table` DDL ステートメントの一部としてカタログ名に直接渡します。コマンドは、読みやすい形式にしてあります。

```
$ aws athena start-query-execution
 --query-string "show create table hms-catalog-1.hms_tpch_partitioned.lineitem"
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

次の DDL `show create table` ステートメントの例では、`Catalog` の `--query-execution-context` パラメータを使用して Hive メタストアカタログ名 `hms-catalog-1` に渡します。コマンドは、読みやすい形式にしてあります。

```
$ aws athena start-query-execution
 --query-string "show create table lineitem"
 --query-execution-context "Catalog=hms-catalog-1,Database=hms_tpch_partitioned"
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

DML ステートメント

次の DML `select` ステートメントの例では、カタログ名をクエリに直接渡します。コマンドは、読みやすい形式にしてあります。

```
$ aws athena start-query-execution
 --query-string "select * from hms-catalog-1.hms_tpch_partitioned.customer limit 100"
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

次の DML select ステートメントの例では、Catalog の --query-execution-context パラメータを使用して Hive メタストアカタログ名 hms-catalog-1 に渡します。コマンドは、読みやすい形式にしてあります。

```
$ aws athena start-query-execution
 --query-string "select * from customer limit 100"
 --query-execution-context "Catalog=hms-catalog-1,Database=hms_tpch_partitioned"
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

リファレンス実装

Athena は、GitHub.com (https://github.com/awslabs/aws-athena-hive-metastore) で外部 Hive メタストア用のコネクタのリファレンス実装を提供します。

リファレンス実装は、次のモジュールを持つ Apache Maven プロジェクトです。

- **hms-service-api** – Lambda 関数と Athena サービスクライアント間の API オペレーションが含まれています。これらの API オペレーションは、HiveMetaStoreService インターフェイスで定義されます。これはサービス契約であるため、このモジュールでは何も変更しないでください。
- **hms-lambda-handler** – すべての Hive メタストア API コールを処理する一連のデフォルト Lambda ハンドラーです。クラス MetadataHandler は、すべての API コールのディスパッチャーです。このパッケージを変更する必要はありません。
- **hms-lambda-func** – 以下のコンポーネントを持つ Lambda 関数の例です。
  - **HiveMetaStoreLambdaFunc** – MetadataHandler を拡張する Lambda 関数の例。
  - **ThriftHiveMetaStoreClient** – Hive メタストアと通信する Thrift クライアント。このクライアントは Hive 2.3.0 用に書かれています。別の Hive バージョンを使用する場合は、このクラスを更新して、応答オブジェクトに互換性があることを確認する必要があります。
  - **ThriftHiveMetaStoreClientFactory** – Lambda 関数の動作を制御します。たとえば、getHandlerProvider() メソッドをオーバーライドして、独自のハンドラープロバイダーを提供できます。
  - hms.properties – Lambda 関数を設定します。ほとんどの場合、次の 2 つのプロパティのみを更新する必要があります。
    - hive.metastore.uris – thrift://*<host_name>*:9083 形式の Hive メタストアの URI。
    - hive.metastore.response.spill.location - レスポンスオブジェクトのサイズが所定のしきい値 (4 MB など) を超える場合にこれらを保存する Amazon S3 の場所。しきい値はプ

ロパティ hive.metastore.response.spill.threshold で定義されます。デフォルト値を変更することは推奨されません。

> **Note**
>
> これら 2 つのプロパティは、Lambda 環境変数の HMS_URIS および SPILL_LOCATION で上書きできます。これらの変数は、異なる Hive メタストアまたはスピルの場所で Lambda 関数を使用するときに、関数のソースコードを再コンパイルする代わりに使用します。

- **hms-lambda-layer** – hms-service-api、hms-lambda-handler、およびその依存関係を .zip ファイルに入れる Maven アセンブリプロジェクト。.zip ファイルは、複数の Lambda 関数による使用のために、Lambda レイヤーとして登録されます。
- **hms-lambda-rnp** - Lambda 関数からのレスポンスを記録し、それを使用してレスポンスを再生します。このモデルを使用して、テスト用の Lambda レスポンスをシミュレートできます。

アーティファクトを自分で構築する

ほとんどのユースケースでは、リファレンス実装を変更する必要はありません。ただし、必要に応じて、ソースコードを変更してアーティファクトを独自に構築し、それらを Amazon S3 の場所にアップロードすることができます。

アーティファクトを構築する前に、hive.metastore.uris モジュール内の hive.metastore.response.spill.location ファイルにあるプロパティ hms.properties および hms-lambda-func を更新します。

アーティファクトを構築するには、Apache Maven をインストールし、mvn install コマンドを実行する必要があります。これは、モジュール hms-lambda-layer の target と呼ばれる出力フォルダにレイヤー .zip ファイルを生成し、モジュール hms-lambd-func に Lambda 関数 .jar ファイルを生成します。

Amazon Athena 横串検索の使用

Amazon S3 以外のソースにデータがある場合は、Athena の横串検索を使用してインプレースでデータをクエリしたり、複数のデータソースからデータを抽出して Amazon S3 に保存するパイプラインを構築したりすることができます。Athena の横串検索では、リレーショナル、非リレーショナル、オブジェクト、およびカスタムデータソースに保存されているデータの全体で SQL クエリを実行することができます。

Athena は横串検索を実行するために、AWS Lambda で実行されるデータソースコネクタを使用します。データソースコネクタは、ターゲットデータソースと Athena 間での変換を実行できるコードです。コネクタは、Athena のクエリエンジンの拡張機能として考えることができます。事前構築された Athena データソースコネクタは、Amazon CloudWatch Logs、Amazon DynamoDB、Amazon DocumentDB、Amazon RDS、および MySQL や Apache 2.0 ライセンスに基づく PostgreSQL などの JDBC 準拠のリレーショナルデータソースといったデータソース用のものです。Athena Query Federation SDK を使用してカスタムコネクタを記述することもできます。データソースコネクタを選択し、設定し、アカウントにデプロイするには、Athena および Lambda コンソールまたは AWS Serverless Application Repository を使用できます。データソースコネクタをデプロイした後、コネクタは SQL クエリで指定できるカタログに関連付けられます。複数のカタログの SQL ステートメントを組み合わせて、1 つのクエリの範囲に複数のデータソースを入れることができます。

データソースに対してクエリが送信されると、Athena が対応するコネクタを呼び出して、読み取る必要があるテーブルの箇所を特定し、並列処理を管理して、フィルター述語をプッシュダウンします。クエリを送信するユーザーに基づいて、コネクタは特定のデータ要素へのアクセスを提供または制限できます。コネクタは、クエリで要求されたデータを返す形式として Apache Arrow を使用します。これにより、コネクタを C、C++、Java、Python、Rust などの言語で実装できます。コネクタは Lambda で処理されるため、Lambda からアクセスできる、クラウドまたはオンプレミスの任意のデータソースからのデータへのアクセスに使用できます。

独自のデータソースコネクタを記述するには、Athena Query Federation SDK を使用して、Amazon Athena が提供し、維持する事前構築されたコネクタの 1 つをカスタマイズできます。GitHub リポジトリからのソースコードのコピーを変更し、コネクタ公開ツールを使用して独自の AWS Serverless Application Repository パッケージを作成できます。

> **Note**
>
> サードパーティーデベロッパーは、データソースコネクタの記述に Athena Query Federation SDK を使用している可能性があります。これらのデータソースコネクタのサポート、またはライセンスに関する問題については、コネクタプロバイダーにお問い合わせください。AWS はこれらのコネクタをテストしておらず、サポートも提供しません。

Athena によって記述され、テストされたデータソースコネクタのリストについては、「使用可能なデータソースコネクタ」を参照してください。

独自のデータソースコネクタの記述については、GitHub の「Example Athena connector」(Example Athena コネクタ) を参照してください。

考慮事項と制約事項

- エンジンバージョン – Athena Federated Query は、Athena エンジンバージョン 2 以降でのみサポートされています。Athena エンジンバージョンの詳細については、「Athena エンジンのバージョニング」を参照してください。
- ビュー – フェデレーティッドデータソースでビューを作成してクエリできるようになりました。フェデレーティッドビューは、基盤となるデータソースではなく、AWS Glue に格納されます。詳細については、「フェデレーションされたビューのクエリ」を参照してください。
- 書き込み操作 — INSERT INTO などの書き込み操作はサポートされていません。これを試みた場合、「This operation is currently not supported for external catalogs (この操作は現在、外部カタログではサポートされていません)」というエラーメッセージが表示されることがあります。
- 料金 – 料金の情報については、「Amazon Athena の料金」を参照してください。

  JDBC ドライバー – フェデレーティッドクエリや外部 Hive メタストアで JDBC ドライバーを使用するには、JDBC 接続文字列に `MetadataRetrievalMethod=ProxyAPI` を含めてください。JDBC ドライバーの詳細については、「JDBC を使用した Amazon Athena への接続」を参照してください。
- Secrets Manager – AWS Secrets Manager で Athena 横串検索機能を使用するには、Secrets Manager に Amazon VPC プライベートエンドポイントを設定する必要があります。詳細については、「AWS Secrets Manager ユーザーガイド」の「Secrets Manager VPC プライベートエンドポイントを作成する」を参照してください。

データソースコネクタは、正しく機能するために、次のリソースへのアクセスが必要になる場合があります。事前に構築されたコネクタを使用する場合は、コネクタの情報を確認して、VPC が正しく設定されていることを確認してください。また、クエリを実行し、コネクタを作成する IAM プリンシパルに、必要なアクションに対する権限があることを確認してください。詳細については、「Athena Federated Query を許可する IAM 許可ポリシーの例」を参照してください。

- Simple Storage Service (Amazon S3) – データコネクタは、Athena のクエリ結果が保存される Amazon S3 内の場所にクエリ結果を書き込むほか、Amazon S3 のスピルバケットにも書き込みます。この Amazon S3 の場所に対する接続と許可が必要です。
- Athena – データソースには、クエリステータスのチェックとオーバースキャンの防止のために、Athena への (または Athena からの) 接続が必要です。
- AWS Glue Data Catalog – コネクタが補足メタデータまたはプライマリメタデータのためにデータカタログを使用する場合は、接続とアクセス許可が必要です。

動画

以下の動画を視聴して、Athena の横串検索の使用に関する詳細を確認してください。

動画: Amazon QuickSight で Amazon Athena 横串検索の結果を分析する

以下の動画は、Athena 横串検索の結果を Amazon QuickSight で分析する方法を説明します。

Analyze results of federated query in Amazon Athena in Amazon QuickSight (Amazon QuickSight で Amazon Athena フェデレーティッドクエリの結果を分析する)

動画: Gaming Analytics Pipeline

以下の動画は、Amazon Athena の横串検索を使用して、ゲームやサービスからのテレメトリデータの取り込み、保存、および分析を実行するためのスケーラブルなサーバーレスデータパイプラインをデプロイする方法を紹介しています。

Game analytics pipeline (ゲーム分析パイプライン)

使用可能なデータソースコネクタ

このセクションでは、Amazon S3 外部のさまざまなデータソースをクエリするために使用できる、事前構築された Athena データソースコネクタをリストします。Athena クエリでコネクタを使用するには、コネクタを設定してアカウントにデプロイします。

考慮事項と制約事項

- 一部の事前構築されたコネクタでは、コネクタを使用する前に VPC とセキュリティグループを作成する必要があります。VPC の作成方法の詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。
- AWS Secrets Manager で Athena 横串検索機能を使用するには、Secrets Manager に Amazon VPC プライベートエンドポイントを設定する必要があります。詳細については、「AWS Secrets Manager ユーザーガイド」の「Secrets Manager VPC プライベートエンドポイントを作成する」を参照してください。
- 述語のプッシュダウンをサポートしていないコネクタの場合、述語を含むクエリの実行には大幅に時間がかかります。小規模なデータセットの場合、スキャンされるデータはほとんどなく、クエリは平均約 2 分で終わります。しかし、大規模なデータセットでは、多くのクエリがタイムアウトする可能性があります。
- 一部のフェデレーティッドデータソースでは、データオブジェクトを参照する用語に、Athena とは異なるものが使用されています。詳細については、「Athena とフェデレーションテーブル名修飾子」を参照してください。

- テーブルを一覧表示するときにページ分割をサポートしないコネクタでは、データベースに多数のテーブルとメタデータがある場合にウェブサービスがタイムアウトする可能性があります。次のコネクタは、テーブルの一覧表示のページ分割をサポートします。
  - DocumentDB
  - DynamoDB
  - MySQL
  - OpenSearch
  - Oracle
  - PostgreSQL
  - Redshift
  - SQL Server

追加情報

- Athena データソースコネクタのデプロイについては、「データソースコネクタのデプロイ」を参照してください。
- Athena データソースコネクタを使用するクエリについては、「フェデレーティッドクエリの実行」を参照してください。
- Athena データソースコネクタの詳細については、GitHub の「Available Connectors」(使用可能なコネクタ) を参照してください。

Athena データソースコネクタ

- Amazon Athena Azure Data Lake Storage (ADLS) Gen2 コネクタ
- Amazon Athena Azure Synapse コネクタ
- Amazon Athena Cloudera Hive コネクタ
- Amazon Athena Cloudera Impala コネクタ
- Amazon Athena CloudWatch コネクタ
- Amazon Athena CloudWatch Metrics コネクタ
- Amazon Athena AWS CMDB コネクタ
- Amazon Athena IBM Db2 コネクタ
- Amazon Athena IBM Db2 AS/400 (Db2 iSeries) コネクタ
- Amazon Athena DocumentDB コネクタ

- Amazon Athena DynamoDB コネクタ
- Amazon Athena Google BigQuery コネクタ
- Amazon Athena Google Cloud Storage コネクタ
- Amazon Athena HBase コネクタ
- Amazon Athena Hortonworks コネクタ
- Amazon Athena の Apache Kafka コネクタ
- Amazon Athena MSK コネクタ
- Amazon Athena MySQL コネクタ
- Amazon Athena Neptune コネクタ
- Amazon Athena OpenSearch コネクタ
- Amazon Athena Oracle コネクタ
- Amazon Athena PostgreSQL コネクタ
- Amazon Athena Redis コネクタ
- Amazon Athena Redshift コネクタ
- Amazon Athena SAP HANA コネクタ
- Amazon Athena Snowflake コネクタ
- Amazon Athena Microsoft SQL Server コネクタ
- Amazon Athena Teradata コネクタ
- Amazon Athena Timestream コネクタ
- Amazon Athena TPC Benchmark DS (TPC-DS) コネクタ
- Amazon Athena Vertica コネクタ

> **Note**
>
> AthenaJdbcConnector (最新バージョン 2022.4.1) は廃止されました。代わりに、MySQL、Redshift、PostgreSQL などのデータベース専用のコネクタを使用してください。

Amazon Athena Azure Data Lake Storage (ADLS) Gen2 コネクタ

Azure Data Lake Storage (ADLS) Gen2 用の Amazon Athena コネクタを使用すれば、Amazon Athena で ADLS に保存されたデータに SQL クエリを実行できます。Athena は、データレイクに保存されているファイルに直接アクセスすることはできません。

- ワークフロー – このコネクタは、`com.microsoft.sqlserver.jdbc.SQLServerDriver` ドライバーを使用する JDBC インターフェイスを実装します。このコネクタが Azure Synapse エンジンにクエリを渡し、その後、Azure Synapse エンジンがデータレイクにアクセスします。
- データ処理と S3 – 通常、Lambda コネクタは、Amazon S3 に転送することなく、データを直接クエリします。ただし、Lambda 関数から返されたデータが Lambda の制限を超えると、そのデータは、Athena が超過分を読み取れるように指定した Amazon S3 スピルバケットに書き込まれます。
- AAD 認証 – AAD は、Azure Synapse コネクタの認証方法として使用できます。AAD を使用するには、コネクタが使用する JDBC 接続文字列に、`authentication=ActiveDirectoryServicePrincipal`、`AADSecurePrincipalId`、および `AADSecurePrincipalSecret` の URL パラメータが含まれている必要があります。これらのパラメータは、直接渡すことも、Secrets Manager から渡すこともできます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- フィルター条件における日付とタイムスタンプのデータ型は、適切なデータ型にキャストする必要があります。

用語

次の用語は、Azure Data Lake Storage Gen2 コネクタに関連しています。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

Azure Data Lake Storage Gen2 コネクタを設定するには、このセクションの Lambda 環境変数を使用します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
datalakegentwo://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`DataLakeGen2MuxCompositeHandler`
メタデータハンドラー	`DataLakeGen2MuxMetadataHandler`
レコードハンドラー	`DataLakeGen2MuxRecordHandler`

マルチプレックスハンドラーのパラメータ

パラメータ	説明
`$`*`catalog`*`_connection_string`	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが `mydatalakegentwocatalog` の場合、環境変数の名前は `mydatalakegentwocatalog_connection_string` になります。
`default`	必須。デフォルトの接続文字列。この文字列は、カタログが `lambda:${` *`AWS_LAMBDA_FUNCTION_NAME`* `}` の場合に使用されます。

以下は、2 つのデータベースインスタンス (デフォルト値の `datalakegentwo1`、および `datalakegentwo2`) をサポートする、DataLakeGen2 MUX Lambda 関数のプロパティ例です。

プロパティ	Value
`default`	`datalakegentwo://jdbc:sqlserver://adlsgentwo1.`*`hostname`*`:`*`port`*`;databaseName=`*`database_name`*`;${`*`secret1_name`*`}`

プロパティ	Value
datalakegentwo_catalog1_connection_string	datalakegentwo://jdbc:sqlserver://adlsgentwo1. *hostname*:*port*;databaseName= *database_name* ;${*secret1_name* }
datalakegentwo_catalog2_connection_string	datalakegentwo://jdbc:sqlserver://adlsgentwo2. *hostname*:*port*;databaseName= *database_name* ;${*secret2_name* }

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 ${secret1_name} が含まれています。

```
datalakegentwo://jdbc:sqlserver://hostname:port;databaseName=database_name;
${secret1_name}
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
datalakegentwo://
jdbc:sqlserver://
hostname:port;databaseName=database_name;user=user_name;password=password
```

単一接続ハンドラーの使用

単一の Azure Data Lake Storage Gen2 インスタンスに接続するには、次の単一接続メタデータとレコードハンドラーを使用します。

ハンドラーのタイプ	Class
複合ハンドラー	DataLakeGen2CompositeHandler
メタデータハンドラー	DataLakeGen2MetadataHandler
レコードハンドラー	DataLakeGen2RecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の Azure Data Lake Storage Gen2 インスタンス用のプロパティ例を次に示します。

プロパティ	Value
default	datalakegentwo://jdbc:sqlserver://*hostname*:*port*;databaseName=;${ *secret_name* }

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
spill_bucket	必須。スピルバケット名。
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表では、ADLS Gen2 と Arrow に対応させてデータ型を示しています。

ADLS Gen2	Arrow
bit	TINYINT

ADLS Gen2	Arrow
tinyint	SMALLINT
smallint	SMALLINT
整数	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
real	FLOAT4
datetime	Date(MILLISECOND)
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
time	VARCHAR
datetimeoffset	Date(MILLISECOND)
char[n]	VARCHAR
varchar[n/max]	VARCHAR

パーティションと分割

Azure Data Lake Storage Gen2 は、データファイルの保存に Hadoop 互換の Gen2 ブロブストレージを使用します。これらのファイルからのデータは、Azure Synapse エンジンによりクエリされます。Azure Synapse エンジンは、ファイルシステムに保存されている Gen2 データを外部テーブルとして扱います。パーティションはデータの型に基づいて実装されます。データがすでにパーティション化されており、Gen 2 ストレージシステム内で分散されている場合、コネクタは、データを単一のスプリットとして取得します。

パフォーマンス

Azure Data Lake Storage Gen2 コネクタは、複数のクエリを一度に実行するとクエリのパフォーマンスが遅くなり、スロットリングの対象となります。

Athena Azure Data Lake Storage Gen2 コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。単純な述語と複雑な式はコネクタにプッシュダウンされるため、スキャンされるデータ量が減少し、クエリ実行のランタイムが短縮されます。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの `WHERE` 句内の式です。Athena Azure Data Lake Storage Gen2 コネクタは、これらの式を組み合わせて Azure Data Lake Storage Gen2 に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Azure Data Lake Storage Gen2 コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
```

```
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

パススルークエリ

Azure Data Lake Storage Gen2 コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Azure Data Lake Storage Gen2 でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Azure Data Lake Storage Gen2 内のデータソースにクエリをプッシュダウンします。クエリは `customer` テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

JDBC ドライバーの最新バージョンの情報については、GitHub.com で、Azure Data Lake Storage Gen2 コネクタ用の pom.xml ファイルを確認してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Azure Synapse コネクタ

Azure Synapse Analytics 用の Amazon Athena コネクタを使用すると、Amazon Athena で Azure Synapse データベースに JDBC を使用した SQL クエリを実行できます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- フィルター条件では、`Date` と `Timestamp` データ型を適切なデータ型に型変換する必要があります。
- `Real` および `Float` 型の負の値を検索するには、<= または >= 演算子を使用します。
- `binary`、`varbinary`、`image`、および `rowversion` データ型はサポートされていません。

用語

Synapse コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。

- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、connection_string プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して Synapse コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
synapse://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	SynapseMuxCompositeHandler
メタデータハンドラー	SynapseMuxMetadataHandler
レコードハンドラー	SynapseMuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが mysynapse

パラメータ	説明
	catalog の場合、環境変数の名前は mysynapsecatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

synapse1 (デフォルト) と synapse2 の 2 つのデータベースインスタンスをサポートする Synapse MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	synapse://jdbc:synapse://synapse1.hostname:port;databaseName= *<database_name>* ;${*secret1_name* }
synapse_catalog1_connection_string	synapse://jdbc:synapse://synapse1.hostname:port;databaseName= *<database_name>* ;${*secret1_name* }
synapse_catalog2_connection_string	synapse://jdbc:synapse://synapse2.hostname:port;databaseName= *<database_name>* ;${*secret2_name* }

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets

Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列はシークレット名 ${secret_name} を含んでいます。

```
synapse://jdbc:synapse://hostname:port;databaseName=<database_name>;${secret_name}
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
synapse://jdbc:synapse://
hostname:port;databaseName=<database_name>;user=<user>;password=<password>
```

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Synapse インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	`SynapseCompositeHandler`
メタデータハンドラー	`SynapseMetadataHandler`
レコードハンドラー	`SynapseRecordHandler`

単一接続ハンドラーのパラメータ

パラメータ	説明
`default`	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、`default` 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の Synapse インスタンス用のプロパティ例を次に示します。

プロパティ	Value
`defa`	`synapse://jdbc:sqlserver://hostname:port;databaseName=` *`<database_name>`* `;${`*`secret_name`*` }`

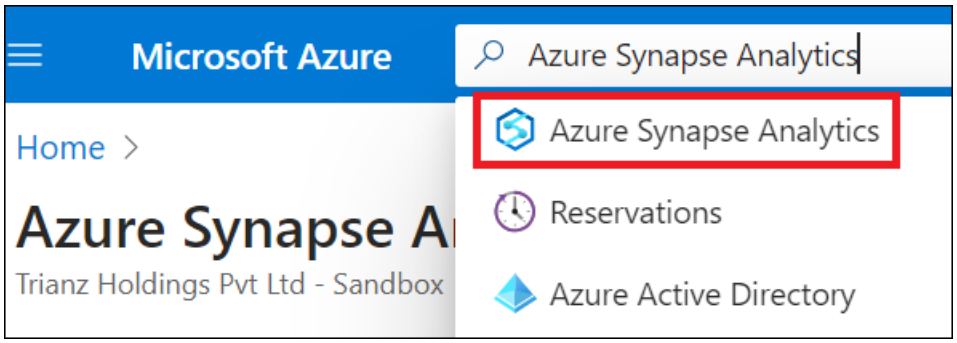
Active Directory 認証の設定

Amazon Athena Azure Synapse コネクタは Microsoft Active Directory 認証をサポートしています。開始する前に、Microsoft Azure ポータルで管理ユーザーを設定し、AWS Secrets Manager を使用してシークレットを作成する必要があります。

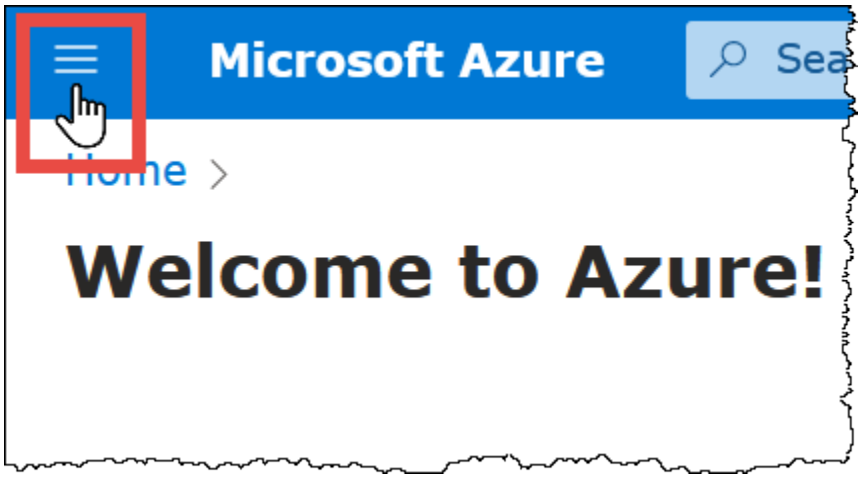
Active Directory 管理者ユーザーを設定するには

1. 管理者権限を持つアカウントを使用して、https://portal.azure.com/ で Microsoft Azure ポータルにサインインします。

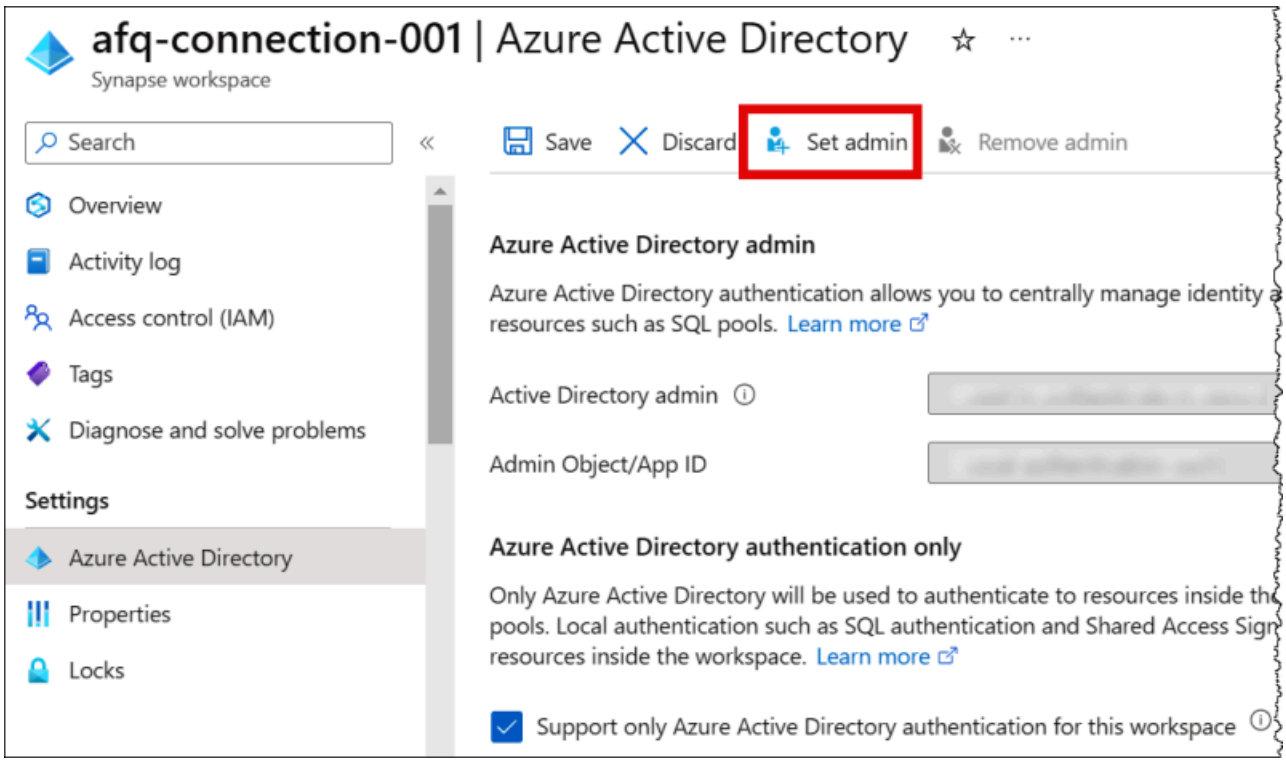
2. 検索ボックスにAzure Synapse Analytics と入力し、[Azure Synapse Analytics] を選択します。



3. 左側のメニューを開きます。



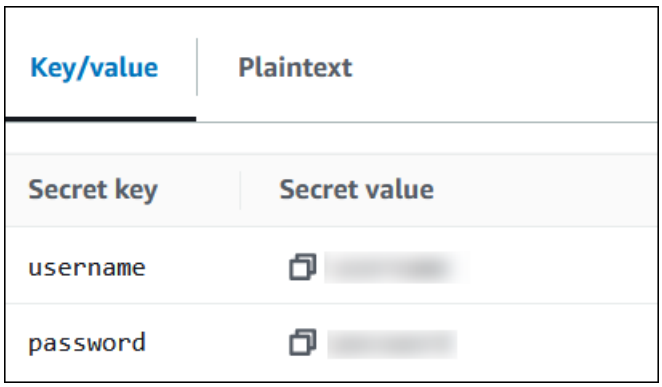
4. ナビゲーションペインで、[Azure Active Directory] を選択します。
5. [管理者の設定] タブで、Active Directory 管理者を新規ユーザーまたは既存のユーザーに設定します。



6. AWS Secrets Manager に、管理者のユーザー名とパスワードの認証情報を保存します。Secrets Manager でのシークレットの作成については、「AWS Secrets Manager シークレットを作成する」を参照してください。

Secrets Manager で自分のシークレットを確認するには

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. ナビゲーションペインで [Secrets] (シークレット) を選択します。
3. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
4. 自分のシークレットの詳細ページで、[Retrieve secret value] (シークレットの値を取得する) を選択します。



接続文字列の変更

コネクタの Active Directory 認証を有効にするには、次の構文を使用して接続文字列を変更します。

```
synapse://jdbc:synapse://
hostname:port;databaseName=database_name;authentication=ActiveDirectoryPassword;
{secret_name}
```

ActiveDirectoryServicePrincipal の使用

Amazon Athena Azure Synapse コネクタも `ActiveDirectoryServicePrincipal` をサポートしています。これを有効にするには、接続文字列を次のように変更します。

```
synapse://jdbc:synapse://
hostname:port;databaseName=database_name;authentication=ActiveDirectoryServicePrincipal;
{secret_name}
```

`secret_name` の場合、ユーザー名にはアプリケーション ID またはクライアント ID を指定し、パスワードにはサービスプリンシパル ID のシークレットを指定します。

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス

パラメータ	説明
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、Synapse と Apache Arrow に対応するデータ型を示します。

Synapse	Arrow
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
整数	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
real	FLOAT4
datetime	Date(MILLISECOND)

Synapse	Arrow
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
time	VARCHAR
datetimeoffset	Date(MILLISECOND)
char[n]	VARCHAR
varchar[n/max]	VARCHAR
nchar[n]	VARCHAR
nvarchar[n/max]	VARCHAR

パーティションと分割

パーティションは、varchar 型の単一パーティション列で表されます。Synapse は範囲によるパーティションをサポートしているため、パーティションは Synapse メタデータテーブルからパーティション列とパーティション範囲を抽出することによって実装されます。これらの範囲の値はスプリットの作成に使用されます。

パフォーマンス

列のサブセットを選択すると、クエリランタイムが大幅に遅くなります。同時実行が原因で、コネクタに著しいスロットリングが発生しています。

Athena Synapse コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。単純な述語と複雑な式はコネクタにプッシュダウンされるため、スキャンされるデータ量が減少し、クエリ実行のランタイムが短縮されます。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena Synapse コネクタは、これらの式を組み合わせて Synapse に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Synapse コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値:
  EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

パススルークエリ

Synapse コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Synapse でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Synapse 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

- Amazon QuickSight と Amazon Athena の横串検索を使用して、Microsoft Azure Synapse データベースに保存されているデータに関するダッシュボードとビジュアライゼーションを構築する方法の記事については、AWS Big Data Blog の「Perform multi-cloud analytics using Amazon QuickSight, Amazon Athena Federated Query, and Microsoft Azure Synapse」を参照してください。
- 最新の JDBC ドライバーのバージョン情報については、GitHub.com の Synapse コネクタ用の pom.xml ファイルを参照してください。
- このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Cloudera Hive コネクタ

Cloudera Hive 用の Amazon Athena コネクタを使用すると、Athena で Cloudera Hive Hadoop 分散に SQL クエリを実行できます。コネクタによって Athena SQL クエリは同等の HiveQL 構文に変換されます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。
- このコネクタを使用する際は、先に VPC とセキュリティグループをセットアップしておきます。詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。

- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

用語

以下に、Cloudera Hive コネクタに関連する用語を示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数により、Cloudera Hive コネクタの設定を行います。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
hive://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`HiveMuxCompositeHandler`
メタデータハンドラー	`HiveMuxMetadataHandler`
レコードハンドラー	`HiveMuxRecordHandler`

マルチプレックスハンドラーのパラメータ

パラメータ	説明
`$`*`catalog`*`_connection_string`	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが `myhivecatalog` の場合、環境変数の名前は `myhivecatalog_connection_string` になります。
`default`	必須。デフォルトの接続文字列。この文字列は、カタログが `lambda:${`*`AWS_LAMBDA_FUNCTION_NAME`*`}` の場合に使用されます。

以下に、2 つのデータベースインスタンス (デフォルト値の `hive1`、および `hive2`) をサポートする、Hive MUX Lambda 関数のプロパティ例を示します。

プロパティ	Value
`default`	`hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}`

プロパティ	Value
hive2_catalog1_connection_string	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive2_catalog2_connection_string	hive://jdbc:hive2://hive2:10000/default?UID=sample&PWD=sample

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

  ```
  {"username": "${username}", "password": "${password}"}
  ```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 ${Test/RDS/hive1} が含まれています。

```
hive://jdbc:hive2://hive1:10000/default?...&${Test/RDS/hive1}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
hive://jdbc:hive2://hive1:10000/default?...&UID=sample2&PWD=sample2&...
```

現在のところ、Cloudera Hive コネクタは、UID および PWD の JDBC プロパティを認識します。

単一接続ハンドラーの使用

以下の単一接続メタデータ、およびレコードハンドラーを使用すると、単一の Cloudera Hive インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	HiveCompositeHandler
メタデータハンドラー	HiveMetadataHandler
レコードハンドラー	HiveRecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

次は、Lambda 関数でサポートされている単一の Cloudera Hive インスタンス用のプロパティ例です。

プロパティ	Value
デフォルト	`hive://jdbc:hive2://hive1:10000/default?secret=${Test/RDS/hive1}`

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス
`spill_put_request_headers`	(オプション) スピルに使用される Amazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表では、JDBC、Cloudera Hive、および Arrow に対応させて、各データ型を示しています。

JDBC	Cloudera Hive	Arrow
ブール値	ブール値	Bit
整数	TINYINT	Tiny
ショート	SMALLINT	Smallint
整数	INT	Int

JDBC	Cloudera Hive	Arrow
Long	BIGINT	Bigint
フロート	float4	Float4
ダブル	float8	Float8
日付	date	DateDay
タイムスタンプ	timestamp	DateMilli
文字列	VARCHAR	Varchar
バイト	bytes	Varbinary
BigDecimal	10 進数	10 進数
配列	該当なし (注記を参照)	リスト

> **Note**
>
> 現在のところ、Cloudera Hive では、集計型 ARRAY、MAP、STRUCT、および UNIONTYPE はサポートされません。集計型の列は、SQL における VARCHAR 列のように扱われます。

パーティションと分割

パーティションは、コネクタを分割する方法を決定するために使用されます。Athena は `varchar` 型の合成列を作成し、コネクタが分割を生成できるようにするために、テーブルに対するパーティションのスキームを示します。コネクタは実際のテーブル定義を変更しません。

パフォーマンス

Cloudera Hive は静的パーティションをサポートしています。Athena Cloudera Hive コネクタは、これらのパーティションからデータを並列に取得できます。パーティション分散が一様で非常に大きなデータセットをクエリする場合は、静的パーティショニングの使用を強くお勧めします。Cloudera Hive コネクタは、同時実行によるスロットリングに強いです。

Athena Cloudera Hive コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、LIMIT 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

LIMIT N ステートメントにより、クエリによってスキャンされるデータが削減されます。LIMIT N プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena Cloudera Hive コネクタは、これらの式を組み合わせて Cloudera Hive に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Cloudera Hive コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

パススルークエリ

Cloudera Hive コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Cloudera Hive でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Cloudera Hive 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

JDBC ドライバーの最新バージョンの情報については、GitHub.com で Cloudera Hive コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Cloudera Impala コネクタ

Amazon Athena Cloudera Impala コネクタは、Athena が Cloudera Impala ディストリビューションで SQL クエリを実行することを可能にします。このコネクタは、Athena SQL クエリを同等の Impala 構文に変換します。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。
- このコネクタを使用する際は、先に VPC とセキュリティグループをセットアップしておきます。詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

用語

以下に、Cloudera Impala コネクタに関連する用語を示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数により、Cloudera Impala コネクタの設定を行います。

接続文字列

次の形式の JDBC 接続文字列を使用して、Impalaクラスターに接続します。

```
impala://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`ImpalaMuxCompositeHandler`
メタデータハンドラー	`ImpalaMuxMetadataHandler`
レコードハンドラー	`ImpalaMuxRecordHandler`

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。Athena カタログの Impala クラスター接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが `myimpalacatalog` の場合、環境変数の名前は `myimpalacatalog_connection_string` になります。
`default`	必須。デフォルトの接続文字列。この文字列は、カタログが `lambda:${ AWS_LAMBDA_FUNCTION_NAME }` の場合に使用されます。

以下に、2 つのデータベースインスタンス (デフォルトの `impala1`、および `impala2`) をサポートする Impala MUX Lambda 関数におけるプロパティ例を示します。

プロパティ	Value
default	impala://jdbc:impala://some.impala.host.name:21050/?${Test/impala1}
impala_catalog1_connection_string	impala://jdbc:impala://someother.impala.host.name:21050/?${Test/impala1}
impala_catalog2_connection_string	impala://jdbc:impala://another.impala.host.name:21050/?UID=sample&PWD=sample

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 ${Test/impala1host} が含まれています。

```
impala://jdbc:impala://Impala1host:21050/?...&${Test/impala1host}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
impala://jdbc:impala://Impala1host:21050/?...&UID=sample2&PWD=sample2&...
```

現在のところ Cloudera Impala は、UID および PWD の JDBC プロパティを認識します。

単一接続ハンドラーの使用

以下の単一接続メタデータとレコードハンドラーを使用すると、単一の Cloudera Impala インスタンスに接続することが可能です。

ハンドラーのタイプ	Class
複合ハンドラー	ImpalaCompositeHandler
メタデータハンドラー	ImpalaMetadataHandler
レコードハンドラー	ImpalaRecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

次は、Lambda 関数でサポートされている単一の Cloudera Impala インスタンスのためのプロパティ例です。

プロパティ	Value
default	impala://jdbc:impala://Impala1host:21050/?secret=${Test/impala1host}

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
spill_bucket	必須。スピルバケット名。
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表では、JDBC、Cloudera Impala、および Arrow と対応させて各データ型を示しています。

JDBC	Cloudera Impala	Arrow
ブール値	ブール値	Bit
整数	TINYINT	Tiny

JDBC	Cloudera Impala	Arrow
ショート	SMALLINT	Smallint
整数	INT	Int
Long	BIGINT	Bigint
フロート	float4	Float4
ダブル	float8	Float8
日付	date	DateDay
タイムスタンプ	timestamp	DateMilli
文字列	VARCHAR	Varchar
バイト	bytes	Varbinary
BigDecimal	10 進数	10 進数
配列	該当なし (注記を参照)	リスト

> **Note**
>
> 現在のところ、Cloudera Impala では集計型 (ARRAY、MAP、STRUCT、および UNIONTYPE) はサポートされません。集計型の列は、SQL における VARCHAR 列のように扱われます。

パーティションと分割

パーティションは、コネクタを分割する方法を決定するために使用されます。Athena は `varchar` 型の合成列を作成し、コネクタが分割を生成できるようにするために、テーブルに対するパーティションのスキームを示します。コネクタは実際のテーブル定義を変更しません。

パフォーマンス

Cloudera Impala では静的パーティションが利用できます。Athena Cloudera Impala コネクタは、これらのパーティションからデータを並列に取得できます。パーティション分散が一様で非常に大きな

データセットをクエリする場合は、静的パーティショニングの使用を強くお勧めします。Cloudera Impala コネクタは、同時実行によりスロットリングに強いです。

Athena Cloudera Impala コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、LIMIT 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

LIMIT N ステートメントにより、クエリによってスキャンされるデータが削減されます。LIMIT N プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena Cloudera Impala コネクタは、これらの式を組み合わせて Cloudera Impala に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Cloudera Impala コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

パススルークエリ

Cloudera Impala コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Cloudera Impala でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Cloudera Impala 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

JDBC ドライバーの最新バージョンの情報については、GitHub.com の pom.xml ファイルで、Cloudera Impala コネクタを確認してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena CloudWatch コネクタ

Amazon Athena CloudWatch コネクタは、Amazon Athena が CloudWatch とやり取りすることを可能にして、ログデータを SQL でクエリできるようにします。

このコネクタは、LogGroups をスキーマとして、および各 LogStream をテーブルとしてマップします。また、コネクタにより、LogGroup 内のすべての LogStream が含まれている特別な all_log_streams ビューもマップされます。このビューでは、各 LogStream を個別に検索するのではなく、LogGroup 内のすべてのログを一度にクエリできます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションの Lambda 環境変数により、CloudWatch コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。

コネクタでは、CloudWatch から Amazon Athena クエリフェデレーション SDK の `ThrottlingInvoker` コンストラクトを介してスロットリングされるイベントの処理用として、AIMD の輻輳 (ふくそう) 制御 もサポートしています。以下のオプションの環境変数のいずれかを設定することで、デフォルトのスロットリング動作を微調整できます。

- throttle_initial_delay_ms – 最初の輻輳イベントの後に適用される最初の呼び出し遅延。デフォルト値は 10 ミリ秒です。
- throttle_max_delay_ms – 呼び出し間の最大遅延。これを 1000 ミリ秒で除算することで、TPS を算出できます。デフォルト値は 1000 ミリ秒です。
- throttle_decrease_factor – Athena が呼び出しのレートを低減する要因。デフォルトは 0.5 です。
- throttle_increase_ms – Athena がコール遅延を低減させた際のレート。デフォルト値は 10 ミリ秒です。

データベースとテーブル

このコネクタは、LogGroups をスキーマ (つまりデータベース) として、また、各 LogStream をテーブルとしてマップします。また、コネクタにより、LogGroup 内のすべての LogStream が含まれている特別な `all_log_streams` ビューもマップされます。このビューでは、各 LogStream を個別に検索するのではなく、LogGroup 内のすべてのログを一度にクエリできます。

Athena CloudWatch コネクタによってマッピングされるすべてのテーブルには、次のスキーマが含まれています。このスキーマにより、CloudWatch Logs が提供するフィールドとのマッチングが行われます。

- log_stream – 対象の行を提供する LogStream の名前が含まれている `VARCHAR`。
- time – ログ行が生成されたエポックタイムを含む `INT64`。
- message – ログメッセージを含む `VARCHAR`。

例

以下の例では、指定された LogStream に対する `SELECT` クエリの実行方法を示しています

```
SELECT *
FROM "lambda:cloudwatch_connector_lambda_name"."log_group_path"."log_stream_name"
LIMIT 100
```

以下の例では、`all_log_streams`ビューを使用して、指定された LogGroup 内のすべての LogStreams に対してクエリを実行する方法を示しています。

```
SELECT *
FROM "lambda:cloudwatch_connector_lambda_name"."log_group_path"."all_log_streams"
LIMIT 100
```

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細は、athena-cloudwatch.yaml ファイルの `Policies` セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- CloudWatch Logs Read/Write – コネクタは、ログデータの読み取りと診断ログの書き込みのために、このアクセス許可を使用します。

パフォーマンス

Athena CloudWatch コネクタは、クエリが必要とするログストリームのスキャンを並列化することで、CloudWatch に対するクエリの最適化を試みます。フィルタで特定の期間を指定する場合、Lambda 関数内と CloudWatch Logs 内の両方で述語のプッシュダウンが実行されます。

最高のパフォーマンスを得るには、ロググループ名とログストリーム名には小文字のみを使用してください。大文字と小文字が混在すると、コネクタは大文字と小文字を区別しない検索を実行するため、計算量が多くなります。

パススルークエリ

CloudWatch コネクタは、CloudWatch Logs Insights クエリ構文を使用するパススルークエリをサポートします。CloudWatch Logs Insights の詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「CloudWatch Logs Insights でログデータを分析する」を参照してください。

CloudWatch でパススルークエリを作成するには、以下の構文を使用します。

```
SELECT * FROM TABLE(
        system.query(
            STARTTIME => 'start_time',
            ENDTIME => 'end_time',
            QUERYSTRING => 'query_string',
            LOGGROUPNAMES => 'log_group-names',
            LIMIT => 'max_number_of_results'
        ))
```

次の CloudWatch パススルークエリの例では、duration フィールドの値が 1000 でない場合にフィルターが適用されてます。

```
SELECT * FROM TABLE(
        system.query(
            STARTTIME => '1710918615308',
            ENDTIME => '1710918615972',
            QUERYSTRING => 'fields @duration | filter @duration != 1000',
            LOGGROUPNAMES => '/aws/lambda/cloudwatch-test-1',
            LIMIT => '2'
            ))
```

ライセンス情報

Amazon Athena CloudWatch コネクタプロジェクトは、Apache-2.0 ライセンスの下で使用許諾されています。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena CloudWatch Metrics コネクタ

Amazon Athena CloudWatch メトリクスコネクタを使用すると、Amazon Athena は SQL を使用して CloudWatch メトリクスデータをクエリできます。

Athena 自体から CloudWatch にクエリメトリクスを公開する方法については、「CloudWatch のメトリクスとイベントを使用したコストの管理とクエリのモニタリング」を参照してください。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションの Lambda 環境変数により、CloudWatch Metrics コネクタの設定を行います。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。

- spill_prefix – (オプション) 指定された athena-federation-spill という spill_bucket の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば a7e63k4b-8loc-40db-a2a1-4d0en2cd8331) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) True に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は False です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。

コネクタでは、CloudWatch から Amazon Athena クエリフェデレーション SDK の ThrottlingInvoker コンストラクトを介してスロットリングされるイベントの処理用として、AIMD の輻輳 (ふくそう) 制御 もサポートしています。以下のオプションの環境変数のいずれかを設定することで、デフォルトのスロットリング動作を微調整できます。

- throttle_initial_delay_ms – 最初の輻輳イベントの後に適用される最初の呼び出し遅延。デフォルト値は 10 ミリ秒です。
- throttle_max_delay_ms – 呼び出し間の最大遅延。これを 1000 ミリ秒で除算することで、TPS を算出できます。デフォルト値は 1000 ミリ秒です。
- throttle_decrease_factor – Athena が呼び出しのレートを低減する要因。デフォルトは 0.5 です。
- throttle_increase_ms – Athena がコール遅延を低減させた際のレート。デフォルト値は 10 ミリ秒です。

データベースとテーブル

Athena CloudWatch Metrics コネクタは、名前空間、ディメンション、メトリクス、およびメトリクスの値を、default という単一のスキーマ内で 2 つのテーブルにマッピングします。

メトリクステーブル

metrics テーブルは、名前空間、セット、名前の組み合わせによって一意に定義された、使用可能なメトリクスで構成されています。この metrics テーブルには次の列が含まれます。

- namespacee – VARCHAR は名前空間を含みます。
- metric_name – VARCHAR はメトリクス名を含みます。
- ディメンション – dim_name（VARCHAR）および dim_value（VARCHAR）から構成される STRUCT オブジェクトの LIST。
- statistic – メトリクスで使用が可能な、VARCH 統計情報の LIST (例: p90、AVERAGE、...)。

metric_samples テーブル

metric_samples テーブルには、metrics テーブルで使用可能な各メトリクスのサンプルが含まれています。この metric_samples テーブルには次の列が含まれます。

- namespace – 名前空間が含まれている VARCHAR。
- metric_name – メトリクス名が含まれている VARCHAR。
- ディメンション – dim_name（VARCHAR）および dim_value（VARCHAR）から構成される STRUCT オブジェクトの LIST。
- dim_name – 単一のディメンション名により簡単にフィルタリングできる、VARCHAR のコンビニエンスフィールド。
- dim_value – 単一のディメンション値により簡単にフィルタリングできる、VARCHAR のコンビニエンスフィールド。
- period – メトリクスの「期間」を、秒単位 (60 秒間メトリクスなど) で表す INT フィールド。
- timestamp – メトリクスがサンプリングを行うエポックタイムを、秒単位で表す BIGINT フィールド。
- value – サンプルされた値を含む FLOAT8 フィールド。
- statistic – サンプル値の統計タイプ (例: AVERAGE、p90) が含まれている VARCHAR 。

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-cloudwatch-metrics.yaml ファイルの Policies セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内のロケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- CloudWatch Metrics ReadOnly – コネクタは、メトリクスのデータをクエリするために、このアクセス許可を使用します。
- CloudWatch Logs Write – コネクタは、このアクセスを使用して診断ログへの書き込みを行います。

パフォーマンス

Athena CloudWatch Metrics コネクタは、クエリに必要なログストリームに対し並列化したスキャンを行うことで、CloudWatch Metrics へのクエリを最適化しようとします。特定の期間や、メトリクス、名前空間、ディメンションフィルタに対しては、Lambda 関数内と CloudWatch Logs 内の両方で述語のプッシュダウンが実行されます。

ライセンス情報

Amazon Athena CloudWatch Metrics コネクタプロジェクトは、Apache-2.0 ライセンスの下で使用許諾されています。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena AWS CMDB コネクタ

Amazon Athena AWS CMDB コネクタは、Amazon Athena と AWS のさまざまなサービスとの通信を可能にし、それらに対し SQL によりクエリできるようにします。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションの Lambda 環境変数により、AWS CMDB コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- default_ec2_image_owner – (オプション) 設定されている場合、Amazon マシンイメージ (AMI) をフィルタリングする、デフォルトの Amazon EC2 イメージ所有者を管理します。この値が設定されておらず、EC2 イメージテーブルに対するクエリが所有者に対するフィルタを含まない場合は、すべてのパブリックイメージが結果に含まれます。

データベースとテーブル

Athena AWS CMDB コネクタは、次のデータベースとテーブルが、AWS のリソースインベントリーをクエリできるようにします。各テーブルで使用できる列の詳細は、Athena コンソールまたは API から、`DESCRIBE` *`database`*`.`*`table`* ステートメントを実行すると確認できます。

- ec2 – このデータベースには、以下をはじめとした Amazon EC2 の関連リソースが含まれています。

- ebs_volumes – Amazon EBS ボリュームの詳細が含まれます。
- ec2_instances – EC2 インスタンスの詳細が含まれます。
- ec2_images – EC2 インスタンスイメージの詳細が含まれます。
- routing_tables – VPC ルートテーブルの詳細が含まれます。
- security_groups – セキュリティグループの詳細が含まれます。
- subnets – VPC サブネットの詳細が含まれます。
- vpcs – VPC の詳細が含まれます。

- emr – このデータベースは、以下をはじめとした Amazon EMR 関連のリソースを含みます。

- emr_clusters – EMR クラスターの詳細が含まれます。

- rds – このデータベースは、以下をはじめとした Amazon RDS 関連のリソースを含みます。

- rds_instances – RDS インスタンスの詳細が含まれます。

- s3 – このデータベースは、以下をはじめとした RDS 関連のリソースを含みます。

- buckets – Amazon S3 バケットの詳細が含まれます。
- objects – Amazon S3 オブジェクトの詳細を含みますが、オブジェクトの内容は含みません。

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細は、athena-aws-cmdb.yaml ファイルの `Policies` セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- S3 List – コネクタは、このアクセス許可を使用して Amazon S3 のバケットとオブジェクトをリストします。

- EC2 Describe – コネクタは、Amazon EC2 インスタンス、セキュリティグループ、VPC、Amazon EBS ボリュームなどのリソースを詳細表示する際に、このアクセス許可を使用します。
- EMR Describe / List – コネクタは、EMR クラスターの詳細を表示する際に、このアクセス許可を使用します。
- RDS Describe – コネクタは、RDS インスタンスの詳細を表示する際に、このアクセス許可を使用します。

パフォーマンス

現在のところ、Athena AWS CMDB コネクタでは、並列スキャンはサポートされません。述語のプッシュダウンは Lambda 関数内で実行されます。可能な場合、部分述語はクエリ対象のサービスにプッシュされます。例えば、特定の Amazon EC2 インスタンスの詳細についてのクエリでは、目的の詳細表示オペレーションを実行するために、特定のインスタンス ID を使用して EC2 API を呼び出します。

ライセンス情報

Amazon Athena AWS CMDB コネクタプロジェクトは、Apache-2.0 ライセンスの下で使用許諾されています。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena IBM Db2 コネクタ

Db2 用 Amazon Athena コネクタは、Amazon Athena で JDBC を使用して IBM Db2 データベースの SQL クエリを実行できるようにします。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。
- このコネクタを使用する際は、先に VPC とセキュリティグループをセットアップしておきます。詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- フィルター条件における日付とタイムスタンプのデータ型は、適切なデータ型にキャストする必要があります。

用語

Db2 コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して Db2 コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
dbtwo://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`Db2MuxCompositeHandler`
メタデータハンドラー	`Db2MuxMetadataHandler`
レコードハンドラー	`Db2MuxRecordHandler`

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが `mydbtwocatalog` の場合、環境変数の名前は `mydbtwocatalog_connection_string` になります。
`default`	必須。デフォルトの接続文字列。この文字列は、カタログが `lambda:${ AWS_LAMBDA_FUNCTION_NAME }` の場合に使用されます。

dbtwo1 (デフォルト) と dbtwo2 の 2 つのデータベースインスタンスをサポートする Db2 MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	dbtwo://jdbc:db2://dbtwo1.hostname:port/ *database_name* :${*secret1_name* }
dbtwo_catalog1_connection_string	dbtwo://jdbc:db2://dbtwo1. hostname:port/ *database_name* :${*secret1_name* }
dbtwo_catalog2_connection_string	dbtwo://jdbc:db2://dbtwo2. hostname:port/ *database_name* :${*secret2_name* }

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 ${secret_name} が含まれています。

```
dbtwo://jdbc:db2://hostname:port/database_name:${secret_name}
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
dbtwo://jdbc:db2://hostname:port/database_name:user=user_name;password=password;
```

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Db2 インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	Db2CompositeHandler
メタデータハンドラー	Db2MetadataHandler
レコードハンドラー	Db2RecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の Db2 インスタンス用のプロパティ例を次に示します。

プロパティ	Value
default	dbtwo://jdbc:db2://hostname:port/*database_name* :${secret_name}

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
spill_bucket	必須。スピルバケット名。
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC と Arrow に対応するデータ型を示します。

Db2	Arrow
CHAR	VARCHAR
VARCHAR	VARCHAR
DATE	DATEDAY
TIME	VARCHAR
TIMESTAMP	DATEMILLI

Db2	Arrow
DATETIME	DATEMILLI
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	DECIMAL
REAL	FLOAT8
DOUBLE	FLOAT8
DECFLOAT	FLOAT8

パーティションと分割

パーティションは、1 つまたは複数のタイプ `varchar` のパーティション列で表されます。Db2 コネクタは、次の組織スキームを使用してパーティションを作成します。

- ハッシュによる分散
- 範囲によるパーティション
- ディメンションによる整理

コネクタは、1 つ以上の Db2 メタデータテーブルから、パーティションの数や列名などのパーティションの詳細を取得します。スプリットは、特定されたパーティションの数に基づいて作成されます。

パフォーマンス

Athena Db2 コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、`LIMIT` 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

LIMIT N ステートメントにより、クエリによってスキャンされるデータが削減されます。LIMIT N プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena Db2 コネクタは、これらの式を組み合わせて Db2 に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Db2 コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

パススルークエリ

Db2 コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Db2 でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
```

```
        ))
```

以下のクエリ例は、Db2 内のデータソースにクエリをプッシュダウンします。クエリは `customer` テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の Db2 コネクタ用の「pom.xml」ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena IBM Db2 AS/400 (Db2 iSeries) コネクタ

Db2 AS/400 用の Amazon Athena コネクタは、Amazon Athena が JDBC を使用して IBM Db2 AS/400 (Db2 iSeries) データベースで SQL クエリを実行することを可能にします。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。
- このコネクタを使用する際は、先に VPC とセキュリティグループをセットアップしておきます。詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。

- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- フィルター条件における日付とタイムスタンプのデータ型は、適切なデータ型にキャストする必要があります。

用語

以下は、Db2 AS/400 コネクタに関連する用語です。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して、Db2 AS/400 コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
db2as400://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	Db2MuxCompositeHandler
メタデータハンドラー	Db2MuxMetadataHandler
レコードハンドラー	Db2MuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが mydb2as400catalog の場合、環境変数の名前は mydb2as400catalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

db2as4001 (デフォルト) と db2as4002 の 2 つのデータベースインスタンスをサポートする Db2 MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	db2as400://jdbc:as400:// *<ip_address>* ;*<properties>* ;:${*<secret name>*};
db2as400_catalog1_connection_string	db2as400://jdbc:as400://db2as4001. hostname/:${ *secret1_name* }
db2as400_catalog2_connection_string	db2as400://jdbc:as400://db2as4002. hostname/:${ *secret2_name* }
db2as400_catalog3_connection_string	db2as400://jdbc:as400:// *<ip_address>* ;user=*<username>* ;password= *<password>* ;*<properties>* ;

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 ${secret_name} が含まれています。

```
db2as400://jdbc:as400://<ip_address>;<properties>;:${<secret_name>};
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
db2as400://jdbc:as400://<ip_address>;user=<username>;password=<password>;<properties>;
```

単一接続ハンドラーの使用

以下の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Db2 AS/400 インスタンスに接続することができます。

ハンドラーのタイプ	Class
複合ハンドラー	Db2CompositeHandler
メタデータハンドラー	Db2MetadataHandler
レコードハンドラー	Db2RecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

以下は、Lambda 関数がサポートする単一の Db2 AS/400 用のプロパティ例です。

プロパティ	Value
default	db2as400://jdbc:as400:// *<ip_address>* ;*<properties>* ;: ${*<secret_name>* };

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
spill_bucket	必須。スピルバケット名。
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC と Apache Arrow に対応するデータ型を示します。

Db2 AS/400	Arrow
CHAR	VARCHAR

Db2 AS/400	Arrow
VARCHAR	VARCHAR
DATE	DATEDAY
TIME	VARCHAR
TIMESTAMF	DATEMILLI
DATETIME	DATEMILLI
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	DECIMAL
REAL	FLOAT8
DOUBLE	FLOAT8
DECFLOAT	FLOAT8

パーティションと分割

パーティションは、1 つまたは複数のタイプ `varchar` のパーティション列で表されます。Db2 AS/400 コネクタは、以下の組織スキームを使用してパーティションを作成します。

- ハッシュによる分散
- 範囲によるパーティション
- ディメンションによる整理

コネクタは、1 つ、または複数の Db2 AS/400 メタデータテーブルから、パーティションの数や列名などのパーティションの詳細情報を取得します。スプリットは、特定されたパーティションの数に基づいて作成されます。

パフォーマンス

パフォーマンスを向上させるには、以下の例にあるように、述語のプッシュダウンを使用して Athena からクエリを実行してください。

```
SELECT * FROM "lambda:<LAMBDA_NAME>"."<SCHEMA_NAME>"."<TABLE_NAME>"
 WHERE integercol = 2147483647
```

```
SELECT * FROM "lambda: <LAMBDA_NAME>"."<SCHEMA_NAME>"."<TABLE_NAME>"
 WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

パススルークエリ

Db2 AS/400 コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Db2 AS/400 でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Db2 AS/400 内のデータソースにクエリをプッシュダウンします。クエリは `customer` テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーバージョンの情報については、GitHub.com で Db2 AS/400 コネクタ用の「pom.xml」ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena DocumentDB コネクタ

Amazon Athena DocumentDB コネクタは、Athena と DocumentDB インスタンスとの通信を可能にします。これにより、DocumentDB データを SQL でクエリできるようになります。また、このコネクタは、MongoDB と互換性のある任意のエンドポイントでも機能します。

従来のリレーショナルデータストアとは異なり、Amazon DocumentDB コレクションには設定されたスキーマがありません。DocumentDB にはメタデータストアはありません。DocumentDB コレクションの各エントリには、さまざまな種類のフィールドとデータ型を含めることが可能です。

DocumentDB コネクタは、テーブルのスキーマ情報を生成するための 2 つのメカニズム (基本スキーマ推論と AWS Glue Data Catalog メタデータ) をサポートしています。

デフォルトでは、スキーマ推論が選択されます。このオプションは、コレクション内で少数のドキュメントをスキャンした上で、すべてのフィールドを結合して、重複しないデータ型を持つフィールドのみを選択的に使用します。このオプションは、ほとんどのエントリが同一であるコレクションに適しています。

より多くのデータ型を持つコレクションに対して、このコネクタでは、AWS Glue Data Catalog からのメタデータの取得をサポートします。DocumentDB データベースとコレクションの名前と一致する AWS Glue データベースおよびテーブルを認識した場合、このコネクタは、対応する AWS Glue テーブルからスキーマ情報を取得します。AWS Glue テーブルを作成する際には、DocumentDB コレクションからのアクセス対象となる、すべてのフィールドのスーパーセットを作成することをお勧めします。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または

「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションに示した Lambda 環境変数により、DocumentDB コネクタを設定できます。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- disable_glue – (オプション) これが存在し、true に設定されている場合、コネクタは AWS Glue からの補足メタデータ取得は試みません。
- glue_catalog – (オプション) クロスアカウントの AWS Glue カタログを指定するために、このオプションを使用します。デフォルトでは、コネクタは自身の AWS Glue アカウントからメタデータを取得しようとします。
- default_docdb – これが存在する場合は、カタログ固有の環境変数が存在しない場合に使用する、DocumentDB の接続文字列を指定します。
- disable_projection_and_casing – (オプション) プロジェクションおよび大文字と小文字の区別を無効にします。大文字と小文字が区別される列名を使用する Amazon DocumentDB テーブルをクエ

リする場合に使用します。disable_projection_and_casing パラメータは、次の値を使用して大文字と小文字の区別、および列のマッピングに関する動作を指定します。

- false – これは、デフォルトの設定です。プロジェクションが有効になっていて、コネクタはすべての列の名前が小文字であると想定します。
- true – プロジェクションおよび大文字と小文字の区別を無効にします。disable_projection_and_casing パメータを使用する場合は、以下の点に注意してください。
  - このパラメータを使用すると、帯域幅の使用量が増加する可能性があります。さらに、Lambda 関数がデータソースと同じ AWS リージョン にない場合、帯域幅の使用量が増えるため、標準の AWS クロスリージョン転送コストが高くなります。クロスリージョン転送コストの詳細については、「AWS Partner Network ブログ」の「サーバーアーキテクチャおよびサーバーレスアーキテクチャの AWS データ転送料金」を参照してください。
  - 転送されるバイト数が増え、またバイト数が多いと逆シリアル化により時間がかかるため、全体のレイテンシが長くなる可能性があります。
- enable_case_insensitive_match – (オプション) true の場合、Amazon DocumentDB 内のスキーマ名とテーブル名に対して大文字と小文字を区別する検索を実行します。デフォルト: false。クエリに大文字のスキーマ名またはテーブル名が含まれる場合に使用します。

接続文字列の指定

コネクタで使用する DocumentDB インスタンスの DocumentDB 接続の詳細を定義するために、複数のプロパティを指定できます。そのためには、Athena で使用するカタログ名と対応した Lambda の環境変数を設定します。たとえば、2 つの異なる DocumentDB インスタンスに対し、Athena から次のクエリを実行するとします。

```
SELECT * FROM "docdb_instance_1".database.table
```

```
SELECT * FROM "docdb_instance_2".database.table
```

これら 2 つの SQL ステートメントを使用する際には、Lambda 関数 (docdb_instance_1 および docdb_instance_2) に、2 つの環境変数を追加しておく必要があります。それぞれの値は、次の形式の DocumentDB 接続文字列にする必要があります。

```
mongodb://:@:/?ssl=true&ssl_ca_certs=rds-combined-ca-bundle.pem&replicaSet=rs0
```

シークレットの使用

オプションで、接続文字列の詳細の一部またはすべての値について、AWS Secrets Manager を使用できます。Athena 横串検索機能を Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

Secrets Manager が提供するシークレット名を接続文字列に入れるために ${my_secret} 構文を使用する場合、コネクタは ${my_secret} を Secrets Manager のプレーンテキスト値にそのまま置き換えます。シークレットは、*<username>*:*<password>* 値付きのプレーンテキストのシークレットとして保存する必要があります。{username:*<username>*,password:*<password>*} として保存されたシークレットは接続文字列に正しく渡されません。

シークレットは接続文字列全体に使用することもでき、ユーザー名とパスワードはシークレット内で定義できます。

たとえば、docdb_instance_1 で Lambda 環境変数を次の値に設定した場合を考えます。

```
mongodb://${docdb_instance_1_creds}@myhostname.com:123/?ssl=true&ssl_ca_certs=rds-combined-ca-bundle.pem&replicaSet=rs0
```

Athena Query Federation SDK は、自動的に docdb_instance_1_creds という名前のシークレットを Secrets Manager から取得しよう試み、取得した値は ${docdb_instance_1_creds} の場所に挿入します。接続文字列の中で、${ } 文字の組み合わせにより囲まれている任意の部分は、Secrets Manager から提供されたシークレットとして認識されます コネクタにより Secrets Manager 内で検出されないシークレット名を指定した場合、コネクタはテキストを置き換えません。

AWS Glueでのデータベースとテーブルのセットアップ

コネクタに組み込まれているスキーマ推論機能がスキャンするドキュメント数には制限があり、また、データ型のサブセットのみをサポートするため、メタデータ用としては AWS Glue の使用が適しています。

Amazon DocumentDB で使用するために AWS Glue テーブルを有効にするには、DocumentDB データベース向けの AWS Glue データベーステーブルと、補足メタデータを提供する先のコレクションを用意しておく必要があります。

補足メタデータのために AWS Glue を使用するには

1. AWS Glue コンソールでテーブルとデータベースを編集する際には、以下のテーブルプロパティを追加します。

   - docdb-metadata-flag – このプロパティは、コネクタが補足メタデータ用としてテーブルを使用できることを、DocumentDB コネクタに対し示します。`docdb-metadata-flag` プロパティがテーブルプロパティのリスト内に存在するのであれば、この `docdb-metadata-flag` に任意の値を指定することが可能です。

2. (オプション) sourceTable テーブルプロパティを追加します。このプロパティは、Amazon DocumentDB 内にあるソーステーブル名を定義します。AWS Glue テーブルの命名規則が原因で、Amazon DocumentDB テーブルと同じ名前で AWS Glue テーブルが作成できない場合にこのプロパティを使用します。たとえば、AWS Glue テーブルの名前には大文字が使用できませんが、Amazon DocumentDB テーブル名としては使用が可能です。

3. (オプション) columnMapping テーブルプロパティを正しく追加します。このプロパティは列名のマッピングを定義します。AWS Glue 列の命名規則が原因で、Amazon DocumentDB テーブルと同じ名前の列名を持つ AWS Glue テーブルを作成できない場合は、このプロパティを使用します。Amazon DocumentDB の列名には大文字を使用できますが、AWS Glue 列名には使用できないため、このプロパティは有用です。

   `columnMapping` プロパティの値は、`col1=Col1,col2=Col2` 形式のマッピングのセットであることが想定されます。

   > **Note**
   >
   > 列マッピングは最上位の列名にのみ適用され、ネストされたフィールドには適用されません。

   AWS GluecolumnMapping テーブルプロパティを正しく追加した後に、`disable_projection_and_casing` Lambda 環境変数を削除できます。

4. このドキュメントに記載されているとおりに、AWS Glue 用として適切なデータ型を使用しているか確認してください。

サポートされるデータ型

このセクションでは、DocumentDB コネクタがスキーマ推論に使用するデータ型と、AWS Glue メタデータが使用されている場合のデータ型を一覧で示します。

スキーマ推論のデータ型

DocumentDB コネクタのスキーマ推論機能は、値を次のいずれかのデータ型に属するものとして推測しようとします。この表では、Amazon DocumentDB、Java、および Apache Arrow に対応させてデータ型を示しています。

Apache Arrow	Java または DocDB
VARCHAR	文字列
INT	整数
BIGINT	Long
BIT	ブール値
FLOAT4	浮動小数点数
FLOAT8	ダブル
TIMESTAMPSEC	日付
VARCHAR	ObjectId
LIST	リスト
STRUCT	ドキュメント

AWS Glue データ型

補足的メタデータ用として AWS Glue を使用する場合は、次のデータ型を設定できます。この表では、AWS Glue と Apache Arrow に対応するデータ型を示しています。

AWS Glue	Apache Arrow
整数	INT
bigint	BIGINT
double	FLOAT8
フロート	FLOAT4
ブール値	BIT
バイナリ	VARBINARY
string	VARCHAR
リスト	LIST
Struct	STRUCT

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-docdb.yaml ファイルの `Policies` セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- AWS Glue Data Catalog – DocumentDB コネクタには、スキーマ情報を取得するために、AWS Glue Data Catalog からの読み込み専用アクセスが必要です。
- CloudWatch Logs – コネクタは、ログを保存するために CloudWatch Logs にアクセスする必要があります。
- AWS Secrets Manager 読み込みアクセス – DocumentDB エンドポイントの詳細を Secrets Manager に保存する場合は、それらのシークレットに対するアクセス権をコネクタに付与する必要があります。

- VPC アクセス – コネクタには、VPC に接続して DocumentDB インスタンスと通信するために、その VPC に対しインターフェイスをアタッチおよびデタッチする機能が必要です。

パフォーマンス

Athena Amazon DocumentDB コネクタは現在、並列スキャンをサポートしていませんが、DocumentDB クエリの一部として述語をプッシュダウンしようとし、DocumentDB コレクションへのインデックスに対する述語は、スキャンされるデータが大幅に少なくなります。

Lambda 関数は、クエリがスキャンするデータを削減するために、射影プッシュダウンを実行します。ただし、列のサブセットを選択した場合は、クエリの実行時間が長くなることがあります。`LIMIT` 句はスキャンされるデータ量を減らしますが、述語を提供しない場合、`LIMIT` 句を含む `SELECT` クエリは少なくとも 16 MB のデータをスキャンすることを想定する必要があります。

パススルークエリ

Athena Amazon DocumentDB コネクタはパススルークエリをサポートしており、NoSQL ベースです。Amazon DocumentDB のクエリについては、「Amazon DocumentDB 開発者ガイド」の「クエリ」を参照してください。

Amazon DocumentDB でパススルークエリを使用するには、以下の構文を使用します。

```
SELECT * FROM TABLE(
        system.query(
            database => 'database_name',
            collection => 'collection_name',
            filter => '{query_syntax}'
        ))
```

以下の例は、TPCDS コレクション内の `example` データベースをクエリし、「Bill of Rights」というタイトルのすべての書籍をフィルタリングします。

```
SELECT * FROM TABLE(
        system.query(
            database => 'example',
            collection => 'tpcds',
            filter => '{title: "Bill of Rights"}'
        ))
```

追加リソース

- Amazon Athena の横串検索を使用して、MongoDB データベースを Amazon QuickSight に接続し、ダッシュボードとビジュアライゼーションを構築する方法の記事については、「AWS Big Data Blog」の「Visualize MongoDB data from Amazon QuickSight using Amazon Athena Federated Query」を参照してください。
- このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena DynamoDB コネクタ

Amazon Athena DynamoDB コネクタは、Amazon Athena が DynamoDB とやり取りすることを可能にして、テーブルを SQL でクエリできるようにします。INSERT INTO などの書き込み操作はサポートされていません。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションの Lambda 環境変数により、DynamoDB コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON で工

ンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- disable_glue – (オプション) これが存在し、true に設定されている場合、コネクタは AWS Glue からの補足メタデータ取得は試みません。
- glue_catalog – (オプション) クロスアカウントの AWS Glue カタログを指定するために、このオプションを使用します。デフォルトでは、コネクタは自身の AWS Glue アカウントからメタデータを取得しようとします。
- disable_projection_and_casing – (オプション) プロジェクションおよび大文字と小文字の区別を無効にします。列名に大文字と小文字の区別がある DynamoDB テーブルをクエリしたいものの、AWS Glue テーブルで `columnMapping` プロパティを指定したくない場合に使用します。

  `disable_projection_and_casing` パラメータは、次の値を使用して大文字と小文字の区別、および列のマッピングに関する動作を指定します。

  - auto – これまでサポートされていなかったタイプが検出され、列名のマッピングがテーブルに設定されていない場合、プロジェクション、および大文字と小文の区別を無効にします。これはデフォルトの設定です。
  - always – プロジェクション、および大文字と小文字の区別を無条件に無効にします。これは、DynamoDB の列名で大文字と小文字を区別しているものの、列名のマッピングを指定したくない場合に便利です。

  `disable_projection_and_casing` パメータを使用する場合は、以下の点に注意してください。

  - このパラメータを使用すると、帯域幅の使用量が増加する可能性があります。さらに、Lambda 関数がデータソースと同じ AWS リージョン にない場合、帯域幅の使用量が増えるため、標準の AWS クロスリージョン転送コストが高くなります。クロスリージョン転送コストの詳細に

ついては、「AWS Partner Network ブログ」の「サーバーアーキテクチャおよびサーバーレスアーキテクチャの AWS データ転送料金」を参照してください。

- 転送されるバイト数が増え、またバイト数が多いと逆シリアル化により時間がかかるため、全体のレイテンシが長くなる可能性があります。

AWS Glueでのデータベースとテーブルのセットアップ

コネクタに組み込まれたスキーマ推論機能には制限があるため、メタデータ用としては AWS Glue の使用が適しています。これを行うには、AWS Glue にデータベースとテーブルが必要です。DynamoDB で使用できるようにするには、そのプロパティを編集する必要があります。

AWS Glue コンソールでデータベースプロパティを編集するには

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Databases](データベース) タブ。

   [Databases] (データベース) ページでは、既存のデータベースを編集するか、[Add database] (データベースを追加) を選択してデータベースを作成できます。
3. データベースのリストで、編集するデータベース用リンクを選択します。
4. [編集] を選択します。
5. [Update a database] (データベースの更新) ページの [Location] (場所) に、文字列 **`dynamo-db-flag`** を追加します。このキーワードは、Athena DynamoDB コネクタが補足メタデータに使用しているテーブルがデータベースに含まれており、`default` 以外の AWS Glue データベースに必要であることを示します。`dynamo-db-flag` プロパティは、多数のデータベースを持つアカウントのデータベースを除外する場合に便利です。

AWS Glue コンソールでテーブルプロパティを編集するには

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Tables] (テーブル) タブを選択します。

   [テーブル] タブで、既存のテーブルを編集します。テーブルを手動またはクローラで追加する方法については、「AWS Glue デベロッパーガイド」の「AWS Glue コンソールでのテーブルの使用」を参照してください。
3. テーブルのリストで、編集するテーブルのリンクを選択します。

4. [Actions] (アクション)、[Edit table] (テーブルの編集) の順に選択します。
5. [Edit table] (テーブルの編集) ページの [Table properties] (テーブルプロパティ) セクションで、必要に応じて以下のテーブルプロパティを追加します。AWS Glue DynamoDB クローラーを使用する場合、これらのプロパティは自動的に設定されます。

   - dynamodb – テーブルを補足メタデータとして使用できることを Athena DynamoDB コネクタに示す文字列。[Classification] (分類) (完全一致) と呼ばれるフィールドにある、テーブルプロパティの dynamodb 文字列を入力します。

     > **Note**
     >
     > AWS Glue コンソールのテーブル作成プロセスの一部である [テーブルプロパティの設定] ページには、[分類] フィールドを含む [データ形式] セクションがあります。ここでは dynamodb を入力も選択もできません。代わりに、テーブルを作成した後、手順に従ってテーブルを編集し、[テーブルプロパティ] セクションにキー値のペアとして `classification` および dynamodb を入力します。

   - sourceTable – DynamoDB 内にあるソーステーブル名を定義するオプションのテーブルプロパティ。AWS Glue テーブルの命名規則が原因で、DynamoDB テーブルと同じ名前で AWS Glue テーブルが作成できない場合に使用します。例えば、AWS Glue テーブルの名前に大文字は許可されていませんが、DynamoDB テーブル名としては使用が可能です。
   - columnMapping – カラム名のマッピングを定義するオプションのテーブルプロパティ。AWS Glue 列の命名規則が原因で、DynamoDB テーブルと同じ名前の列を持つ AWS Glue テーブルを作成できない場合に使用します。例えば、AWS Glue では列名として大文字は許可されていませんが、DynamoDB の列名では使用が可能です。このプロパティ値の形式は、col1=Col1、col2=Col2 のようにします。列マッピングは最上位の列名にのみ適用され、ネストされたフィールドには適用されないことに注意してください。
   - defaultTimeZone – 明示的なタイムゾーンが含まれない `date` 値、および `datetime` 値に適用される、オプションのテーブルプロパティ。データソースのデフォルトタイムゾーンと Athena セッションのタイムゾーンの間に不一致が生じないようにするためには、この値を設定することが適切です。
   - datetimeFormatMapping – AWS Glue `date` または `timestamp` データ型の列のデータを解析する際に、`date` または `datetime` 形式が使用されること指定する、オプションのテーブルプロパティ。このプロパティが指定されていない場合、コネクタは ISO-8601 形式が使用されるものと推定します。コネクタが `date` または `datetime` 形式を推定できない、あるいは未加工の文字列を解析できない場合、この値は結果から除外されます。

datetimeFormatMapping 値は、col1=someformat1,col2=someformat2 形式である必要があります。以下に形式の例をいくつか示します。

```
yyyyMMdd'T'HHmmss
ddMMyyyy'T'HH:mm:ss
```

列にタイムゾーンのない date または datetime 値が含まれており、その列を WHERE 句で使用したい場合には、対象の列で datetimeFormatMapping プロパティを設定します。

6. 手動で列を定義する場合は、適切なデータ型を使用するように注意してください。クローラーを使用している場合は、クローラーが検出した列とタイプを確認します。

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-dynamodb.yaml ファイルの Policies セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内のロケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- AWS Glue Data Catalog – DynamoDB コネクタには、スキーマ情報を取得するために、AWS Glue Data Catalog に対する読み込み専用アクセスが必要です。
- CloudWatch Logs – コネクタは、ログを保存するために CloudWatch Logs にアクセスする必要があります。
- DynamoDB 読み込みアクセス – このコネクタでは、DescribeTable、ListSchemas、ListTables、Query、および Scan のAPI オペレーションを使用します。

パフォーマンス

Athena DynamoDB コネクタは並列スキャンをサポートしており、DynamoDB クエリの一部として述語のプッシュダウンを試みます。X が異なる値を持つハッシュキー述語を使用すると、DynamoDB に対する X クエリ呼び出しが発生します。他のすべての述語シナリオでは、スキャン呼び出しの数が Y となります。この際 Y は、テーブルのサイズとプロビジョニングされたスループットに基づい

て、ヒューリスティックに決定されます。ただし、列のサブセットを選択すると、クエリのランタイムが長くなる場合があります。

LIMIT 句と単純な述語がプッシュダウンされるため、スキャンされるデータの量が減少し、クエリ実行のランタイムの短縮につながります。

LIMIT 句

LIMIT N ステートメントにより、クエリによってスキャンされるデータが削減されます。LIMIT N プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。機能を強化し、スキャンされるデータの量を減らすために、Athena DynamoDB コネクタはこれらの式を組み合わせて、DynamoDB に直接プッシュできます。

次の Athena DynamoDB コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND
- 等値:
  EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10 and col_b < 10
LIMIT 10
```

DynamoDB などのフェデレーテッドクエリのパフォーマンスを向上させるために述語プッシュダウンを使用する方法の記事については、「AWS Big Data Blog」の「Improve federated queries with predicate pushdown in Amazon Athena」を参照してください。

パススルークエリ

DynamoDB コネクタはパススルークエリをサポートし、PartiQL 構文を使用します。DynamoDB GetItem API オペレーションはサポートされません。PartiQL を使用した DynamoDB のクエリに関

する詳細については、「Amazon DynamoDB デベロッパーガイド」の「PartiQL select statements for DynamoDB」を参照してください。

DynamoDB でパススルークエリを使用するには、以下の構文を使用します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query_string'
        ))
```

以下の DynamoDB パススルークエリ例は、PartiQL を使用して、2022 年 12 月 24 日より後の DateWatched プロパティを持つ Fire TV Stick デバイスのリストを返します。

```
SELECT * FROM TABLE(
        system.query(
           query => 'SELECT Devices
                       FROM WatchList
                       WHERE Devices.FireStick.DateWatched[0] > '12/24/22''
        ))
```

トラブルシューティング

ソートキー列上の複数フィルター

エラーメッセージ: KeyConditionExpression にはキーごとに 1 つの条件のみを含める必要があります

原因: この問題は、Athena エンジンバージョン 3 で、DynamoDB ソートキー列上に下限フィルターと上限フィルターの両方があるクエリで発生する可能性があります。DynamoDB はソートキー上で複数のフィルター条件をサポートしていないため、両方の条件が適用されたクエリをコネクタがプッシュダウンしようとするとエラーが発生します。

解決策: コネクタをバージョン 2023.11.1 以降に更新します。コネクタを更新する手順については、「データソースコネクタの更新」を参照してください。

コスト

コネクタの使用料金は、基礎として使用されている AWS リソースによって異なります。スキャンを使用するクエリでは、大量のリードキャパシティーユニット (RCU) を利用することがあるため、「Amazon DynamoDB pricing」 (Amazon DynamoDB の料金) に記載された情報を慎重に検討してください。

追加リソース

- Amazon Athena DynamoDB コネクタの使用方法の概要については、「AWS Prescriptive Guidance Patterns」ガイドの「Access, query, and join Amazon DynamoDB tables using Athena」を参照してください。
- Amazon Athena DynamoDB コネクタを、Amazon DynamoDB、Athena、および Amazon QuickSight と共に使用して、シンプルなガバナンスダッシュボードを作成する方法の記事については、「AWS Big Data Blog」の「Query cross-account Amazon DynamoDB tables using Amazon Athena Federated Query」を参照してください。
- このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Google BigQuery コネクタ

Google BigQuery のために Amazon Athena コネクタを使用すると、Amazon Athena から、Google BigQuery のデータに対する SQL クエリを実行できるようになります。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- Lambda 関数における最大タイムアウト時間は 15 分です。分割が発生するたびに BigQuery に対しクエリが実行されます。Athena が読み込む結果を保存するためには、十分な時間のクエリが必要です。Lambda 関数がタイムアウトした場合には、クエリは失敗します。
- Google BigQuery では、大文字と小文字が区別されます。コネクタは、データセット名とテーブル名の大文字と小文字を修正しようとしますが、プロジェクト ID に関しは、この修正は行いません。これが必要なのは、Athena では、すべてのメタデータを小文字で扱うからです。これらの修正により、Google BigQuery への余分な呼び出しが多数発生します。
- 二進数のデータ型はサポートされません。
- Google BigQuery の同時実行性とクォータについての制限が原因で、コネクタにおいて、Google のクォータ制限に関する問題が発生する場合があります。このような問題を回避するには、可

能な限り多くの制約 を Google BigQuery に適用します。BigQuery でのクォータの詳細については、Google BigQuery のドキュメントで「割り当てと上限」を参照してください。

パラメータ

このセクションの Lambda 環境変数により、Google BigQuery コネクタの設定を行います。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。この口ケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- gcp_project_id – コネクタの読み込み対象となるデータセットを含むプロジェクト ID (プロジェクト名ではありません、例:`semiotic-primer-1234567`)。
- secret_manager_gcp_creds_name – JSON 形式の BigQuery 認証情報が含まれている、AWS Secrets Manager 内のシークレットの名前 (例: `GoogleCloudPlatformCredentials`)。
- big_query_endpoint — (オプション) BigQuery プライベートエンドポイントの URL。このパラメータは、プライベートエンドポイント経由で BigQuery にアクセスする場合に使用します。

分割とビュー

BigQuery コネクタはテーブルのクエリに BigQuery Storage Read API を使用し、BigQuery Storage API はビューをサポートしないため、コネクタはビューに単一の分割を用いる BigQuery クライアントを使用します。

パフォーマンス

BigQuery コネクタはテーブルのクエリに BigQuery Storage Read API を使用し、この API は BigQuery マネージドストレージへの迅速なアクセスを提供する RPC ベースのプロトコルを使用します。BigQuery Storage Read API の詳細については、Google Cloud ドキュメントの「Use the BigQuery Storage Read API to read table data」を参照してください。

列のサブセットを選択すると、クエリランタイムが大幅に短縮され、スキャンされるデータが減ります。このコネクタは、同時実行数が増えるとクエリに失敗する可能性があり、一般に動作が遅いコネクタです。

Athena Google BigQuery コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、`LIMIT` 句、`ORDER BY` 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

`LIMIT N` ステートメントにより、クエリによってスキャンされるデータが削減されます。`LIMIT N` プッシュダウンを使用すると、コネクタは `N` 行のみを Athena に返します。

上位 N 件のクエリ

上位 `N` 件のクエリは、結果セットの順序と返される行数に対する制限を指定します。このタイプのクエリを使用して、データセットの上位 `N` 個の最大値または上位 `N` 個の最小値を決定できます。上位 `N` 件のプッシュダウンを使用すると、コネクタは `N` 件の順序付けられた行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの `WHERE` 句内の式です。Athena Google BigQuery コネクタは、これらの式を組み合わせて Google BigQuery に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Google BigQuery コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値:
  EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
ORDER BY col_a DESC
LIMIT 10;
```

パススルークエリ

Google BigQuery コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Google BigQuery でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Google BigQuery 内のデータソースにクエリをプッシュダウンします。クエリは `customer` テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

Amazon Athena Google BigQuery コネクタプロジェクトは、Apache-2.0 ライセンスの下で使用許諾されています。

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Google Cloud Storage コネクタ

Amazon Athena Google Cloud Storage コネクタを使用すると、Amazon Athena は、Google Cloud Storage (GCS) バケットに保存されている Parquet および CSV ファイルに対してクエリを実行できます。1 つ以上の Parquet ファイルまたは CSV ファイルを GCS バケットの未分割または分割されたフォルダーにグループ化すると、AWS Glue データベーステーブルに整理できます。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

前提条件

- Google Cloud Storage のバケットとフォルダに対応する AWS Glue データベースとテーブルを設定します。手順については、このドキュメントの後半にある「AWS Glue でのデータベースとテーブルのセットアップ」を参照してください。
- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

- 現在、コネクタはパーティション列 (AWS Glue テーブルスキーマにある `string` または `varchar`) の VARCHAR タイプのみをサポートしています。他のパーティションフィールドタイプでは、Athena でクエリを実行するとエラーが発生します。

用語

GCS コネクタに関連する用語を次に示します。

- ハンドラー — GCS バケットにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – GCS バケットからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – GCS バケットからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — GCS バケットからメタデータとデータレコードの両方を取得する Lambda ハンドラー。

[サポートされているファイルの種類]

GCS コネクタは Parquet ファイルタイプと CSV ファイルタイプをサポートしています。

> **Note**
>
> CSV ファイルと Parquet ファイルの両方を同じ GCS バケットまたはパスに配置しないようにしてください。これを行うと、Parquet ファイルを CSV として読み込もうとしたり、その逆を行おうとしたりするため、ランタイムエラーが発生する可能性があります。

パラメータ

このセクションの Lambda 環境変数を使用して GCS コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON で工

ンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- secret_manager_gcp_creds_name – JSON 形式の GCS 認証情報が含まれている、AWS Secrets Manager 内のシークレットの名前 (例: `GoogleCloudPlatformCredentials`)。

AWS Glue でのデータベースとテーブルのセットアップ

GCS コネクタの組み込みスキーマ推論機能には制限があるため、AWS Glue メタデータに使用することをお勧めします。以下の手順では、AWS Glue で Athena からアクセスできるデータベースとテーブルを作成する方法を示しています。

AWS Glue でデータベースを作成する

AWS Glue コンソールを使用して、GCS コネクタで使用するデータベースを作成できます。

AWS Glue にデータベースを作成するには

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインから、[Databases] (データベース) を選択します。
3. [Add database] (データベースの追加) を選択します。
4. [Name] (名前) には、GCS コネクタで使用するデータベースの名前を入力します。
5. [Location] (場所) には、`s3://google-cloud-storage-flag.` を指定します。この場所は、AWS Glue データベースに Athena でクエリされる GCS データのテーブルが含まれていることを GCS コネクタに伝えます。コネクタは、このフラグのある Athena 内のデータベースを認識し、そうでないデータベースは無視します。
6. [データベースの作成] を選択します。

AWS Glue でのテーブルの作成

これで、データベース用のテーブルを作成できるようになりました。GCS コネクタで使用する AWS Glue テーブルを作成するときは、追加のメタデータを指定する必要があります。

AWS Glue コンソールでテーブルを作成するには

1. AWS Glue コンソールで、ナビゲーションペインから [Tables] (テーブル) を選択します。
2. [Tables] (テーブル) ページで、[Add table] (テーブルの追加) を選択します。
3. [Set table properties] (テーブルプロパティの設定) ページで、次の情報を入力します。
   - 名前 – テーブル用に一意の名前を入力します。
   - データベース — GCS コネクタ用に作成した AWS Glue データベースを選択します。
   - インクルードパス — [Data store] (データストア) セクションの [Include path] (インクルードパス) に、`gs://` で始まる GCS の URI の場所を入力します (例: `gs://gcs_table/data/`)。パーティションフォルダが 1 つ以上ある場合は、パスに含めないでください。

     > **Note**
     >
     > `s3://` テーブル以外のパスを入力すると、AWS Glue コンソールにエラーが表示されます。このエラーは無視して構いません。テーブルは正常に作成されます。

   - データ形式 — [Classification] (分類) には、[CSV] または [Parquet] を選択します。
4. [Next] を選択します。
5. [Choose or define schema] (スキーマの選択または定義) ページでは、テーブルスキーマの定義を強く推奨していますが、必須ではありません。スキーマを定義しない場合、GCS コネクタは自動的にスキーマを推測しようとします。

   次のいずれかを行います。

   - GCS コネクタに自動的にスキーマを推測させたい場合は、[Next] (次へ) を選択してから [Create] (作成) を選択します。
   - 自分でスキーマを定義するには、次のセクションの手順に従います。

AWS Glue でのテーブルスキーマの定義

AWS Glue でテーブルスキーマを定義すると、より多くの手順が必要になりますが、テーブル作成プロセスをより細かく制御できます。

AWS Glue でテーブルのスキーマを定義するには

1. [Choose or define schema] (スキーマの選択または定義) ページで、[Add] (追加) を選択します。
2. [Add schema entry] (スキーマエントリの追加) ダイアログボックスを使用して、列名とデータタイプを指定します。
3. 列をパーティション列として指定するには、[Set as partition key] (パーティションキーとして設定) オプションを選択します。
4. [Save] (保存) を選択して列を保存します。
5. さらに他の列を追加するには、[Add] (追加) を選択します。
6. 列の追加が完了したら、[Next] (次へ) を選択します。
7. [Review and create] (確認して作成) ページで、テーブルを確認し、[Create] (作成) を選択します。
8. スキーマにパーティション情報が含まれている場合は、次のセクションの手順に従って、AWS Glue のテーブルのプロパティにパーティションパターンを追加します。

AWS Glue のテーブルプロパティへのパーティションパターンの追加

GCS バケットにパーティションがある場合は、AWS Glue のテーブルのプロパティにパーティションパターンを追加する必要があります。

テーブルプロパティ AWS Glue にパーティション情報を追加するには

1. AWS Glue で作成したテーブルの詳細ページで、[Actions] (アクション)、[Edit] (編集) の順に選択します。
2. [Edit table] (テーブルの編集) ページで、[Table properties] (テーブルプロパティ) セクションまで下にスクロールします。
3. [Add] (追加) を選択して、パーティションキーを追加します。
4. [Key] (キー) に「**partition.pattern**」と入力します。このキーはフォルダーパスパターンを定義します。
5. [Value] (値) には、**StateName=${statename}/ZipCode=${zipcode}/** のようなフォルダーパスパターンを入力します。**${}** で囲まれた **statename** と **zipcode** はパーティションの列名です。GCS コネクタは、Hive パーティションスキームと Hive 以外のパーティションスキームの両方をサポートします。
6. 完了したら、[Save] を選択します。

7. 作成したばかりのテーブルプロパティを表示するには、[Advanced properties] (詳細プロパティ) タブを選択します。

この時点で、Athena コンソールに移動できます。AWS Glue で作成したデータベースとテーブルは、Athena でクエリできます。

サポートされるデータ型

次の表に、CSV と Parquet 向けにサポートされているデータ型を示します。

CSV

[Nature of data] (データの性質)	[Inferred Data Type] (推定されたデータ型)
数字のように見えるデータ	BIGINT
文字列のように見えるデータ	VARCHAR
浮動小数点 (浮動小数点、倍精度浮動小数点、10 進数) のように見えるデータ	DOUBLE
日付のように見えるデータ	タイムスタンプ
true/false の値を含むデータ	BOOL

Parquet

[PARQUET]	[Athena (Arrow)] (Athena (矢印))
BINARY	VARCHAR
BOOLEAN	BOOL
DOUBLE	DOUBLE
ENUM	VARCHAR
FIXED_LEN_BYTE_ARRAY	DECIMAL

[PARQUET]	[Athena (Arrow)] (Athena (矢印))
FLOAT	FLOAT (32 ビット)
INT32	1. INT32 2. DATEDAY (Parquet 列の論理タイプが DATE である場合)
INT64	1. INT64 2. TIMESTAMP (Parquet 列の論理タイプが TIMESTAMP である場合)
INT96	タイムスタンプ
MAP	MAP
STRUCT	STRUCT
LIST	LIST

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-gcs.yaml ファイルの `Policies` セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- AWS Glue Data Catalog – GCS コネクタには、スキーマ情報を取得するために AWS Glue Data Catalog への読み取り専用アクセス権が必要です。
- CloudWatch Logs – コネクタは、ログを保存するために CloudWatch Logs にアクセスする必要があります。

パフォーマンス

テーブルスキーマにパーティションフィールドが含まれていて、`partition.pattern` テーブルプロパティが正しく設定されている場合は、クエリの `WHERE` 句にパーティションフィールドを含め

ることができます。このようなクエリの場合、GCS コネクタはパーティション列を使用して GCS フォルダーのパスを絞り込み、GCS フォルダー内の不要なファイルをスキャンしないようにします。

Parquet データセットには、列のサブセットを選択すると、スキャンされるデータが少なくなります。通常、これにより、列プロジェクションが適用されるときのクエリ実行時間が短くなります。

CSV データセットでは列の投影はサポートされていないため、スキャンされるデータ量が減ることはありません。

`LIMIT` 句はスキャンされるデータ量を減らしますが、述語が提供されない場合は、`LIMIT` 句を含む `SELECT` クエリは少なくとも 16 MB のデータをスキャンすることを想定する必要があります。GCS コネクタでは、`LIMIT` 句が適用されているかに関わらず、小さいデータセットよりも大きいデータセットに対する方が、多くのデータをスキャンします。例えば、クエリ `SELECT * LIMIT 10000` は、小さな基盤データセットよりも大きな基盤データセットに対して、より多くのデータをスキャンします。

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena HBase コネクタ

Amazon Athena HBase コネクタは、Amazon Athena での Apache HBase インスタンスとの通信を可能にし、HBase データを SQL でクエリできるようにします。

従来のリレーショナルデータストアとは異なり、HBase コレクションにはスキーマが設定されていません。HBase にはメタデータストアがありません。HBase コレクションの各エントリには、異なるフィールドとデータ型を設定できます。

HBase コネクタは、テーブルスキーマ情報を生成するために、基本スキーマ推論と AWS Glue Data Catalog メタデータの 2 つのメカニズムをサポートしています。

デフォルトでは、スキーマ推論が選択されます。このオプションは、コレクション内の少数のドキュメントをスキャンし、すべてのフィールドの合併集合を作成して、強制的にフィールドのデータ型が

重複しないようにします。このオプションは、ほとんどのエントリが同一であるコレクションに適しています。

より多くのデータ型を持つコレクションに対して、このコネクタでは、AWS Glue Data Catalog からのメタデータの取得をサポートします。HBase 名前空間およびコレクション名と一致する AWS Glue データベースとテーブルがある場合、コネクタは対応する AWS Glue テーブルからスキーマ情報を取得します。AWS Glue テーブルを作成するときは、HBase コレクションからアクセスする可能性のあるすべてのフィールドのスーパーセットにすることをお勧めします。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションの Lambda 環境変数を使用して HBase コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。

- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- disable_glue – (オプション) これが存在し、true に設定されている場合、コネクタは AWS Glue からの補足メタデータ取得は試みません。
- glue_catalog – (オプション) クロスアカウントの AWS Glue カタログを指定するために、このオプションを使用します。デフォルトでは、コネクタは自身の AWS Glue アカウントからメタデータを取得しようとします。
- default_hbase – 存在する場合、カタログ固有の環境変数が存在しない場合に使用する HBase 接続文字列を指定します。

接続文字列の指定

コネクタで使用する HBase インスタンスの HBase 接続の詳細を定義する 1 つまたは複数のプロパティを指定できます。そのためには、Athena で使用するカタログ名と対応した Lambda の環境変数を設定します。例えば、次のクエリを使用して、Athena の 2 つの異なる HBase インスタンスにクエリを実行するとします。

```
SELECT * FROM "hbase_instance_1".database.table
```

```
SELECT * FROM "hbase_instance_2".database.table
```

これら 2 つの SQL ステートメントを使用する際には、Lambda 関数 (`hbase_instance_1` および `hbase_instance_2`) に、2 つの環境変数を追加しておく必要があります。それぞれの値は、次の形式の HBase 接続文字列になります。

```
master_hostname:hbase_port:zookeeper_port
```

シークレットの使用

オプションで、接続文字列の詳細の一部またはすべての値について、AWS Secrets Manager を使用できます。Athena 横串検索機能を Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

Secrets Manager が提供するシークレット名を接続文字列に入れるために ${my_secret} 構文を使用する場合、コネクタは、シークレット名を Secrets Manager のユーザー名とパスワードの値に置き換えます。

たとえば、hbase_instance_1 で Lambda 環境変数を次の値に設定した場合を考えます。

```
${hbase_host_1}:${hbase_master_port_1}:${hbase_zookeeper_port_1}
```

Athena Query Federation SDK は、自動的に hbase_instance_1_creds という名前のシークレットを Secrets Manager から取得しよう試み、取得した値は ${hbase_instance_1_creds} の場所に挿入します。接続文字列の中で、${ } 文字の組み合わせにより囲まれている任意の部分は、Secrets Manager から提供されたシークレットとして認識されます コネクタにより Secrets Manager 内で検出されないシークレット名を指定した場合、コネクタはテキストを置き換えません。

AWS Glue でのデータベースとテーブルのセットアップ

コネクタの組み込みスキーマ推論は、HBase で文字列としてシリアル化された値のみをサポートします (例: String.valueOf(int))。コネクタの組み込みスキーマ推論機能は限られているため、代わりにメタデータに AWS Glue を使用したい場合があるかもしれません。HBase で使用するために AWS Glue テーブルを有効にするには、補足メタデータを供給したい HBase 名前空間とテーブルに一致する名前の AWS Glue データベースとテーブルが必要です。HBase 列ファミリーの命名規則の使用はオプションで、必須ではありません。

補足メタデータのために AWS Glue を使用するには

1. AWS Glue コンソールでテーブルとデータベースを編集する場合は、次のテーブルプロパティを追加します。

   - hbase-metadata-flag – このプロパティは、補足メタデータのテーブルをコネクタが使用できることを HBase コネクタに示します。hbase-metadata-flag プロパティがテーブルプロパティのリスト内に存在するのであれば、この hbase-metadata-flag に任意の値を指定することが可能です。
   - hbase-native-storage-flag – このフラグを使用して、コネクタでサポートされている 2 つの値のシリアル化モードを切り替えます。デフォルトでは、このフィールドが存在しない場合、コネクタはすべての値が HBase に文字列として格納されていると見なします。そのため、HBase からの INT、BIGINT、DOUBLE などのデータ型を文字列として解析しようとします。このフィールドに AWS Glue のテーブルのいずれかの値が設定されている

場合、コネクタは「ネイティブ」ストレージモードに切り替わり、次の関数を使用して INT、BIGINT、BIT、および DOUBLE をバイトとして読み取ろうとします。

```
ByteBuffer.wrap(value).getInt()
ByteBuffer.wrap(value).getLong()
ByteBuffer.wrap(value).get()
ByteBuffer.wrap(value).getDouble()
```

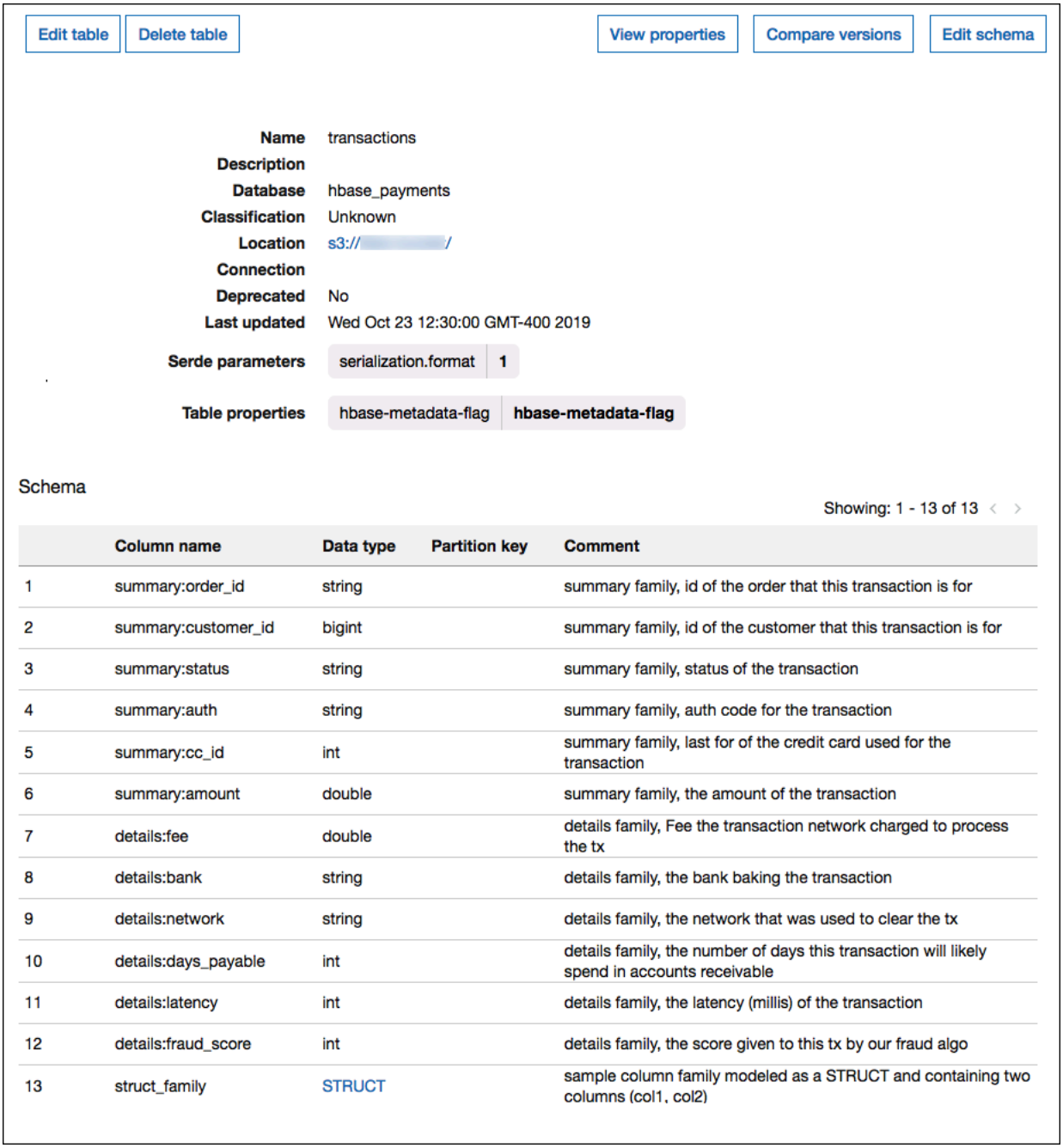
2. このドキュメントに記載されているとおりに、AWS Glue 用として適切なデータ型を使用しているか確認してください。

列ファミリーのモデリング

Athena HBase コネクタは HBase 列ファミリーをモデル化する 2 つの方法 (`family:column` のような完全修飾 (フラット) 命名法、または STRUCT オブジェクトの使用) をサポートしています。

STRUCT モデルでは、STRUCT フィールドの名前は列ファミリーと一致し、STRUCT の子要素はファミリーの列の名前と一致するようになっています。ただし、述語プッシュダウンと列指向の読み込みが STRUCT のような複合型ではまだ完全にはサポートされていないので、STRUCT の使用は現時点では推奨されていません。

次の画像は、2 つのアプローチを組み合わせて使用している AWS Glue で設定されたテーブルを示しています。



サポートされるデータ型

コネクタはすべての HBase 値を基本バイト型として取得します。次に、AWS Glue データカタログでテーブルをどのように定義したかに基づいて、値を次の表の Apache Arrow データ型の 1 つにマッピングします。

AWS Glue データ型	Apache Arrow データ型
整数	INT
bigint	BIGINT
double	FLOAT8
フロート	FLOAT4
ブール値	BIT
バイナリ	VARBINARY
string	VARCHAR

> **Note**
>
> AWS Glue を使用してメタデータを補足しない場合、コネクタのスキーマ推論ではデータ型 `BIGINT`、`FLOAT8`、および `VARCHAR` のみを使用します。

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-hbase.yaml ファイルの `Policies` セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- AWS Glue Data Catalog – HBase コネクタには、スキーマ情報を取得するために AWS Glue Data Catalog への読み込み専用アクセス権が必要です。
- CloudWatch Logs – コネクタは、ログを保存するために CloudWatch Logs にアクセスする必要があります。

- AWS Secrets Manager 読み込みアクセス – HBase エンドポイントの詳細を Secrets Manager に保存する場合は、それらのシークレットへのアクセスをコネクタに許可する必要があります。
- VPC アクセス – コネクタには、VPC に接続して HBase インスタンスと通信できるように、VPC にインターフェイスをアタッチおよびデタッチできることが必要です。

パフォーマンス

Athena HBase コネクタは、各リージョンサーバーを並行して読み込むことによって、HBase インスタンスに対するクエリを並列化しようとします。Athena HBase コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。

また、Lambda 関数は、クエリがスキャンするデータを削減するために、射影プッシュダウンを実行します。ただし、列のサブセットを選択した場合は、クエリの実行時間が長くなることがあります。`LIMIT` 句はスキャンされるデータ量を減らしますが、述語を提供しない場合、`LIMIT` 句を含む `SELECT` クエリは少なくとも 16 MB のデータをスキャンすることを想定する必要があります。

HBase はクエリに失敗しやすく、クエリの実行時間が変動しやすいです。クエリを正常に実行するために、何度も再試行する必要がある場合があります。HBase コネクタは同時実行によるスロットリングに強いです。

パススルークエリ

HBase コネクタはパススルークエリをサポートしており、NoSQL ベースです。Apache HBase のクエリに関する詳細については、Apache ドキュメントの「Querying HBase」を参照してください。

HBase でパススルークエリを使用するには、以下の構文を使用します。

```
SELECT * FROM TABLE(
        system.query(
            database => 'database_name',
            collection => 'collection_name',
            filter => '{query_syntax}'
        ))
```

以下の HBase パススルークエリ例は、`default` データベースの `employee` コレクション内で 24 歳または 30 歳の従業員をフィルタリングします。

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'default',
```

```
            COLLECTION => 'employee',
            FILTER => 'SingleColumnValueFilter(''personaldata'', ''age'', =,
''binary:30'')' ||
                      ' OR SingleColumnValueFilter(''personaldata'', ''age'', =,
''binary:24'')'
        ))
```

ライセンス情報

Amazon Athena HBase コネクタプロジェクトは、Apache-2.0 License でライセンスされています。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Hortonworks コネクタ

Hortonworks 用の Amazon Athena コネクタを使用すると、Amazon Athena で Cloudera Hortonworks データプラットフォームに SQL クエリを実行できます。コネクタによって Athena SQL クエリは同等の HiveQL 構文に変換されます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

用語

Hortonworks Hive コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して Hortonworks Hive コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
hive://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	HiveMuxCompositeHandler
メタデータハンドラー	HiveMuxMetadataHandler
レコードハンドラー	HiveMuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが myhivecatalog の場合、環境変数の名前は myhivecatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

以下に、2 つのデータベースインスタンス (デフォルト値の hive1、および hive2) をサポートする、Hive MUX Lambda 関数のプロパティ例を示します。

プロパティ	Value
default	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive_catalog1_connection_string	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive_catalog2_connection_string	hive://jdbc:hive2://hive2:10000/default?UID=sample&PWD=sample

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

  ```
  {"username": "${username}", "password": "${password}"}
  ```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${Test/RDS/hive1host}` が含まれています。

```
hive://jdbc:hive2://hive1host:10000/default?...&${Test/RDS/hive1host}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
hive://jdbc:hive2://hive1host:10000/default?...&UID=sample2&PWD=sample2&...
```

現在、Hortonworks Hive コネクタは UID と PWD の JDBC プロパティを認識します。

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Hortonworks Hive インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	HiveCompositeHandler
メタデータハンドラー	HiveMetadataHandler
レコードハンドラー	HiveRecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の Hortonworks Hive インスタンス用のプロパティ例を次に示します。

プロパティ	Value
default	hive://jdbc:hive2://hive1host:10000/default?secret=${Test/RDS/hive1host}

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス
`spill_put_request_headers`	(オプション) スピルに使用される Amazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC、Hortonworks Hive、および Arrow の間で対応するデータ型を示します。

JDBC	Hortonworks Hive	Arrow
ブール値	ブール値	Bit
整数	TINYINT	Tiny
ショート	SMALLINT	Smallint
整数	INT	Int
Long	BIGINT	Bigint
フロート	float4	Float4
ダブル	float8	Float8
日付	date	DateDay

JDBC	Hortonworks Hive	Arrow
タイムスタンプ	timestamp	DateMilli
文字列	VARCHAR	Varchar
バイト	bytes	Varbinary
BigDecimal	10 進数	10 進数
配列	該当なし (注記を参照)	リスト

> **Note**
>
> 現在、Hortonworks Hive は集計型 ARRAY、MAP、STRUCT、UNIONTYPE をサポートしていません。集計型の列は、SQL における VARCHAR 列のように扱われます。

パーティションと分割

パーティションは、コネクタを分割する方法を決定するために使用されます。Athena は `varchar` 型の合成列を作成し、コネクタが分割を生成できるようにするために、テーブルに対するパーティションのスキームを示します。コネクタは実際のテーブル定義を変更しません。

パフォーマンス

Hortonworks Hive は静的パーティションをサポートしています。Athena Hortonworks Hive コネクタは、これらのパーティションからデータを並列に取得できます。パーティション分散が一様で非常に大きなデータセットをクエリする場合は、静的パーティショニングの使用を強くお勧めします。列のサブセットを選択すると、クエリランタイムが大幅に短縮され、スキャンされるデータが減ります。Hortonworks Hive コネクタは同時実行によるスロットリングに強いです。

Athena Hortonworks Hive コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、LIMIT 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

`LIMIT N` ステートメントにより、クエリによってスキャンされるデータが削減されます。`LIMIT N` プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena Hortonworks Hive コネクタは、これらの式を組み合わせて Hortonworks Hive に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Hortonworks Hive コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

パススルークエリ

Hortonworks Hive コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Hortonworks Hive でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Hortonworks Hive 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の Hortonworks Hive コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena の Apache Kafka コネクタ

Apache Kafka 用の Amazon Athena コネクタを使用して、Amazon Athena で Apache Kafka トピックに対して SQL クエリを実行できます。このコネクタを使用すると、Apache Kafka トピックをテーブルとして、メッセージを Athena の行として表示できます。

前提条件

Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウント にコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- フィルター条件における日付とタイムスタンプのデータ型は、適切なデータ型にキャストする必要があります。

- 日付とタイムスタンプのデータ型は、CSV ファイルタイプではサポートされていないため、varchar 値として扱われます。
- ネストされた JSON フィールドへのマッピングはサポートされていません。コネクタは最上位のフィールドのみをマッピングします。
- コネクタは複合型をサポートしていません。複合型は文字列として解釈されます。
- 複合型の JSON 値を抽出または処理するには、Athena に用意されている JSON 関連の関数を使用してください。詳細については、「JSON からのデータの抽出」を参照してください。
- コネクタは、Kafka メッセージメタデータへのアクセスをサポートしていません。

用語

- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- Kafka エンドポイント - Kafka インスタンスへの接続を確立するテキスト文字列。

クラスター間の互換性

Kafka コネクタは、次のクラスタータイプで使用できます。

- スタンドアロン Kafka - Kafka への直接の接続 (認証または非認証)。
- Confluent — Confluent Kafka への直接接続。Confluent Kafka データに Athena を使用する方法については、「AWS Business Intelligence Blog」の「Visualize Confluent data in Amazon QuickSight using Amazon Athena」を参照してください。

Confluent への接続

Confluent への接続には以下の手順が必要です。

1. Confluent から API キーを生成する。
2. Confluent API キーのユーザー名とパスワードを入力する。AWS Secrets Manager
3. Kafka コネクターの `secrets_manager_secret` 環境変数にシークレット名を指定します。

4. このドキュメントの Kafka コネクタのセットアップ セクションの手順に従ってください。

サポートされた認証方法

コネクタでは、次の認証方法がサポートされています。

- SSL
- SASL/SCRAM
- SASL/PLAIN
- SASL/PLAINTEXT
- NO_AUTH
- セルフマネージド型の Kafka と Confluent Platform — SSL、SASL/SCRAM、SASL/プレーンテキスト、NO_AUTH
- セルフマネージド型の Kafka および Confluent Cloud — SASL/PLAIN

詳細については、「Athena Kafka コネクタの認証を設定」を参照してください。

対応する入力データ形式

コネクタは、次の入力データ形式をサポートします。

- JSON
- CSV

パラメータ

このセクションに示した Lambda 環境変数を使って、Athena Kafka コネクタを設定できます。

- auth_type - クラスターの認証タイプを指定します。コネクタは、次のタイプの認証をサポートしています。
  - NO_AUTH - Kafka に直接接続します (たとえば、認証を使用しない EC2 インスタンスにデプロイされた Kafka クラスターなど)。
  - SASL_SSL_PLAIN - このメソッドは、SASL_SSL セキュリティプロトコルと PLAIN SASL メカニズムを使用します。詳細については、Apache Kafka ドキュメントの「SASL 設定」を参照してください。

- SASL_PLAINTEXT_PLAIN - このメソッドは、SASL_PLAINTEXT セキュリティプロトコルと PLAIN SASL メカニズムを使用します。詳細については、Apache Kafka ドキュメントの「SASL 設定」を参照してください。
- SASL_SSL_SCRAM_SHA512 - この認証タイプを使用して Apache Kafka クラスターへのアクセスを制御できます。このメソッドでは、ユーザー名とパスワードを AWS Secrets Manager に保存します。シークレットは Kafka クラスターに関連付けられている必要があります。詳細については、Apache Kafka ドキュメントの「SASL/SCRAM を使用する認証」を参照してください。
- SASL_PLAINTEXT_SCRAM_SHA512 - このメソッドは、SASL_PLAINTEXT セキュリティプロトコルと SCRAM_SHA512 SASL メカニズムを使用します。このメソッドでは、AWS Secrets Manager に保存したユーザー名とパスワードを使用します。詳細については、Apache Kafka ドキュメントの「SASL 設定」セクションを参照してください。
- SSL - SSL 認証では、キーストアとトラストストアのファイルを使用して Apache Kafka クラスターに接続します。トラストストアファイルとキーストアファイルを生成し、それらを Amazon S3 バケットにアップロードして、コネクタをデプロイするときに Amazon S3 への参照を提供する必要があります。キーストア、トラストストア、および SSL キーは AWS Secrets Manager に保存されます。コネクタをデプロイする際に、クライアントは AWS のシークレットキーを提供する必要があります。詳細については、Apache Kafka ドキュメントの「SSL を使用した暗号化と認証」を参照してください。

  詳細については、「Athena Kafka コネクタの認証を設定」を参照してください。

- certificates_s3_reference - 証明書 (キーストアとトラストストアのファイル) を含む Amazon S3 の場所。
- disable_spill_encryption – (オプション) True に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は False です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- kafka_endpoint - Kafka に提供するエンドポイントの詳細。
- secrets_manager_secret - 認証情報が保存されている AWS シークレットの名前。
- スピルパラメータ - Lambda 関数は、メモリに収まらないデータを Amazon S3 に一時的に保存 (「スピル」) します。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。次の表のパラメータを使用して、スピル場所を指定します。

パラメータ	説明
spill_bucket	必須。Lambda 関数がデータをスピルできる Amazon S3 バケットの名前。
spill_prefix	必須。Lambda 関数がデータをスピルさせることができるスピルバケット内のプリフィックス。
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

- サブネット ID — Lambda 関数がデータソースへのアクセスに使用できるサブネットに対応する 1 つ以上のサブネット ID。
  - パブリックな Kafka クラスターまたは標準の Confluent Cloud クラスタ — コネクターを NAT ゲートウェイのあるプライベートサブネットに関連付けます。
  - プライベート接続の Confluent Cloud クラスタ — Confluent Cloud クラスタへのルートがあるプライベートサブネットにコネクタを関連付けます。
    - AWS Transit Gateway の場合、サブネットは Confluent Cloud が使用するのと同じトランジットゲートウェイに接続されている VPC 内にある必要があります。
    - VPC ピアリングの場合、サブネットは Confluent Cloud VPC にピアリングされている VPC 内にある必要があります。
    - AWS PrivateLinkの場合、サブネットは Confluent Cloud に接続する VPC エンドポイントへのルートがある VPC 内にある必要があります。

> **Note**
>
> プライベートリソースにアクセスするためにコネクタを VPC にデプロイし、Confluent などのパブリックにアクセス可能なサービスにも接続する場合は、NAT ゲートウェイを持つプライベートサブネットにコネクタを関連付ける必要があります。詳細については、Amazon VPC ユーザーガイドの「NAT ゲートウェイ」を参照してください。

サポートされるデータ型

次の表に、Kafka と Apache Arrow に対応するデータ型を示します。

Kafka	Arrow
CHAR	VARCHAR
VARCHAR	VARCHAR
TIMESTAMP	ミリ秒
DATE	DAY
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	FLOAT8
DOUBLE	FLOAT8

パーティションと分割

Kafka トピックはパーティションに分割されています。各パーティションは順序付けられています。パーティション内の各メッセージには、オフセットと呼ばれるインクリメンタル ID があります。各 Kafka パーティションは、並列処理のためにさらに複数のスプリットに分割されます。データは、Kafka クラスターに設定された保持期間中、利用できます。

ベストプラクティス

ベストプラクティスとして、次の例のように、Athena にクエリを実行するときは、述語プッシュダウンを使用してください。

```
SELECT *
FROM "kafka_catalog_name"."glue_schema_registry_name"."glue_schema_name"
```

```
WHERE integercol = 2147483647
```

```
SELECT *
FROM "kafka_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

Kafka コネクタのセットアップ

コネクタを使用する前に、Apache Kafka クラスターをセットアップし、AWS Glue スキーマレジストリを使用してスキーマを定義し、コネクタの認証を設定する必要があります。

AWS Glue スキーマレジストリを使用する際は、次の点に注意してください。

- AWS Glue スキーマレジストリの [Description] (説明) フィールドのテキストに文字列 `{AthenaFederationKafka}` が含まれていることを確認してください。このマーカー文字列は、Amazon Athena Kafka コネクタで使用する AWS Glue レジストリに必要です。
- 最高のパフォーマンスを得るには、 データベース名とテーブル名には小文字のみを使用してください。大文字と小文字が混在すると、コネクタは大文字と小文字を区別しない検索を実行するため、計算量が多くなります。

Apache Kafka 環境と AWS Glue スキーマレジストリをセットアップするには

1. Apache Kafka 環境をセットアップします。
2. JSON 形式の Kafka トピック記述ファイル (つまり、そのスキーマ) を AWS Glue スキーマレジストリにアップロードします。詳細については、AWS Glue デベロッパーガイドの「AWS Glue スキーマ登録と連携する」を参照してください。スキーマの例については、次のセクションを参照してください。

AWS Glue スキーマレジストリのスキーマの例

スキーマを AWS Glue スキーマレジストリにアップロードする際は、このセクションの例の形式を使用してください。

JSON タイプのスキーマの例

次の例では、AWS Glue スキーマレジストリに作成されるスキーマは、`dataFormat` の値として `json` を指定し、`topicName` の値として `datatypejson` を使用します。

> **Note**
>
> topicName の値は、Kafka のトピック名と同じ大文字と小文字を使用する必要があります。

```
{
  "topicName": "datatypejson",
  "message": {
    "dataFormat": "json",
    "fields": [
      {
        "name": "intcol",
        "mapping": "intcol",
        "type": "INTEGER"
      },
      {
        "name": "varcharcol",
        "mapping": "varcharcol",
        "type": "VARCHAR"
      },
      {
        "name": "booleancol",
        "mapping": "booleancol",
        "type": "BOOLEAN"
      },
      {
        "name": "bigintcol",
        "mapping": "bigintcol",
        "type": "BIGINT"
      },
      {
        "name": "doublecol",
        "mapping": "doublecol",
        "type": "DOUBLE"
      },
      {
        "name": "smallintcol",
        "mapping": "smallintcol",
        "type": "SMALLINT"
      },
      {
```

```
        "name": "tinyintcol",
        "mapping": "tinyintcol",
        "type": "TINYINT"
      },
      {
        "name": "datecol",
        "mapping": "datecol",
        "type": "DATE",
        "formatHint": "yyyy-MM-dd"
      },
      {
        "name": "timestampcol",
        "mapping": "timestampcol",
        "type": "TIMESTAMP",
        "formatHint": "yyyy-MM-dd HH:mm:ss.SSS"
      }
    ]
  }
}
```

CSV タイプスキーマの例

次の例では、AWS Glue スキーマレジストリに作成されるスキーマは、dataFormat の値として csv を指定し、topicName の値として datatypecsvbulk を使用します。topicName の値は、Kafka のトピック名と同じ大文字と小文字を使用する必要があります。

```
{
  "topicName": "datatypecsvbulk",
  "message": {
    "dataFormat": "csv",
    "fields": [
      {
        "name": "intcol",
        "type": "INTEGER",
        "mapping": "0"
      },
      {
        "name": "varcharcol",
        "type": "VARCHAR",
        "mapping": "1"
      },
      {
        "name": "booleancol",
```

```
        "type": "BOOLEAN",
        "mapping": "2"
      },
      {
        "name": "bigintcol",
        "type": "BIGINT",
        "mapping": "3"
      },
      {
        "name": "doublecol",
        "type": "DOUBLE",
        "mapping": "4"
      },
      {
        "name": "smallintcol",
        "type": "SMALLINT",
        "mapping": "5"
      },
      {
        "name": "tinyintcol",
        "type": "TINYINT",
        "mapping": "6"
      },
      {
        "name": "floatcol",
        "type": "DOUBLE",
        "mapping": "7"
      }
    ]
  }
}
```

Athena Kafka コネクタの認証を設定

SSL、SASL/SCRAM、SASL/PLAIN、SASL/PLAINTEXT など、Apache Kafka クラスターへの認証にはさまざまなメソッドを使用できます。

次の表は、コネクタの認証タイプと、それぞれのセキュリティプロトコルおよび SASL メカニズムを示しています。詳細については、Apache Kafka ドキュメントの「セキュリティ」セクションを参照してください。

auth_type	security.protocol	sasl.mechanism	クラスタータイプの互換性
SASL_SSL_PLAIN	SASL_SSL	PLAIN	• セルフマネージド型の Kafka • Confluent Platform • Confluent Cloud
SASL_PLAINTEXT_PLAIN	SASL_PLAINTEXT	PLAIN	• セルフマネージド型の Kafka • Confluent Platform
SASL_SSL_SCRAM_SHA512	SASL_SSL	SCRAM-SHA-512	• セルフマネージド型の Kafka • Confluent Platform
SASL_PLAINTEXT_SCRAM_SHA512	SASL_PLAINTEXT	SCRAM-SHA-512	• セルフマネージド型の Kafka • Confluent Platform
SSL	SSL	該当なし	• セルフマネージド型の Kafka • Confluent Platform

SSL

クラスターが SSL 認証されている場合、トラストストアとキーストアファイルを生成して Amazon S3 バケットにアップロードする必要があります。コネクタをデプロイする際に、この Amazon S3 リファレンスを提供する必要があります。キーストア、トラストストア、および SSL キーは AWS Secrets Manager に保存されます。コネクタをデプロイする際に、AWS のシークレットキーを提供します。

Secrets Manager でのシークレットの作成については、「AWS Secrets Manager シークレットを作成する」を参照してください。

この認証タイプを使用するには、以下の表の説明どおりに環境変数を設定します。

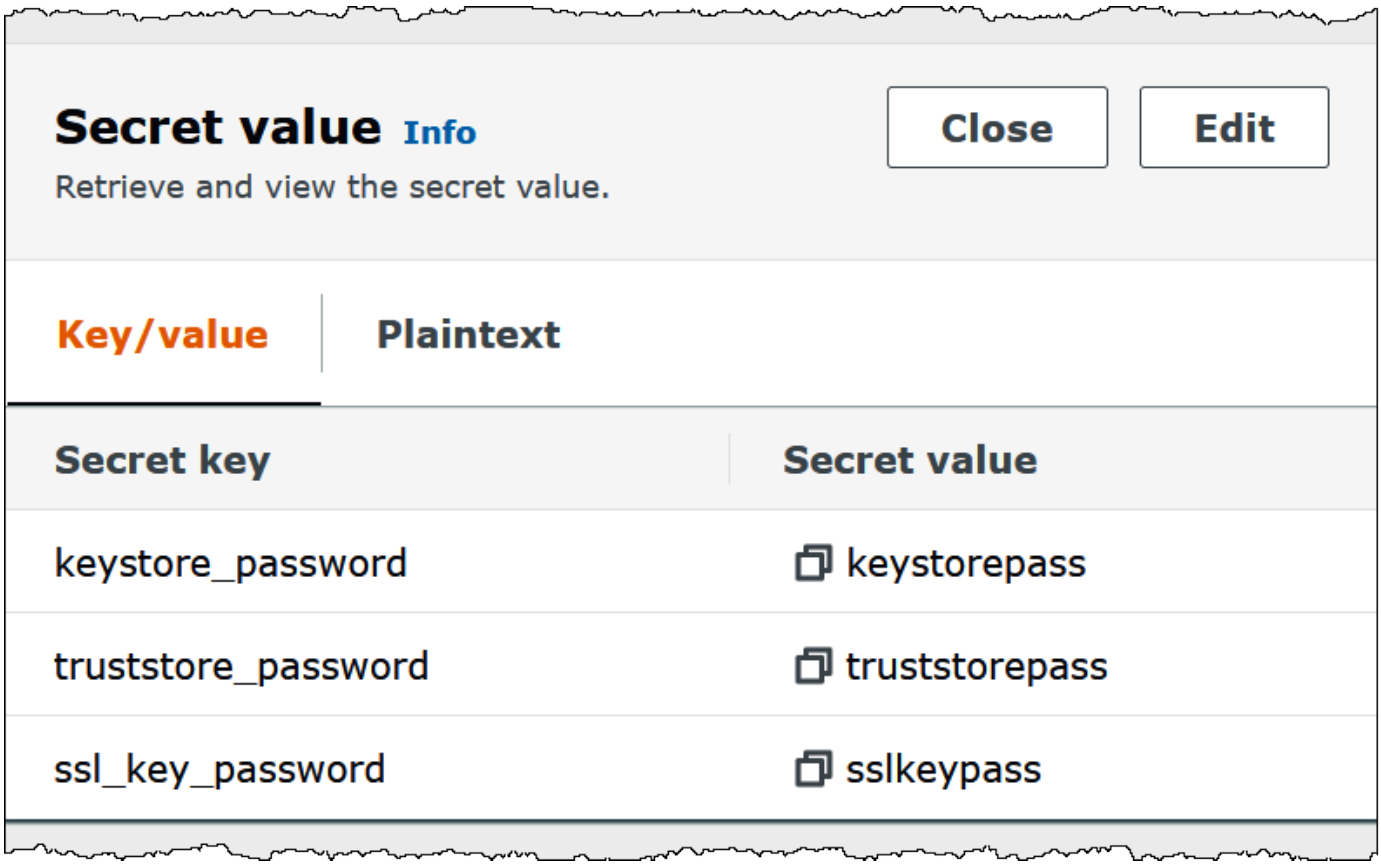
パラメータ	Value
auth_type	SSL
certificates_s3_reference	証明書を格納する Amazon S3 の場所。
secrets_manager_secret	AWS シークレットキーの名前。

Secrets Manager でシークレットを作成した後、Secrets Manager コンソールでそのシークレットを確認できます。

Secrets Manager で自分のシークレットを確認するには

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. ナビゲーションペインで [Secrets] (シークレット) を選択します。
3. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
4. 自分のシークレットの詳細ページで、[Retrieve secret value] (シークレットの値を取得する) を選択します。

   次の画像は、3 組のキー/値のペア (keystore_password、truststore_password、ssl_key_password) を持つシークレットの例を示しています。



Kafka で SSL を使用する際の詳細については、Apache Kafka ドキュメントの「SSL を使用した暗号化と認証」を参照してください。

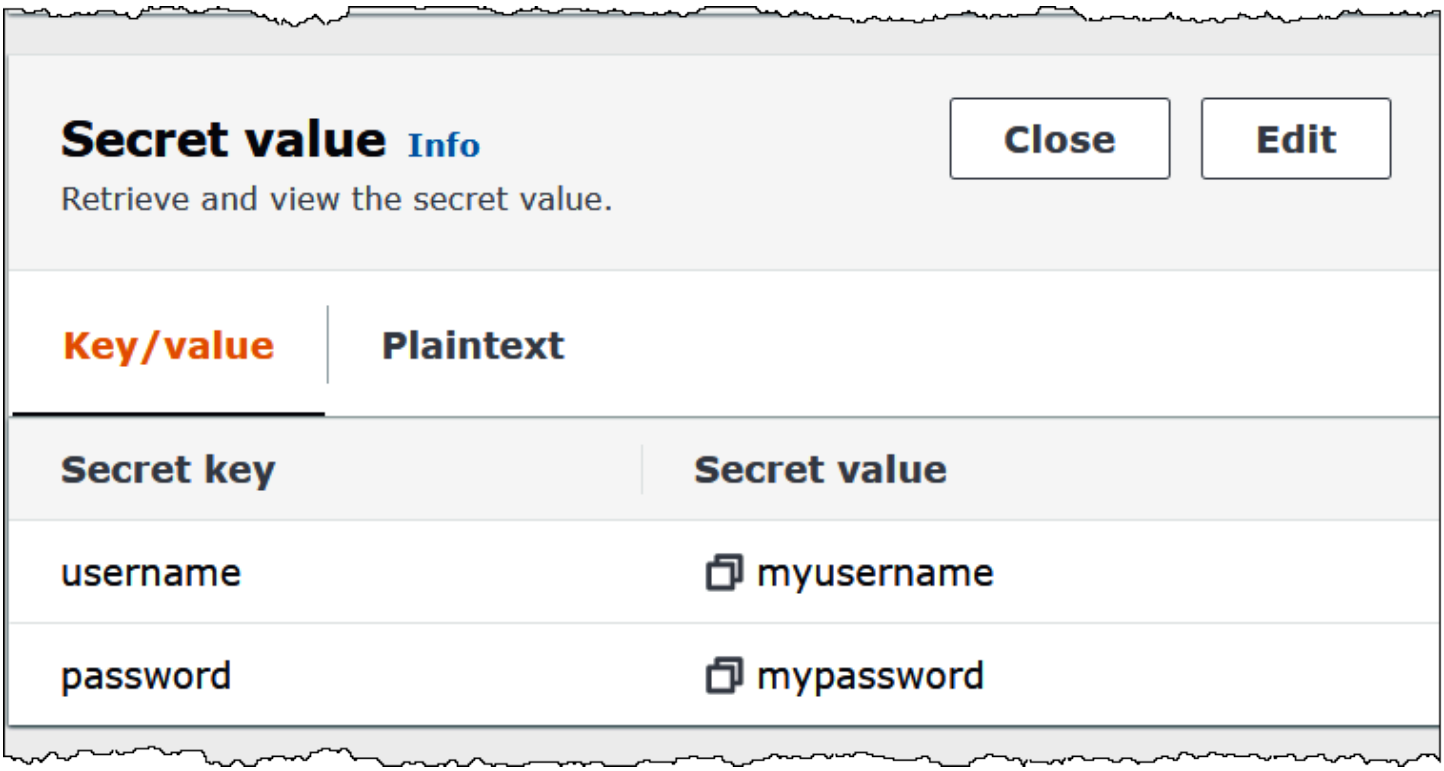
SASL/SCRAM

クラスターが SCRAM 認証を使用している場合は、コネクタをデプロイするときに、クラスターに関連付けられている Secrets Manager キーを指定します。ユーザーの AWS 認証情報 (シークレットキーとアクセスキー) は、クラスターとの認証に使用されます。

下表のように環境変数を設定します。

パラメータ	Value
auth_type	SASL_SSL_SCRAM_SHA512
secrets_manager_secret	AWS シークレットキーの名前。

次の画像は、Secrets Manager コンソールにある 2 つのキー/値のペア (1 つは `username` 用、もう 1 つは `password` 用) のシークレットの例を示しています。



Kafka で SASL/SCRAM を使用する際の詳細については、Apache Kafka ドキュメントの「SASL/SCRAM 認証の使用」を参照してください。

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena MSK コネクタ

Amazon MSK 用の Amazon Athena コネクタを使用すると、Amazon Athena で Apache Kafka トピックに対する SQL クエリを実行できるようになります。このコネクタを使用すると、Apache Kafka トピックをテーブルとして、メッセージを Athena の行として表示できます。追加情報につい

ては、「AWS ビッグデータブログ」の「Amazon Athena を使用して Amazon MSK でリアルタイムストリーミングデータを分析する」を参照してください。

前提条件

Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウント にコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- フィルター条件における日付とタイムスタンプのデータ型は、適切なデータ型にキャストする必要があります。
- 日付とタイムスタンプのデータ型は、CSV ファイルタイプではサポートされていないため、varchar 値として扱われます。
- ネストされた JSON フィールドへのマッピングはサポートされていません。コネクタは最上位のフィールドのみをマッピングします。
- コネクタは複合型をサポートしていません。複合型は文字列として解釈されます。
- 複合型の JSON 値を抽出または処理するには、Athena に用意されている JSON 関連の関数を使用してください。詳細については、「JSON からのデータの抽出」を参照してください。
- コネクタは、Kafka メッセージメタデータへのアクセスをサポートしていません。

用語

- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- Kafka エンドポイント - Kafka インスタンスへの接続を確立するテキスト文字列。

クラスター間の互換性

MSK コネクタは、次のクラスタータイプで使用できます。

- MSK プロビジョニングクラスター - クラスター容量を手動で指定、監視、スケーリングします。
- MSK Serverless クラスター - アプリケーション I/O のスケーリングに合わせて自動的にスケーリングされるオンデマンドの容量を提供します。
- スタンドアロン Kafka - Kafka への直接の接続 (認証または非認証)。

サポートされた認証方法

コネクタでは、次の認証方法がサポートされています。

- SASL/IAM
- SSL
- SASL/SCRAM
- SASL/PLAIN
- SASL/PLAINTEXT
- NO_AUTH

  詳細については、「Athena MSK コネクタの認証の設定」を参照してください。

対応する入力データ形式

コネクタは、次の入力データ形式をサポートします。

- JSON
- CSV

パラメータ

このセクションに示した Lambda 環境変数により、Athena MSK コネクタを設定できます。

- auth_type - クラスターの認証タイプを指定します。コネクタは、次のタイプの認証をサポートしています。
  - NO_AUTH - 認証なしで Kafka に直接接続します (例えば、認証を使用しない EC2 インスタンスにデプロイされた Kafka クラスターなど)。

- SASL_SSL_PLAIN - このメソッドは、SASL_SSL セキュリティプロトコルと PLAIN SASL メカニズムを使用します。
- SASL_PLAINTEXT_PLAIN - このメソッドは、SASL_PLAINTEXT セキュリティプロトコルと PLAIN SASL メカニズムを使用します。

> **Note**
>
> SASL_SSL_PLAIN および SASL_PLAINTEXT_PLAIN 認証タイプは、Apache Kafka ではサポートされていますが、Amazon MSK ではサポートされていません。

- SASL_SSL_AWS_MSK_IAM - Amazon MSK の IAM アクセス制御により、MSK クラスターの認証と認可の両方を処理できます。ユーザーの AWS 認証情報 (シークレットキーとアクセスキー) は、クラスターへの接続に使用されます。詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「IAM access control」(IAM アクセスコントロール) を参照してください。
- SASL_SSL_SCRAM_SHA512 - この認証タイプを使用して Amazon MSK クラスターへのアクセスを制御できます。このメソッドでは、ユーザー名とパスワードを AWS Secrets Manager に保存します。シークレットは Amazon MSK クラスターに関連付けられている必要があります。詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「Amazon MSK クラスターの SASL/SCRAM 認証のセットアップ」を参照してください。
- SSL - SSL 認証では、キーストアとトラストストアのファイルを使用して Amazon MSK クラスターに接続します。トラストストアファイルとキーストアファイルを生成し、それらを Amazon S3 バケットにアップロードして、コネクタをデプロイするときに Amazon S3 への参照を提供する必要があります。キーストア、トラストストア、および SSL キーは AWS Secrets Manager に保存されます。コネクタをデプロイする際に、クライアントは AWS のシークレットキーを提供する必要があります。詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「相互 TLS 認証」を参照してください。

  詳細については、「Athena MSK コネクタの認証の設定」を参照してください。

- certificates_s3_reference - 証明書 (キーストアとトラストストアのファイル) を含む Amazon S3 の場所。
- disable_spill_encryption – (オプション) True に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は False です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。

- kafka_endpoint - Kafka に提供するエンドポイントの詳細。例えば、Amazon MSK クラスターの場合、クラスターのブートストラップ URL を指定します。
- secrets_manager_secret - 認証情報が保存されている AWS シークレットの名前。このパラメータは IAM 認証には必要ありません。
- スピルパラメータ - Lambda 関数は、メモリに収まらないデータを Amazon S3 に一時的に保存 (「スピル」) します。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。次の表のパラメータを使用して、スピル場所を指定します。

パラメータ	説明
spill_bucket	必須。Lambda 関数がデータをスピルできる Amazon S3 バケットの名前。
spill_prefix	必須。Lambda 関数がデータをスピルさせることができるスピルバケット内のプリフィックス。
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、Kafka と Apache Arrow に対応するデータ型を示します。

Kafka	Arrow
CHAR	VARCHAR
VARCHAR	VARCHAR
TIMESTAMP	ミリ秒
DATE	DAY

Kafka	Arrow
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	FLOAT8
DOUBLE	FLOAT8

パーティションと分割

Kafka トピックはパーティションに分割されています。各パーティションは順序付けられています。パーティション内の各メッセージには、オフセットと呼ばれるインクリメンタル ID があります。各 Kafka パーティションは、並列処理のためにさらに複数のスプリットに分割されます。データは、Kafka クラスターに設定された保持期間中、利用できます。

ベストプラクティス

ベストプラクティスとして、次の例のように、Athena にクエリを実行するときは、述語プッシュダウンを使用してください。

```
SELECT *
FROM "msk_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE integercol = 2147483647
```

```
SELECT *
FROM "msk_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

MSK コネクタのセットアップ

コネクタを使用する前に、Amazon MSK クラスターをセットアップし、AWS Glue スキーマレジストリを使用してスキーマを定義し、コネクタの認証を設定する必要があります。

> **Note**
>
> プライベートリソースにアクセスするためにコネクタを VPC にデプロイし、Confluent などのパブリックにアクセス可能なサービスにも接続する場合は、NAT ゲートウェイを持つプライベートサブネットにコネクタを関連付ける必要があります。詳細については、Amazon VPC ユーザーガイドの「NAT ゲートウェイ」を参照してください。

AWS Glue スキーマレジストリを使用する際は、次の点に注意してください。

- AWS Glue スキーマレジストリの [Description] (説明) フィールドのテキストに文字列 `{AthenaFederationMSK}` が含まれていることを確認してください。このマーカー文字列は、Amazon Athena MSK コネクタで使用する AWS Glue レジストリに必要です。
- 最高のパフォーマンスを得るには、 データベース名とテーブル名には小文字のみを使用してください。大文字と小文字が混在すると、コネクタは大文字と小文字を区別しない検索を実行するため、計算量が多くなります。

Amazon MSK 環境と AWS Glue スキーマレジストリをセットアップするには

1. Amazon MSK 環境をセットアップします。詳細と手順については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「Amazon MSK のセットアップ」および「Amazon MSK を使い始める」を参照してください。
2. JSON 形式の Kafka トピック記述ファイル (つまり、そのスキーマ) を AWS Glue スキーマレジストリにアップロードします。詳細については、AWS Glue デベロッパーガイドの「AWS Glue スキーマ登録と連携する」を参照してください。スキーマの例については、次のセクションを参照してください。

AWS Glue スキーマレジストリのスキーマの例

スキーマを AWS Glue スキーマレジストリにアップロードする際は、このセクションの例の形式を使用してください。

JSON タイプのスキーマの例

次の例では、AWS Glue スキーマレジストリに作成されるスキーマは、`dataFormat` の値として `json` を指定し、`topicName` の値として `datatypejson` を使用します。

> **Note**
>
> topicName の値は、Kafka のトピック名と同じ大文字と小文字を使用する必要があります。

```
{
  "topicName": "datatypejson",
  "message": {
    "dataFormat": "json",
    "fields": [
      {
        "name": "intcol",
        "mapping": "intcol",
        "type": "INTEGER"
      },
      {
        "name": "varcharcol",
        "mapping": "varcharcol",
        "type": "VARCHAR"
      },
      {
        "name": "booleancol",
        "mapping": "booleancol",
        "type": "BOOLEAN"
      },
      {
        "name": "bigintcol",
        "mapping": "bigintcol",
        "type": "BIGINT"
      },
      {
        "name": "doublecol",
        "mapping": "doublecol",
        "type": "DOUBLE"
      },
      {
        "name": "smallintcol",
        "mapping": "smallintcol",
        "type": "SMALLINT"
      },
      {
```

```
        "name": "tinyintcol",
        "mapping": "tinyintcol",
        "type": "TINYINT"
      },
      {
        "name": "datecol",
        "mapping": "datecol",
        "type": "DATE",
        "formatHint": "yyyy-MM-dd"
      },
      {
        "name": "timestampcol",
        "mapping": "timestampcol",
        "type": "TIMESTAMP",
        "formatHint": "yyyy-MM-dd HH:mm:ss.SSS"
      }
    ]
  }
}
```

CSV タイプスキーマの例

次の例では、AWS Glue スキーマレジストリに作成されるスキーマは、dataFormat の値として csv を指定し、topicName の値として datatypecsvbulk を使用します。topicName の値は、Kafka のトピック名と同じ大文字と小文字を使用する必要があります。

```
{
  "topicName": "datatypecsvbulk",
  "message": {
    "dataFormat": "csv",
    "fields": [
      {
        "name": "intcol",
        "type": "INTEGER",
        "mapping": "0"
      },
      {
        "name": "varcharcol",
        "type": "VARCHAR",
        "mapping": "1"
      },
      {
        "name": "booleancol",
```

```
        "type": "BOOLEAN",
        "mapping": "2"
      },
      {
        "name": "bigintcol",
        "type": "BIGINT",
        "mapping": "3"
      },
      {
        "name": "doublecol",
        "type": "DOUBLE",
        "mapping": "4"
      },
      {
        "name": "smallintcol",
        "type": "SMALLINT",
        "mapping": "5"
      },
      {
        "name": "tinyintcol",
        "type": "TINYINT",
        "mapping": "6"
      },
      {
        "name": "floatcol",
        "type": "DOUBLE",
        "mapping": "7"
      }
    ]
  }
}
```

Athena MSK コネクタの認証の設定

Amazon MSK クラスターへの認証には、IAM、SSL、SCRAM、スタンドアロン Kafka など、さまざまな方法を使用できます。

次の表は、コネクタの認証タイプと、それぞれのセキュリティプロトコルおよび SASL メカニズムを示しています。詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「Apache Kafka APIの認証と認可」を参照してください。

auth_type	security.protocol	sasl.mechanism
SASL_SSL_PLAIN	SASL_SSL	PLAIN
SASL_PLAINTEXT_PLAIN	SASL_PLAINTEXT	PLAIN
SASL_SSL_AWS_MSK_IAM	SASL_SSL	AWS_MSK_IAM
SASL_SSL_SCRAM_SHA512	SASL_SSL	SCRAM-SHA-512
SSL	SSL	該当なし

> **Note**
>
> SASL_SSL_PLAIN および SASL_PLAINTEXT_PLAIN 認証タイプは、Apache Kafka ではサポートされていますが、Amazon MSK ではサポートされていません。

SASL/IAM

クラスターが IAM 認証を使用する場合、クラスターをセットアップする際にユーザーの IAM ポリシーを設定する必要があります。詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「IAM access control」(IAM アクセスコントロール) を参照してください。

この認証タイプを使用するには、コネクタの auth_type Lambda 環境変数を SASL_SSL_AWS_MSK_IAM に設定します。

SSL

クラスターが SSL 認証されている場合、トラストストアとキーストアファイルを生成して Amazon S3 バケットにアップロードする必要があります。コネクタをデプロイする際に、この Amazon S3 リファレンスを提供する必要があります。キーストア、トラストストア、および SSL キーは AWS Secrets Manager に保存されます。コネクタをデプロイする際に、AWS のシークレットキーを提供します。

Secrets Manager でのシークレットの作成については、「AWS Secrets Manager シークレットを作成する」を参照してください。

この認証タイプを使用するには、以下の表の説明どおりに環境変数を設定します。

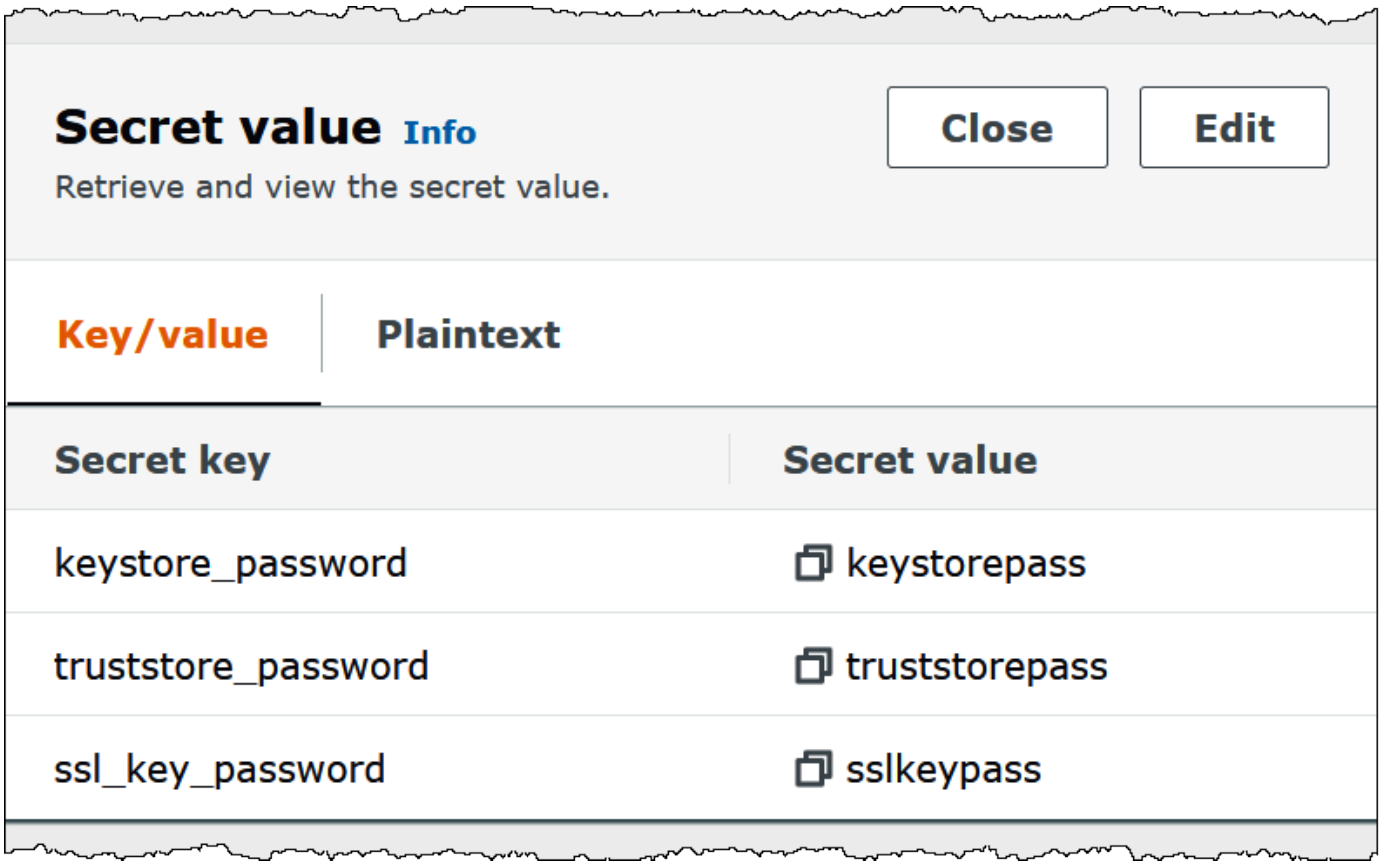
パラメータ	Value
auth_type	SSL
certificates_s3_reference	証明書を格納する Amazon S3 の場所。
secrets_manager_secret	AWS シークレットキーの名前。

Secrets Manager でシークレットを作成した後、Secrets Manager コンソールでそのシークレットを確認できます。

Secrets Manager で自分のシークレットを確認するには

1. Secrets Manager のコンソール (https://console.aws.amazon.com/secretsmanager/) を開きます。
2. ナビゲーションペインで [Secrets] (シークレット) を選択します。
3. [Secrets] (シークレット) ページで、自分のシークレットを選択します。
4. 自分のシークレットの詳細ページで、[Retrieve secret value] (シークレットの値を取得する) を選択します。

   次の画像は、3 組のキー/値のペア (keystore_password、truststore_password、ssl_key_password) を持つシークレットの例を示しています。



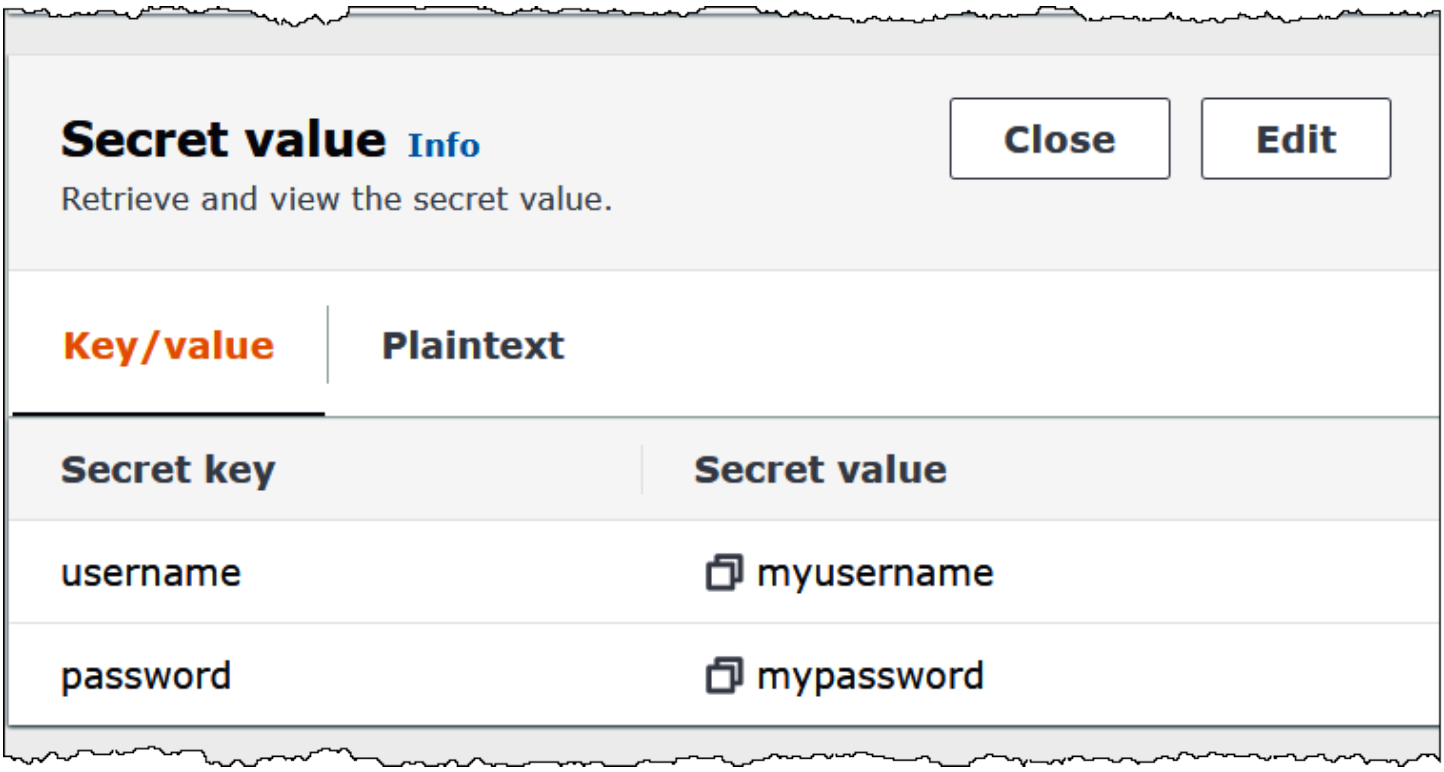
SASL/SCRAM

クラスターが SCRAM 認証を使用している場合は、コネクタをデプロイするときに、クラスターに関連付けられている Secrets Manager キーを指定します。ユーザーの AWS 認証情報 (シークレットキーとアクセスキー) は、クラスターとの認証に使用されます。

下表のように環境変数を設定します。

パラメータ	Value
`auth_type`	`SASL_SSL_SCRAM_SHA512`
`secrets_manager_secret`	AWS シークレットキーの名前。

次の画像は、Secrets Manager コンソールにある 2 つのキー/値のペア (1 つは `username` 用、もう 1 つは `password` 用) のシークレットの例を示しています。



ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena MySQL コネクタ

Amazon Athena Lambda MySQL コネクタは、Amazon Athena での MySQL のデータベースへのアクセスを可能にします。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

- このコネクタを使用する際は、先に VPC とセキュリティグループをセットアップしておきます。詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- Athena はクエリを小文字に変換するため、MySQL テーブル名は小文字である必要があります。たとえば、`myTable` という名前のテーブルに対する Athena のクエリは失敗します。

用語

MySQL コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して MySQL コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
mysql://${jdbc_connection_string}
```

> **Note**
>
> MySQL テーブルに対して SELECT クエリを実行したときに「java.sql.SQLException: Zero date value prohibited」(java.sql.SQLException: ゼロの日付値は禁止されています) というエラーが発生したら、接続文字列に次のパラメータを追加します。
>
> ```
> zeroDateTimeBehavior=convertToNull
> ```
>
> 詳細については、GitHub.com で「Error 'Zero date value prohibited' while trying to select from MySQL table」(MySQL テーブルから選択しようとしたときの「ゼロ日付値は禁止されています」というエラー) を参照してください。

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	MySqlMuxCompositeHandler
メタデータハンドラー	MySqlMuxMetadataHandler
レコードハンドラー	MySqlMuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが mymysqlcatalog の場合、環境変数の名前は mymysqlcatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

mysql1 (デフォルト) と mysql2 の 2 つのデータベースインスタンスをサポートする MySql MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	mysql://jdbc:mysql://mysql2.host:3333/default? user=sample2&password=sample2
mysql_catalog1_connection_string	mysql://jdbc:mysql://mysql1.host:3306/default?${Test/RDS/MySql1}
mysql_catalog2_connection_string	mysql://jdbc:mysql://mysql2.host:3333/default? user=sample2&password=sample2

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

  ```
  {"username": "${username}", "password": "${password}"}
  ```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${Test/RDS/MySql1}` が含まれています。

```
mysql://jdbc:mysql://mysql1.host:3306/default?...&${Test/RDS/MySql1}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
mysql://jdbc:mysql://mysql1host:3306/default?...&user=sample2&password=sample2&...
```

現在、MySQL コネクタは `user` と `password` の JDBC プロパティを認識します。

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の MySQL インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	`MySqlCompositeHandler`
メタデータハンドラー	`MySqlMetadataHandler`
レコードハンドラー	`MySqlRecordHandler`

単一接続ハンドラーのパラメータ

パラメータ	説明
`default`	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、`default` 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の MySQL インスタンス用のプロパティ例を次に示します。

プロパティ	Value
`default`	`mysql://mysql1.host:3306/default?secret=Test/RDS/` MySql1

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。

パラメータ	説明
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC と Arrow に対応するデータ型を示します。

JDBC	Arrow
ブール値	Bit
整数	Tiny
ショート	Smallint
整数	Int
Long	Bigint
フロート	Float4
ダブル	Float8
日付	DateDay
タイムスタンプ	DateMilli
文字列	Varchar
バイト	Varbinary
BigDecimal	10 進数

JDBC	Arrow
配列	リスト

パーティションと分割

パーティションは、コネクタを分割する方法を決定するために使用されます。Athena は `varchar` 型の合成列を作成し、コネクタが分割を生成できるようにするために、テーブルに対するパーティションのスキームを示します。コネクタは実際のテーブル定義を変更しません。

パフォーマンス

MySQL はネイティブパーティションをサポートしています。Athena MySQL コネクタは、これらのパーティションからデータを並列に取得できます。均一なパーティション分散の非常に大きなデータセットをクエリする場合は、ネイティブパーティションを強くお勧めします。

Athena MySQL コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、`LIMIT` 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

`LIMIT N` ステートメントにより、クエリによってスキャンされるデータが削減されます。`LIMIT N` プッシュダウンを使用すると、コネクタは `N` 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの `WHERE` 句内の式です。Athena MySQL コネクタは、これらの式を組み合わせて MySQL に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena MySQL コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

MySQL などのフェデレーテッドクエリのパフォーマンスを向上させるために述語プッシュダウンを使用する方法の記事については、AWS Big Data Blog の「Improve federated queries with predicate pushdown in Amazon Athena」を参照してください。

パススルークエリ

MySQL コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

MySQL でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

次のクエリ例は、MySQL 内のデータソースにクエリをプッシュダウンします。クエリは `customer` テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の MySQL コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Neptune コネクタ

Amazon Neptune は、高速で信頼性に優れたフルマネージド型のグラフデータベースサービスで、高度に接続されたデータセットを使用するアプリケーションの構築と実行を容易にします。Neptune の高性能専用グラフデータベースエンジンは、数十億のリレーションシップを最適に保存し、ミリ秒単位のレイテンシーでグラフをクエリします。詳細については、「Neptune ユーザーガイド」を参照してください。

Amazon Athena Neptune コネクタは、Athena が Neptune グラフデータベースインスタンスとやり取りすることを可能にして、SQL クエリがNeptune グラフデータにアクセスできるようにします。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

前提条件

Neptune コネクタを使用するには、次の 3 つのステップが必要です。

- Neptune クラスターのセットアップ
- AWS Glue Data Catalog のセットアップ
- AWS アカウント へのコネクタのデプロイ。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。Neptune コネクタのデプロイに関するその他の詳細については、GitHub.com で「Deploy the Amazon Athena Neptune Connector」(Amazon Athena Neptune コネクタのデプロイ) を参照してください。

制限事項

現在、Neptune コネクタには以下のような制限があります。

- プライマリキー (ID) を含む列の射影はサポートされていません。

Neptune クラスターのセットアップ

使用したい既存の Amazon Neptune クラスターとプロパティグラフデータセットがない場合は、セットアップする必要があります。

Neptune クラスターをホストする VPC にインターネットゲートウェイと NAT ゲートウェイがあることを確認してください。Neptune コネクタ Lambda 関数が使用するプライベートサブネットには、この NAT ゲートウェイを経由するインターネットへのルートが必要です。Neptune コネクタの Lambda 関数は NAT ゲートウェイを使用して AWS Glue と通信します。

新しい Neptune クラスターをセットアップしてサンプルデータセットとともにロードする手順については、GitHub.com で「Sample Neptune Cluster Setup」(Neptune クラスターのセットアップ例) を参照してください。

AWS Glue Data Catalog のセットアップ

従来のリレーショナルデータストアとは異なり、Neptune グラフ DB ノードとエッジは設定されたスキーマを使用しません。各エントリには、異なるフィールドとデータ型を設定できます。ただし、Neptune コネクタは AWS Glue Data Catalog からメタデータを取得するので、必要なスキーマのテーブルを含む AWS Glue データベースを作成する必要があります。AWS Glue データベースとテーブル作成したら、コネクタは Athena からのクエリに使用できるテーブルのリストに入力できます。

大文字と小文字を区別しない列の照合を有効にする

AWS Glue の列名がすべて小文字である場合でも、Neptune テーブルの列名を正しい大文字と小文字で解決するには、大文字と小文字を区別しない照合を行うように Neptune コネクタを構成できます。

この機能を有効にするには、Neptune コネクタの Lambda 関数で `enable_caseinsensitivematch` 環境変数を `true` に設定します。

大文字と小文字を区別するテーブル名に AWS Glue glabel テーブルパラメータを指定する

AWS Glue は小文字のテーブル名のみをサポートしているため、Neptune 用の AWS Glue テーブルを作成する際には、`glabel` AWS Glue テーブルパラメータを指定することが重要です。

AWS Glue テーブルの定義に `glabel` パラメータを含めて、その値を元の大文字と小文字を区別したテーブル名に設定します。これにより、AWS Glue が Neptune テーブルとやり取りするときに、大文字と小文字が正しく区別された状態が維持されます。次の例では、`glabel` の値をテーブル名 `Airport` に設定します。

```
glabel = Airport
```

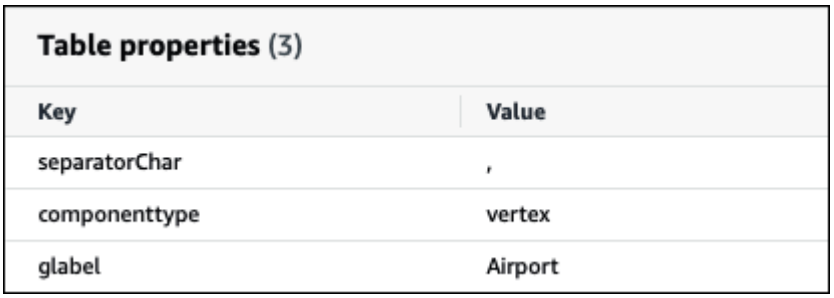


Table properties (3)

Key	Value
separatorChar	,
componenttype	vertex
glabel	Airport

Neptune とのやり取りを考慮して AWS Glue Data Catalog を設定するための詳細については、GitHub.com に掲載されている「Set up AWS Glue Catalog」を参照してください。

パフォーマンス

Athena Neptune コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。ただし、プライマリキーを使用する述語はクエリに失敗します。LIMIT 句はスキャンされるデータ量を減らしますが、述語が提供されない場合は、LIMIT 句を含む SELECT クエリは少なくとも 16 MB のデータをスキャンすることを想定する必要があります。Neptune コネクタは、同時実行によるスロットリングに強いです。

パススルークエリ

Neptune コネクタは、パススルークエリをサポートします。この機能を使用すると、プロパティグラフで Gremlin クエリを実行したり、RDF データに対して SPARQL クエリを実行したりできます。

Neptune でパススルークエリを作成するには、以下の構文を使用します。

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'database_name',
            COLLECTION => 'collection_name',
            QUERY => 'query_string'
        ))
```

次の Neptune パススルークエリの例では、コード ATL の空港がフィルタリングされています。

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'graph-database',
            COLLECTION => 'airport',
```

```
            QUERY => 'g.V().has(\"airport\", \"code\", \"ATL\").valueMap()',
        ))
```

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena OpenSearch コネクタ

OpenSearch Service

Amazon Athena OpenSearch コネクタは、Amazon Athena でのOpenSearch インスタンスとの通信を可能にし、OpenSearch データを SQL でクエリできるようにします。

> **Note**
>
> 既知の問題により、OpenSearch コネクタは VPC では使用できません。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

用語

OpenSearch コネクタに関連する用語を次に示します。

- ドメイン – このコネクタが OpenSearch インスタンスのエンドポイントに関連付ける名前。ドメインはデータベース名としても使用されます。Amazon OpenSearch Service 内で定義されている OpenSearch インスタンスの場合、ドメインは自動検出可能です。他のインスタンスでは、ドメイン名とエンドポイント間のマッピングを指定する必要があります。
- インデックス – OpenSearch インスタンスで定義されているデータベーステーブル。

- マッピング – インデックスがデータベーステーブルの場合、マッピングはそのスキーマ (すなわち、フィールドと属性の定義) です。

  このコネクタは、OpenSearch インスタンスからのメタデータ取得と AWS Glue Data Catalog からのメタデータ取得の両方をサポートします。OpenSearch ドメインとインデックス名に一致する AWS Glue データベースとテーブルをコネクタが検出した場合、コネクタはそれらをスキーマ定義に使用しようとします。AWS Glue テーブルは、OpenSearch インデックスのすべてのフィールドのスーパーセットになるように作成することをお勧めします。
- ドキュメント – データベーステーブル内のレコード。
- データストリーム — 複数のバッキングインデックスで構成される時間ベースのデータ。詳細については、OpenSearch ドキュメントの「Data Streams」および、「Amazon OpenSearch Service 開発者ガイド」の「Data Streams の使用開始」を参照してください。

> **Note**
>
> データストリームインデックスは OpenSearch によって内部で作成管理されるため、コネクタは最初の使用可能なインデックスからスキーママッピングを選択します。このため、補足メタデータのソースとして AWS Glue テーブルを設定することを強く推奨します。詳細については、「AWS Glue でのデータベースとテーブルのセットアップ」を参照してください。

パラメータ

このセクションの Lambda 環境変数を使用して OpenSearch コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS

が生成したより強力な暗号化キー (たとえば a7e63k4b-8loc-40db-a2a1-4d0en2cd8331) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。

- disable_spill_encryption – (オプション) True に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は False です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- disable_glue – (オプション) これが存在し、true に設定されている場合、コネクタは AWS Glue からの補足メタデータ取得は試みません。
- query_timeout_cluster – 並列スキャンの生成に使用されるクラスターヘルスクエリのタイムアウト時間 (秒単位)。
- query_timeout_search – インデックスからのドキュメントの取得に使用される検索クエリのタイムアウト時間 (秒単位)。
- auto_discover_endpoint – ブール値。デフォルト: true。Amazon OpenSearch Service を使用する際に、このパラメータを true に設定すると、コネクタは OpenSearch Service で適切な describe API オペレーションまたは list API オペレーションを呼び出すことで、ドメインとエンドポイントを自動検出できます。その他のタイプの OpenSearch インスタンス (セルフホスト型など) では、関連するドメインエンドポイントを domain_mapping 変数に指定する必要があります。auto_discover_endpoint=true の場合、コネクタは AWS 認証情報を使用して OpenSearch Service に対する認証を行います。それ以外の場合、コネクタは domain_mapping 変数を通じて AWS Secrets Manager からユーザー名とパスワードの認証情報を取得します。
- domain_mapping – auto_discover_endpoint が false に設定されている場合にのみ使用され、ドメイン名とそれに関連付けられているエンドポイントとのマッピングを定義します。domain_mapping 変数は、次の形式の複数の OpenSearch エンドポイントに対応できます。

```
domain1=endpoint1,domain2=endpoint2,domain3=endpoint3,...
```

OpenSearch エンドポイントへの認証を目的として、コネクタは AWS Secrets Manager から取得したユーザー名とパスワードを含む ${SecretName}: の形式を使用して挿入された置換文字列をサポートします。式の最後のコロン (:) は、エンドポイントの残りの部分との区切り文字として機能します。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets

Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

次の例では、opensearch-creds シークレットを使用しています。

```
movies=https://${opensearch-creds}:search-movies-ne...qu.us-east-1.es.amazonaws.com
```

実行時には、次の例のように、${opensearch-creds} はユーザー名とパスワードとして表示されます。

```
movies=https://myusername@mypassword:search-movies-ne...qu.us-east-1.es.amazonaws.com
```

domain_mapping パラメータでは、ドメインとエンドポイントのペアごとに異なるシークレットを使用できます。シークレット自体は#####@#####の形式で指定する必要があります。パスワードに @ 記号が含まれている場合がありますが、最初の @ が#####との区切りになります。

カンマ (,) と等号 (=) がこのコネクタによってドメインとエンドポイントのペアの区切り文字として使用されていることにも注意してください。そのため、これらの文字を保存されたシークレットの中で使用するべきではありません。

AWS Glue でのデータベースとテーブルのセットアップ

コネクタは、AWS Glue または OpenSearch を使用してメタデータ情報を取得します。AWS Glue テーブルを補足メタデータ定義のソースとしてセットアップできます。この機能を有効にするには、補足するソースのドメインとインデックスに一致する AWS Glue データベースとテーブルを定義します。コネクタは、指定されたインデックスのマッピングを取得することで、OpenSearch インスタンスに保存されているメタデータ定義を利用することもできます。

OpenSearch での配列のメタデータの定義

OpenSearch には専用の配列データ型はありません。データ型が同じであれば、どのフィールドにもゼロ以上の値を含められます。OpenSearch をメタデータ定義のソースとして使用する場合は、リストまたは配列と見なされるフィールドの Athena で使用されるすべてのインデックスの _meta プロパティを定義する必要があります。このステップを完了しなかった場合、クエリはリストフィールドの最初の要素のみを返します。_meta プロパティを指定すると、フィールド名はネストされた

JSON 構造に対して完全修飾である必要があります (たとえば、street が address ストラクチャの内部にネストされたフィールドである場合、address.street)。

次の例では、movies テーブル内で actor リストと genre リストを定義しています。

```
PUT movies/_mapping
{
  "_meta": {
    "actor": "list",
    "genre": "list"
  }
}
```

データ型

OpenSearch コネクタは、AWS Glue と OpenSearch インスタンスのどちらからでもメタデータの定義を抽出できます。コネクタは、次の表のマッピングを使用して定義を Apache Arrow データ型に変換します。これには、次のセクションで説明する点が含まれます。

OpenSearch	Apache Arrow	AWS Glue
text、keyword、binary	VARCHAR	string
long	BIGINT	bigint
scaled_float	BIGINT	SCALED_FLOAT(...)
integer	INT	整数
short	SMALLINT	smallint
バイト	TINYINT	tinyint
double	FLOAT8	double
float、half_float	FLOAT4	float
ブール値	BIT	ブール値
date、date_nanos	DATEMILLI	timestamp
JSON の構造	STRUCT	STRUCT

OpenSearch	Apache Arrow	AWS Glue
_meta (詳細については、「OpenSearch での配列のメタデータの定義」セクションを参照してください)	LIST	配列

データ型に関する注意事項

- 現在、コネクタは OpenSearch と前の表にリストされている AWS Glue データタイプのみをサポートしています。
- `scaled_float` は、浮動小数点数を固定の double 型のスケーリング係数でスケーリングしたもので、Apache Arrow では `BIGINT` として表されます。たとえば、0.756 は、スケーリング係数が 100 の場合、四捨五入されて 76 になります。
- AWS Glue で `scaled_float` を定義するには、列に `array` 型を選択し、SCALED_FLOAT(########) という形式を使用してフィールドを宣言する必要があります。

  有効な例を次に示します。

```
SCALED_FLOAT(10.51)
SCALED_FLOAT(100)
SCALED_FLOAT(100.0)
```

  有効でない例を次に示します。

```
SCALED_FLOAT(10.)
SCALED_FLOAT(.5)
```

- `date_nanos` から `DATEMILLI` に変換する場合、ナノ秒は最も近いミリ秒に四捨五入されます。`date` と `date_nanos` の有効な例には、次の形式が含まれますが、これらに限定されません。

```
"2020-05-18T10:15:30.123456789"
"2020-05-15T06:50:01.123Z"
"2020-05-15T06:49:30.123-05:00"
1589525370001 (epoch milliseconds)
```

- OpenSearch `binary` は、`Base64` を使用してエンコードされたバイナリ値の文字列表現で、`VARCHAR` に変換されます。

SQL クエリの実行

このコネクタで使用できる DDL クエリの例を次に示します。これらの例では、*function_name* は Lambda 関数の名前に対応し、*domain* はクエリするドメインの名前で、*index* はインデックスの名前です。

```
SHOW DATABASES in `lambda:function_name`
```

```
SHOW TABLES in `lambda:function_name`.domain
```

```
DESCRIBE `lambda:function_name`.domain.index
```

パフォーマンス

Athena OpenSearch コネクタは、シャードベースの並列スキャンをサポートしています。コネクタは、OpenSearch インスタンスから取得したクラスターヘルス情報を使用して、ドキュメント検索クエリに対する複数のリクエストを生成します。リクエストはシャードごとに分割され、同時に実行されます。

また、コネクタはドキュメント検索クエリの一部として述語をプッシュダウンします。次のクエリと述語の例は、コネクタがどのように述語プッシュダウンを使用するかを示しています。

Query

```
SELECT * FROM "lambda:elasticsearch".movies.movies
WHERE year >= 1955 AND year <= 1962 OR year = 1996
```

述語

```
(_exists_:year) AND year:([1955 TO 1962] OR 1996)
```

パススルークエリ

OpenSearch コネクタは、パススルークエリをサポートし、Query DSL 言語を使用します。Query DSL を使用したクエリの詳細については、Elasticsearch ドキュメントの「Query DSL」、または OpenSearch ドキュメントの「Query DSL」を参照してください。

OpenSearch コネクタでパススルークエリを使用するには、以下の構文を使用します。

```
SELECT * FROM TABLE(
        system.query(
            schema => 'schema_name',
            index => 'index_name',
            query => "{query_string}"
        ))
```

以下の OpenSearch パススルークエリ例は、default スキーマの employee インデックス内でアクティブ雇用ステータスを持つ従業員をフィルタリングします。

```
SELECT * FROM TABLE(
        system.query(
            schema => 'default',
            index => 'employee',
            query => "{ ''bool'':{''filter'':{''term'':{''status'': ''active''}}}}"
        ))
```

追加リソース

- Amazon Athena OpenSearch コネクタを使用して、1 つのクエリで Amazon OpenSearch Service および Amazon S3 のデータをクエリする方法の記事については、AWS Big Data Blog の「Query data in Amazon OpenSearch Service using SQL from Amazon Athena」を参照してください。
- このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Oracle コネクタ

Oracle 用の Amazon Athena コネクタを使用すると、Amazon Athena で オンプレミスや、Amazon EC2、Amazon RDS 上で稼働する Oracle に保存されたデータに SQL クエリを実行できます。このコネクタを使用して Oracle Exadata のデータにクエリを実行することもできます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

用語

Oracle コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して Oracle コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
oracle://${jdbc_connection_string}
```

> **Note**
>
> パスワードに特殊文字が含まれている場合は (例: `some.password`)、パスワードを接続文字列に渡すときにそのパスワードを二重引用符で囲みます (例: `"some.password"`)。これを実行しない場合、「無効な Oracle URL が指定されました」というエラーが発生する可能性があります。

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`OracleMuxCompositeHandler`
メタデータハンドラー	`OracleMuxMetadataHandler`
レコードハンドラー	`OracleMuxRecordHandler`

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが `myoraclecatalog` の場合、環境変数の名前は `myoraclecatalog_connection_string` になります。

パラメータ	説明
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

oracle1 (デフォルト) と oracle2 の 2 つのデータベースインスタンスをサポートする Oracle MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	oracle://jdbc:oracle:thin:${Test/RDS/Oracle1}@//oracle1.hostname:port/servicename
oracle_catalog1_connection_string	oracle://jdbc:oracle:thin:${Test/RDS/Oracle1}@//oracle1.hostname:port/servicename
oracle_catalog2_connection_string	oracle://jdbc:oracle:thin:${Test/RDS/Oracle2}@//oracle2.hostname:port/servicename

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の

「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

> **Note**
>
> パスワードに特殊文字が含まれている場合は (例: `some.password`)、Secrets Manager にパスワードを保存するときにそのパスワードを二重引用符で囲みます (例: `"some.password"`)。これを実行しない場合、「無効な Oracle URL が指定されました」というエラーが発生する可能性があります。

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${Test/RDS/Oracle}` が含まれています。

```
oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@//hostname:port/servicename
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
oracle://jdbc:oracle:thin:username/password@//hostname:port/servicename
```

現在、Oracle コネクタは `UID` と `PWD` の JDBC プロパティを認識します。

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Oracle インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	`OracleCompositeHandler`
メタデータハンドラー	`OracleMetadataHandler`
レコードハンドラー	`OracleRecordHandler`

単一接続ハンドラーのパラメータ

パラメータ	説明
`default`	必須。デフォルトの接続文字列。
`IsFIPSEnabled`	オプション。FIPS モードが有効になっている場合は、`true` に設定します。デフォルト: `false`。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、`default` 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

コネクタは Amazon RDS インスタンスへの SSL ベースの接続をサポートします。このサポートは、Transport Layer Security (TLS) プロトコルと、クライアントによるサーバーの認証に限定されています。相互認証は Amazon RDS ではサポートされていません。下の表の 2 行目は、SSL を使用するための構文を示しています。

Lambda 関数でサポートされる単一の Oracle インスタンス用のプロパティ例を次に示します。

プロパティ	Value
`default`	`oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@//hostname:port/servicename`

プロパティ	Value
	`oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS) (HOST=<HOST_NAME>)(PORT=))(CONNECT_DATA=(SID=))(SECURITY=(SSL_SERVER_CERT_DN=)))`

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス
`spill_put_request_headers`	(オプション) スピルに使用される Amazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC、Oracle、Arrow に対応するデータ型を示します。

JDBC	Oracle	Arrow
ブール値	ブール値	Bit
整数	該当なし	Tiny
ショート	smallint	Smallint

JDBC	Oracle	Arrow
整数	integer	Int
Long	bigint	Bigint
フロート	float4	Float4
ダブル	float8	Float8
日付	date	DateDay
タイムスタンプ	timestamp	DateMilli
文字列	text	Varchar
バイト	bytes	Varbinary
BigDecimal	numeric(p,s)	10 進数
配列	該当なし (注記を参照)	リスト

パーティションと分割

パーティションは、コネクタを分割する方法を決定するために使用されます。Athena は `varchar` 型の合成列を作成し、コネクタが分割を生成できるようにするために、テーブルに対するパーティションのスキームを示します。コネクタは実際のテーブル定義を変更しません。

パフォーマンス

Oracle はネイティブパーティションをサポートしています。Athena Oracle コネクタは、これらのパーティションからデータを並列に取得できます。均一なパーティション分散の非常に大きなデータセットをクエリする場合は、ネイティブパーティションを強くお勧めします。列のサブセットを選択すると、クエリランタイムが大幅に短縮され、スキャンされるデータが減ります。Oracle コネクタは、同時実行によるスロットリングに強いです。ただし、クエリランタイムは長くなる傾向があります。

Athena Oracle コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。単純な述語と複雑な式はコネクタにプッシュダウンされるため、スキャンされるデータ量が減少し、クエリ実行のランタイムが短縮されます。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena Oracle コネクタは、これらの式を組み合わせて Oracle に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Oracle コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値:
  EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

パススルークエリ

Oracle コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Oracle でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Oracle 内のデータソースにクエリをプッシュダウンします。クエリは、customer テーブル内のすべての列を選択します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の Oracle コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena PostgreSQL コネクタ

Amazon Athena PostgreSQL コネクタは、Athena での PostgreSQL データベースへのアクセスを可能にします。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

用語

PostgreSQL コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して PostgreSQL コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
postgres://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	PostGreSqlMuxCompositeHandler
メタデータハンドラー	PostGreSqlMuxMetadataHandler
レコードハンドラー	PostGreSqlMuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが mypostgrescatalog の場合、環境変数の名前は mypostgrescatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

postgres1 (デフォルト) と postgres2 の 2 つのデータベースインスタンスをサポートする PostgreSql MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	postgres://jdbc:postgresql://postgres1.host:5432/default?${Test/RDS/PostGres1}
postgres_catalog1_connection_string	postgres://jdbc:postgresql://postgres1.host:5432/default?${Test/RDS/PostGres1}
postgres_catalog2_	postgres://jdbc:postgresql://postgres2.host:5432/default?user=sample&password=sample

プロパティ	Value
connectio n_string	

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

  ```
  {"username": "${username}", "password": "${password}"}
  ```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${Test/RDS/PostGres1}` が含まれています。

```
postgres://jdbc:postgresql://postgres1.host:5432/default?...&${Test/RDS/PostGres1}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
postgres://jdbc:postgresql://postgres1.host:5432/
default?...&user=sample2&password=sample2&...
```

現在、PostgreSQL コネクタは `user` と `password` の JDBC プロパティを認識します。

SSL を有効にしています

PostgreSQL 接続で SSL をサポートするには、接続文字列に以下を追加します。

```
&sslmode=verify-ca&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory
```

例

次の接続文字列の例は SSL を使用していません。

```
postgres://jdbc:postgresql://example-asdf-aurora-postgres-endpoint:5432/asdf?
user=someuser&password=somepassword
```

SSL を有効にするには、文字列を次のように変更します。

```
postgres://jdbc:postgresql://example-asdf-aurora-postgres-
endpoint:5432/asdf?user=someuser&password=somepassword&sslmode=verify-
ca&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory
```

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の PostgreSQL インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	`PostGreSqlCompositeHandler`

ハンドラーのタイプ	Class
メタデータハンドラー	`PostGreSqlMetadataHandler`
レコードハンドラー	`PostGreSqlRecordHandler`

単一接続ハンドラーのパラメータ

パラメータ	説明
`default`	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、`default` 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の PostgreSQL インスタンス用のプロパティ例を次に示します。

プロパティ	Value
`default`	`postgres://jdbc:postgresql://postgres1.host:5432/default?secret=${Test/RDS/PostgreSQL1}`

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス
`spill_put_request_headers`	(オプション) スピルに使用される Amazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encrypt`

パラメータ	説明
	ion" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC、PostgreSQL、Arrow に対応するデータ型を示します。

JDBC	PostgreSQL	Arrow
ブール値	ブール値	Bit
整数	該当なし	Tiny
ショート	smallint	Smallint
整数	integer	Int
Long	bigint	Bigint
フロート	float4	Float4
ダブル	float8	Float8
日付	date	DateDay
タイムスタンプ	timestamp	DateMilli
文字列	text	Varchar
バイト	bytes	Varbinary
BigDecimal	numeric(p,s)	10 進数
配列	該当なし (注記を参照)	リスト

> **Note**
>
> ARRAY 型は PostgreSQL コネクタでサポートされていますが、多次元配列 (*<data_type>*[][] またはネストされた配列) はサポートされていないという制約があります。サポートされていない ARRAY データ型の列は文字列要素の配列 (array<varchar>) に変換されます。

パーティションと分割

パーティションは、コネクタを分割する方法を決定するために使用されます。Athena は varchar 型の合成列を作成し、コネクタが分割を生成できるようにするために、テーブルに対するパーティションのスキームを示します。コネクタは実際のテーブル定義を変更しません。

パフォーマンス

PostgreSQL はネイティブパーティションをサポートしています。Athena PostgreSQL コネクタは、これらのパーティションからデータを並列に取得できます。均一なパーティション分散の非常に大きなデータセットをクエリする場合は、ネイティブパーティションを強くお勧めします。

Athena PostgreSQL コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、LIMIT 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。ただし、列のサブセットを選択すると、クエリのランタイムが長くなる場合があります。

LIMIT 句

LIMIT N ステートメントにより、クエリによってスキャンされるデータが削減されます。LIMIT N プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena PostgreSQL コネクタは、これらの式を組み合わせて PostgreSQL に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena PostgreSQL コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT

- 等値:
  EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

パススルークエリ

PostgreSQL コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

PostgreSQL でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、PostgreSQL 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の PostgreSQL コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Redis コネクタ

Amazon Athena Redis コネクタは、Amazon Athena が Redis インスタンスとやり取りすることを可能にして、Redis データを SQL でクエリできるようにします。AWS Glue Data Catalog を使用して、Redis のキーと値のペアを仮想テーブルにマッピングできます。

従来のリレーショナルデータストアとは異なり、Redis にはテーブルや列の概念がありません。代わりに、Redis にはキーと値のアクセスパターンが用意されています。キーは基本的に `string` で、値は `string`、`z-set`、または `hmap` です。

AWS Glue Data Catalog を使用して、スキーマを作成し、仮想テーブルを設定します。特別なテーブルプロパティによって、Redis のキーと値をテーブルにマッピングする方法が Athena Redis コネクタに伝えられます。詳細については、このドキュメントで後述する「AWS Glue でのデータベースとテーブルのセットアップ」を参照してください。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

Amazon Athena Redis コネクタは Amazon MemoryDB for Redis と Amazon ElastiCache for Redis をサポートしています。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。
- このコネクタを使用する際は、先に VPC とセキュリティグループをセットアップしておきます。詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。

パラメータ

このセクションの Lambda 環境変数を使用して Redis コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。

- spill_prefix – (オプション) 指定された athena-federation-spill という spill_bucket の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば a7e63k4b-8loc-40db-a2a1-4d0en2cd8331) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) True に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は False です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- glue_catalog – (オプション) クロスアカウントの AWS Glue カタログを指定するために、このオプションを使用します。デフォルトでは、コネクタは自身の AWS Glue アカウントからメタデータを取得しようとします。

AWS Glue でのデータベースとテーブルのセットアップ

AWS Glue テーブルを Redis で使用できるようにするには、テーブルに redis-endpoint および redis-value-type、さらに redis-keys-zset または redis-key-prefix のテーブルプロパティを設定します。

さらに、Redis テーブルを含む AWS Glue データベースには、データベースの URI プロパティに redis-db-flag がある必要があります。redis-db-flag URI プロパティ設定するには、AWS Glue コンソールでデータベースを編集します。

次のリストは、テーブルプロパティについての説明です。

- redis-endpoint – (必須) このテーブルのデータを含む Redis サーバーの####:###:##### (例: athena-federation-demo.cache.amazonaws.com:6379)。または、${*Secret_Name*} をテーブルプロパティ値として使用して、エンドポイント、またはエンドポイントの一部を AWS Secrets Manager に保存することもできます。

> **Note**
>
> Athena 横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

- redis-keys-zset – (`redis-key-prefix` が使用されていない場合は必須) 値が zset であるキーのカンマ区切りリスト (例: `active-orders,pending-orders`)。zset 内のそれぞれの値は、テーブルの一部であるキーとして扱われます。`redis-keys-zset` プロパティまたは `redis-key-prefix` プロパティのいずれかを設定する必要があります。
- redis-key-prefix – (`redis-keys-zset` が使用されていない場合は必須) テーブル内の値をスキャンするキープレフィックスのカンマ区切りリスト (例: `accounts-*,acct-`)。`redis-key-prefix` プロパティまたは `redis-keys-zset` プロパティのいずれかを設定する必要があります。
- redis-value-type – (必須) `redis-key-prefix` または `redis-keys-zset` のいずれかをテーブルにマッピングすることによって、キーの値がどのように定義されるかを定義します。リテラルは単一の列にマッピングされます。また、zset は単一の列にマッピングされますが、各キーには多数の行を格納できます。ハッシュを使用すると、各キーを複数の列 (ハッシュ、リテラル、zset など) から成る行にすることができます。
- redis-ssl-flag – (オプション) `True` の場合、SSL/TLS を使用する Redis 接続を作成します。デフォルト: `False`。
- edis-cluster-flag – (オプション) `True` の場合、クラスター化された Redis インスタンスのサポートを有効にします。デフォルト: `False`。
- redis-db-number – (オプション) スタンドアロンのクラスター化されていないインスタンスにのみ適用されます。デフォルト以外の Redis データベースから読み込むには、この番号 (1、2、3 など) を設定します。デフォルトは Redis 論理データベース 0 です。この番号は、Athena や AWS Glue のデータベースではなく、Redis の論理データベースを示すものです。詳細については、Redis ドキュメントの「SELECT index」を参照してください。

データ型

Redis コネクタは、次のデータ型に対応しています。Redis ストリームはサポートされていません。

- 文字列
- ハッシュ

- ソート済みセット (ZSet)

すべての Redis 値は `string` データ型として取得されます。次に、テーブルが AWS Glue Data Catalog でどのように定義されているかに基づいて、次の Apache Arrow データ型のいずれかに変換されます。

AWS Glue データ型	Apache Arrow データ型
整数	INT
string	VARCHAR
bigint	BIGINT
double	FLOAT8
フロート	FLOAT4
smallint	SMALLINT
tinyint	TINYINT
ブール値	BIT
バイナリ	VARBINARY

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-redis.yaml ファイルの `Policies` セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- AWS Glue Data Catalog – Redis コネクタには、スキーマ情報を取得するために AWS Glue Data Catalog への読み込み専用アクセス権が必要です。

- CloudWatch Logs – コネクタは、ログを保存するために CloudWatch Logs にアクセスする必要があります。
- AWS Secrets Manager 読み込みアクセス – Redis エンドポイントの詳細を Secrets Manager に保存する場合は、それらのシークレットへのアクセスをコネクタに許可する必要があります。
- VPC アクセス – コネクタには、VPC に接続して Redis インスタンスと通信できるように、VPC にインターフェイスをアタッチおよびデタッチできることが必要です。

パフォーマンス

Athena Redis コネクタは、定義したテーブルのタイプ (zset キーやプレフィックスキーなど) に従って、Redis インスタンスに対するクエリを並列化しようとします。

Athena Redis コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。ただし、プライマリキーに対する述語を含むクエリはタイムアウトで失敗します。`LIMIT` 句はスキャンされるデータ量を減らしますが、述語を提供しない場合、`LIMIT` 句を含む `SELECT` クエリは少なくとも 16 MB のデータをスキャンすることを想定する必要があります。Redis コネクタは同時実行によるスロットリングに強いです。

ライセンス情報

Amazon Athena Redis コネクタプロジェクトは、Apache-2.0 License でライセンスされています。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Redshift コネクタ

Amazon Athena Redshift コネクタは、Amazon Athena が Amazon Redshift データベースと Amazon Redshift Serverless データベース (Redshift Serverless ビューを含む) にアクセスすることを可能にします。どちらのサービスにも、このページで説明されている JDBC 接続文字列設定を使用して接続できます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- Redshift は外部パーティションをサポートしていないため、クエリで指定されたすべてのデータが毎回取得されます。

用語

Redshift コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して Redshift コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
redshift://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`RedshiftMuxCompositeHandler`
メタデータハンドラー	`RedshiftMuxMetadataHandler`
レコードハンドラー	`RedshiftMuxRecordHandler`

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが `myredshiftcatalog` の場合、環境変数の名前は `myredshiftcatalog_connection_string` になります。
`default`	必須。デフォルトの接続文字列。この文字列は、カタログが `lambda:${ AWS_LAMBDA_FUNCTION_NAME }` の場合に使用されます。

`redshift1` (デフォルト) と `redshift2` の 2 つのデータベースインスタンスをサポートする Redshift MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	redshift://jdbc:redshift://redshift1.host:5439/dev?user=sample2&password=sample2
redshift_catalog1_connection_string	redshift://jdbc:redshift://redshift1.host:3306/default?${Test/RDS/Redshift1}
redshift_catalog2_connection_string	redshift://jdbc:redshift://redshift2.host:3333/default?user=sample2&password=sample2

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列はシークレット名 ${Test/RDS/ Redshift1} を含んでいます。

```
redshift://jdbc:redshift://redshift1.host:3306/default?...&${Test/RDS/Redshift1}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
redshift://jdbc:redshift://redshift1.host:3306/
default?...&user=sample2&password=sample2&...
```

現在、Redshift コネクタは `user` と `password` の JDBC プロパティを認識します。

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス
`spill_put_request_headers`	(オプション) スピルに使用される Amazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC と Apache Arrow に対応するデータ型を示します。

JDBC	Arrow
ブール値	Bit
整数	Tiny
ショート	Smallint
整数	Int
Long	Bigint
フロート	Float4
ダブル	Float8
日付	DateDay
タイムスタンプ	DateMilli
文字列	Varchar
バイト	Varbinary
BigDecimal	10 進数
配列	リスト

パーティションと分割

Redshift は外部パーティションをサポートしていません。パフォーマンス関連の問題については、「パフォーマンス」を参照してください。

パフォーマンス

Athena Redshift コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮する

ために、LIMIT 句、ORDER BY 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。ただし、列のサブセットを選択すると、クエリのランタイムが長くなる場合があります。Amazon Redshift は、複数のクエリを同時に実行すると、特にクエリの実行速度が低下しやすくなります。

LIMIT 句

LIMIT N ステートメントにより、クエリによってスキャンされるデータが削減されます。LIMIT N プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

上位 N 件のクエリ

上位 N 件のクエリは、結果セットの順序と返される行数に対する制限を指定します。このタイプのクエリを使用して、データセットの上位 N 個の最大値または上位 N 個の最小値を決定できます。上位 N 件のプッシュダウンを使用すると、コネクタは N 件の順序付けられた行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena Redshift コネクタは、これらの式を組み合わせて Redshift に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Redshift コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値:
  EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
```

```
ORDER BY col_a DESC
LIMIT 10;
```

Amazon Redshift などのフェデレーテッドクエリのパフォーマンスを向上させるために述語プッシュダウンを使用する方法の記事については、AWS Big Data Blog の「Improve federated queries with predicate pushdown in Amazon Athena」を参照してください。

パススルークエリ

Redshift コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Redshift でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Redshift 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の Redshift コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena SAP HANA コネクタ

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または

「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- SAP HANA では、オブジェクト名が SAP HANA データベースに保存される際に大文字に変換されます。ただし、引用符で囲まれた名前は大文字と小文字が区別されるため、2 つのテーブルに同じ名前が小文字と大文字で存在する場合があります (たとえば、`EMPLOYEE` と `employee`)。

  Athena の横串検索では、スキーマのテーブル名が小文字で Lambda 関数に渡されます。この問題を回避するには、クエリヒント `@schemaCase` を指定して、大文字と小文字が区別される名前のテーブルからデータを取得します。クエリヒントを含む 2 つのサンプルクエリを次に示します。

```
SELECT *
FROM "lambda:saphanaconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=upper"
```

```
SELECT *
FROM "lambda:saphanaconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=lower"
```

用語

SAP HANA コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。

- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して SAP HANA コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
saphana://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`SaphanaMuxCompositeHandler`
メタデータハンドラー	`SaphanaMuxMetadataHandler`
レコードハンドラー	`SaphanaMuxRecordHandler`

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが mysaphanacatalog の場合、環境変数の名前は mysaphanacatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

saphana1 (デフォルト) と saphana2 の 2 つのデータベースインスタンスをサポートする Saphana MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}
saphana_catalog1_connection_string	saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}
saphana_catalog2_connection_string	saphana://jdbc:sap://saphana2.host:port/?user=sample2&password=sample2

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

  ```
  {"username": "${username}", "password": "${password}"}
  ```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${Test/RDS/Saphana1}` が含まれています。

```
saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
saphana://jdbc:sap://saphana1.host:port/?user=sample2&password=sample2&...
```

現在、SAP HANA コネクタは `user` と `password` の JDBC プロパティを認識します。

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の SAP HANA インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	`SaphanaCompositeHandler`
メタデータハンドラー	`SaphanaMetadataHandler`
レコードハンドラー	`SaphanaRecordHandler`

単一接続ハンドラーのパラメータ

パラメータ	説明
`default`	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、`default` 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の SAP HANA インスタンス用のプロパティ例を次に示します。

プロパティ	Value
`default`	`saphana://jdbc:sap://saphana1.host:port/?secret=Test/RDS/Saphana1`

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
spill_bucket	必須。スピルバケット名。
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC と Apache Arrow に対応するデータ型を示します。

JDBC	Arrow
ブール値	Bit
整数	Tiny
ショート	Smallint
整数	Int
Long	Bigint
フロート	Float4
ダブル	Float8
日付	DateDay
タイムスタンプ	DateMilli
文字列	Varchar

JDBC	Arrow
バイト	Varbinary
BigDecimal	10 進数
配列	リスト

データ型変換

JDBC から Arrow への変換に加えて、コネクタは特定の別の変換を実行して SAP HANA ソースと Athena データ型との互換性を保ちます。これらの変換は、クエリを正常に実行するのに役立ちます。次の表に、これらの変換を示します。

ソースデータタイプ (SAP HANA)	変換されたデータ型 (Athena)
DECIMAL	BIGINT
INTEGER	INT
DATE	DATEDAY
TIMESTAMP	DATEMILLI

その他のサポートされていないデータ型はすべて VARCHAR に変換されます。

パーティションと分割

パーティションは、Integer 型の単一パーティション列で表されます。この列には、SAP HANA テーブルで定義されているパーティションのパーティション名が含まれます。パーティション名のないテーブルの場合、* が返されます。これは 1 つのパーティションと同じです。パーティションはスプリットと同等です。

名前	型	説明
PART_ID	整数	SAP HANA の名前付きパーティション。

パフォーマンス

SAP HANA はネイティブパーティションをサポートしています。Athena SAP HANA コネクタは、これらのパーティションからデータを並列に取得できます。均一なパーティション分散の非常に大きなデータセットをクエリする場合は、ネイティブパーティションを強くお勧めします。列のサブセットを選択すると、クエリランタイムが大幅に短縮され、スキャンされるデータが減ります。コネクタでは、同時実行が原因で大幅なスロットリングが発生し、場合によってはクエリに失敗することもあります。

Athena SAP HANA コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、LIMIT 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

LIMIT N ステートメントにより、クエリによってスキャンされるデータが削減されます。LIMIT N プッシュダウンを使用すると、コネクタは N 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena SAP HANA コネクタは、これらの式を組み合わせて SAP HANA に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena SAP HANA コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
```

```
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

パススルークエリ

SAP HANA コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

SAP HANA でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、SAP HANA 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の SAP HANA コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Snowflake コネクタ

Snowflake 用の Amazon Athena コネクタを使用すると、Amazon Athena で Snowflake SQL または RDS インスタンスに保存されたデータに対して JDBC を使用して SQL クエリを実行できます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- 現在、単一スプリットの Snowflake ビューがサポートされています。
- Snowflake では、オブジェクト名は大文字と小文字が区別されるため、2 つのテーブルに同じ名前が小文字と大文字で存在する場合があります (たとえば、EMPLOYEE と employee)。Athena の横串検索では、スキーマのテーブル名が小文字で Lambda 関数に渡されます。この問題を回避するには、クエリヒント @schemaCase を指定して、大文字と小文字が区別される名前のテーブルからデータを取得します。クエリヒントを含む 2 つのサンプルクエリを次に示します。

```
SELECT *
FROM "lambda:snowflakeconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=upper"
```

```
SELECT *
FROM "lambda:snowflakeconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=lower"
```

用語

Snowflake コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。

- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して Snowflake コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
snowflake://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	`SnowflakeMuxCompositeHandler`
メタデータハンドラー	`SnowflakeMuxMetadataHandler`

Handler	Class
レコードハンドラー	SnowflakeMuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが mysnowflakecatalog の場合、環境変数の名前は mysnowflakecatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

snowflake1 (デフォルト) と snowflake2 の 2 つのデータベースインスタンスをサポートする Snowflake MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	snowflake://jdbc:snowflake://snowflake1.host:port/?warehouse=warehousename&db=db1&schema=schema1&${Test/RDS/Snowflake1}
snowflake_catalog1_connection_string	snowflake://jdbc:snowflake://snowflake1.host:port/?warehouse=warehousename&db=db1&schema=schema1${Test/RDS/Snowflake1}
snowflake_catalog2_connection_string	snowflake://jdbc:snowflake://snowflake2.host:port/?warehouse=warehousename&db=db1&schema=schema1&user=sample2&password=sample2

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

  ```
  {"username": "${username}", "password": "${password}"}
  ```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${Test/RDS/Snowflake1}` が含まれています。

```
snowflake://jdbc:snowflake://snowflake1.host:port/?
warehouse=warehousename&db=db1&schema=schema1${Test/RDS/Snowflake1}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
snowflake://jdbc:snowflake://snowflake1.host:port/
warehouse=warehousename&db=db1&schema=schema1&user=sample2&password=sample2&...
```

現在、Snowflake は user と password の JDBC プロパティを認識します。また、ユーザー名とパスワードを、user や password のキーなしで、*#####*/*#####*の形式で受け付けます。

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Snowflake インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	SnowflakeCompositeHandler
メタデータハンドラー	SnowflakeMetadataHandler
レコードハンドラー	SnowflakeRecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の Snowflake インスタンス用のプロパティ例を次に示します。

プロパティ	Value
default	snowflake://jdbc:snowflake://snowflake1.host:port/?secret=Test/RDS/Snowflake1

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス
`spill_put_request_headers`	(オプション) スピルに使用される Amazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC と Apache Arrow に対応するデータ型を示します。

JDBC	Arrow
ブール値	Bit
整数	Tiny
ショート	Smallint
整数	Int
Long	Bigint
フロート	Float4
ダブル	Float8
日付	DateDay

JDBC	Arrow
タイムスタンプ	DateMilli
文字列	Varchar
バイト	Varbinary
BigDecimal	10 進数
配列	リスト

データ型変換

JDBC から Arrow への変換に加えて、コネクタは特定の別の変換を実行して Snowflake ソースと Athena データ型との互換性を保ちます。これらの変換は、クエリを正常に実行するのに役立ちます。次の表に、これらの変換を示します。

ソースデータ型 (Snowflake)	変換されたデータ型 (Athena)
TIMESTAMP	TIMESTAMPMILLI
DATE	TIMESTAMPMILLI
INTEGER	INT
DECIMAL	BIGINT
TIMESTAMP_NTZ	TIMESTAMPMILLI

その他のサポートされていないデータ型はすべて VARCHAR に変換されます。

パーティションと分割

パーティションは、コネクタを分割する方法を決定するために使用されます。Athena は `varchar` 型の合成列を作成し、コネクタが分割を生成できるようにするために、テーブルに対するパーティションのスキームを示します。コネクタは実際のテーブル定義を変更しません。

この合成列とパーティションを作成するには、Athena がプライマリキーの定義を要求します。ただし、Snowflake はプライマリキーの制約を強制しないため、ユーザー自身が一意性を強制する必要があります。これを行わなければ、Athena がデフォルトの単一分割を実行することになります。

パフォーマンス

最適なパフォーマンスを得るには、可能な限りクエリでフィルターを使用します。さらに、パーティション分散が均一な巨大なデータセットを取得するには、ネイティブパーティションを強くお勧めします。列のサブセットを選択すると、クエリランタイムが大幅に短縮され、スキャンされるデータが減ります。Snowflake コネクタは、同時実行によるスロットリングに強いです。

Athena Snowflake コネクタは述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。スキャンされるデータ量を削減し、クエリ実行のランタイムを短縮するために、`LIMIT` 句、単純な述語、および複雑な式はコネクタにプッシュダウンされます。

LIMIT 句

`LIMIT N` ステートメントにより、クエリによってスキャンされるデータが削減されます。`LIMIT N` プッシュダウンを使用すると、コネクタは `N` 行のみを Athena に返します。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの `WHERE` 句内の式です。Athena Snowflake コネクタは、これらの式を組み合わせて Snowflake に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Snowflake コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
```

```
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

パススルークエリ

Snowflake コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Snowflake でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Snowflake 内のデータソースにクエリをプッシュダウンします。クエリは `customer` テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の Snowflake コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Microsoft SQL Server コネクタ

Microsoft SQL Server 用の Amazon Athena コネクタを使用すると、Amazon Athena で Microsoft SQL Server に保存されたデータに JDBC を使用して SQL クエリを実行できます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。
- フィルター条件では、`Date` と `Timestamp` データ型を適切なデータ型に型変換する必要があります。
- `Real` および `Float` 型の負の値を検索するには、<= または >= 演算子を使用します。
- `binary`、`varbinary`、`image`、および `rowversion` データ型はサポートされていません。

用語

SQL Server コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。

- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、connection_string プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

パラメータ

このセクションの Lambda 環境変数を使用して SQL Server コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
sqlserver://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	SqlServerMuxCompositeHandler
メタデータハンドラー	SqlServerMuxMetadataHandler
レコードハンドラー	SqlServerMuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックス

パラメータ	説明
	します。例えば、Athena に登録されたカタログが mysqlserv ercatalog の場合、環境変数の名前は mysqlserv ercatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${ *AWS_LAMBDA_FUNCTION_NAME* } の場合に使用されます。

sqlserver1 (デフォルト) と sqlserver2 の 2 つのデータベースインスタンスをサポートする SqlServer MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	sqlserver://jdbc:sqlserver://sqlserv er1. *hostname*:*port*;databaseName= *<database _name>* ;${*secret1_name* }
sqlserver_catalog1 _connection_string	sqlserver://jdbc:sqlserver://sqlserv er1. *hostname*:*port*;databaseName= *<database _name>* ;${*secret1_name* }
sqlserver_catalog2 _connection_string	sqlserver://jdbc:sqlserver://sqlserv er2. *hostname*:*port*;databaseName= *<database _name>* ;${*secret2_name* }

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets

Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${secret_name}` が含まれています。

```
sqlserver://jdbc:sqlserver://hostname:port;databaseName=<database_name>;${secret_name}
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
sqlserver://
jdbc:sqlserver://
hostname:port;databaseName=<database_name>;user=<user>;password=<password>
```

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の SQL Server インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	`SqlServerCompositeHandler`
メタデータハンドラー	`SqlServerMetadataHandler`
レコードハンドラー	`SqlServerRecordHandler`

単一接続ハンドラーのパラメータ

パラメータ	説明
`default`	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、`default` 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の SQL Server インスタンス用のプロパティ例を次に示します。

プロパティ	Value
`default`	`sqlserver://jdbc:sqlserver://` *`hostname`*`:`*`port`*`;database Name=` *`<database_name>`* `;${`*`secret_name`*` }`

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。
`spill_prefix`	必須。スピルバケットのキープレフィックス

パラメータ	説明
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、SQL Server と Apache Arrow に対応するデータ型を示します。

SQL Server	Arrow
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
整数	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
real	FLOAT4
datetime	Date(MILLISECOND)

SQL Server	Arrow
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
time	VARCHAR
datetimeoffset	Date(MILLISECOND)
char[n]	VARCHAR
varchar[n/max]	VARCHAR
nchar[n]	VARCHAR
nvarchar[n/max]	VARCHAR
text	VARCHAR
ntext	VARCHAR

パーティションと分割

パーティションは、`varchar` 型の単一パーティション列で表されます。SQL Server コネクタの場合、パーティション関数によってテーブルへのパーティションの適用方法が決まります。パーティション関数と列名の情報は、SQL Server メタデータテーブルから取得されます。次に、カスタムクエリによってパーティションが取得されます。スプリットは、受信した個別のパーティションの数に基づいて作成されます。

パフォーマンス

列のサブセットを選択すると、クエリランタイムが大幅に短縮され、スキャンされるデータが減ります。SQL Server コネクタは、同時実行によるスロットリングに強いです。

Athena SQL Server コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。単純な述語と複雑な式はコネクタにプッシュダウンされるため、スキャンされるデータ量が減少し、クエリ実行のランタイムが短縮されます。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの WHERE 句内の式です。Athena SQL Server コネクタは、これらの式を組み合わせて SQL Server に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena SQL Server コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

パススルークエリ

SQL Server コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

SQL Server でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、SQL Server 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の SQL Server コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Teradata コネクタ

Teradata 用の Amazon Athena コネクタを使用すると、Athena で Teradata データベースに保存されたデータに SQL クエリを実行できます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

制限事項

- DDL の書き込みオペレーションはサポートされていません。
- マルチプレクサの設定では、スピルバケットとプレフィックスが、すべてのデータベースインスタンスで共有されます。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

用語

Teradata コネクタに関連する用語を次に示します。

- データベースインスタンス – オンプレミス、Amazon EC2、または Amazon RDS にデプロイされたデータベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。
- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、`connection_string` プロパティに必須のプレフィックスです。
- マルチプレックスハンドラー – 複数のデータベース接続を受け入れて使用することが可能な Lambda ハンドラー。

Lambda レイヤーの前提条件

Teradata コネクタを Athena で使用するには、Teradata JDBC ドライバーを含む Lambda レイヤーを作成する必要があります。Lambda レイヤーとは、Lambda 関数の追加コードを含む `.zip` ファイルアーカイブです。Teradata コネクタをアカウントにデプロイする場合は、レイヤー の ARN を指定します。これにより、Teradata JDBC ドライバーを含む Lambda レイヤーが Teradata コネクタに接続されるため、Athena で使用できるようになります。

Lambda レイヤーの詳細については、「AWS Lambda デベロッパーガイド」の「Lambda レイヤーの作成と共有」を参照してください。

Teradata コネクタの Lambda レイヤーを作成するには

1. Teradata JDBC ドライバーダウンロードページ (https://downloads.teradata.com/download/connectivity/jdbc-driver) にアクセスしてください。

2. Teradata JDBC ドライバーをダウンロードします。ウェブサイトでは、アカウントを作成し、ファイルをダウンロードするためのライセンス契約に同意する必要があります。
3. ダウンロードしたアーカイブファイルから `terajdbc4.jar` ファイルを抽出します。
4. 次のフォルダ構造を作成し、そこに `.jar` ファイルを配置します。

   ```
   java\lib\terajdbc4.jar
   ```
5. `terajdbc4.jar` ファイルを含むフォルダ構造全体を `.zip` ファイルにします。
6. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
7. ナビゲーションペインで [Layers] (レイヤー) を選択し、[Create layer] (レイヤーの作成) を選択します。
8. [Name] (名前) に、レイヤーの名前 (例: `TeradataJava11LambdaLayer`) を入力します。
9. [Upload a .zip file] (.zip ファイルのアップロード) オプションがオンになっていることを確認します。
10. [Upload] (アップロード) を選択して、Teradata JDBC ドライバーを含む zip に圧縮されたフォルダをアップロードします。
11. [Create] (作成) を選択します。
12. レイヤーの詳細ページで、ページ上部にあるクリップボードアイコンを選択して、レイヤー ARN をコピーします。
13. ARN を参照用に保存します。

パラメータ

このセクションの Lambda 環境変数を使用して Teradata コネクタを設定します。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
teradata://${jdbc_connection_string}
```

マルチプレックスハンドラーの使用

マルチプレクサーを使用すると、単一の Lambda 関数から複数のデータベースインスタンスに接続できます。各リクエストはカタログ名によりルーティングされます。Lambda では以下のクラスを使用します。

Handler	Class
複合ハンドラー	TeradataMuxCompositeHandler
メタデータハンドラー	TeradataMuxMetadataHandler
レコードハンドラー	TeradataMuxRecordHandler

マルチプレックスハンドラーのパラメータ

パラメータ	説明
$*catalog*_connection_string	必須。データベースインスタンスの接続文字列。環境変数には、Athena で使用されているカタログの名前をプレフィックスします。例えば、Athena に登録されたカタログが myteradatacatalog の場合、環境変数の名前は myteradatacatalog_connection_string になります。
default	必須。デフォルトの接続文字列。この文字列は、カタログが lambda:${*AWS_LAMBDA_FUNCTION_NAME*} の場合に使用されます。

teradata1 (デフォルト) と teradata2 の 2 つのデータベースインスタンスをサポートする Teradata MUX Lambda 関数用のプロパティを次に示します。

プロパティ	Value
default	teradata://jdbc:teradata://teradata2.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,user=sample2&password=sample2
teradata_catalog1_connection_string	teradata://jdbc:teradata://teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,${Test/RDS/Teradata1}

プロパティ	Value
teradata_catalog2_connection_string	teradata://jdbc:teradata://teradata2.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,user=sample2&password=sample2

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

  > **Important**
  >
  > セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

  JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

  Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

  ```
  {"username": "${username}", "password": "${password}"}
  ```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 ${Test/RDS/Teradata1} が含まれています。

```
teradata://jdbc:teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,${Test/RDS/
Teradata1}&...
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
teradata://jdbc:teradata://teradata1.host/
TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,...&user=sample2&password=sample2&...
```

現在、Teradata は user と password の JDBC プロパティを認識します。また、ユーザー名とパスワードを、user や password のキーなしで、#####/#####の形式で受け付けます。

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Teradata インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	TeradataCompositeHandler
メタデータハンドラー	TeradataMetadataHandler
レコードハンドラー	TeradataRecordHandler

単一接続ハンドラーのパラメータ

パラメータ	説明
default	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、default 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

Lambda 関数でサポートされる単一の Teradata インスタンス用のプロパティ例を次に示します。

プロパティ	Value
default	teradata://jdbc:teradata://teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,secret=Test/RDS/Teradata1

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
spill_bucket	必須。スピルバケット名。
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表に、JDBC と Apache Arrow に対応するデータ型を示します。

JDBC	Arrow
ブール値	Bit
整数	Tiny
ショート	Smallint

JDBC	Arrow
整数	Int
Long	Bigint
フロート	Float4
ダブル	Float8
日付	DateDay
タイムスタンプ	DateMilli
文字列	Varchar
バイト	Varbinary
BigDecimal	10 進数
配列	リスト

パーティションと分割

パーティションは、`Integer` 型の単一パーティション列で表されます。この列には、Teradata テーブルで定義されているパーティションのパーティション名が含まれます。パーティション名のないテーブルの場合、* が返されます。これは 1 つのパーティションと同じです。パーティションはスプリットと同等です。

名前	型	説明
パーティション	整数	Teradata の名前付きパーティション。

パフォーマンス

Teradata はネイティブパーティションをサポートしています。Athena Teradata コネクタは、これらのパーティションからデータを並列に取得できます。均一なパーティション分散の非常に大きなデータセットをクエリする場合は、ネイティブパーティションを強くお勧めします。列のサブセットを選

択すると、クエリランタイムが大幅に遅くなります。同時実行が原因で、コネクタにスロットリングが発生しています。

Athena Teradata コネクタは、述語のプッシュダウンを実行して、クエリによってスキャンされるデータを減少させます。単純な述語と複雑な式はコネクタにプッシュダウンされるため、スキャンされるデータ量が減少し、クエリ実行のランタイムが短縮されます。

述語

述語は、ブール値に照らして評価し、複数の条件に基づいて行をフィルタリングする SQL クエリの `WHERE` 句内の式です。Athena Teradata コネクタは、これらの式を組み合わせて Teradata に直接プッシュすることで、機能を強化し、スキャンされるデータ量を削減できます。

次の Athena Teradata コネクタ演算子は、述語のプッシュダウンをサポートしています。

- ブーリアン: AND、OR、NOT
- 等値: EQUAL、NOT_EQUAL、LESS_THAN、LESS_THAN_OR_EQUAL、GREATER_THAN、GREATER_THA
- Arithmetic: ADD、SUBTRACT、MULTIPLY、DIVIDE、MODULUS、NEGATE
- その他: LIKE_PATTERN、IN

組み合わせたプッシュダウンの例

クエリ機能を強化するには、次の例のようにプッシュダウンタイプを組み合わせます。

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

パススルークエリ

Teradata コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Teradata でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
```

```
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Teradata 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com の Teradata コネクタ用の pom.xml ファイルを参照してください。

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Timestream コネクタ

Amazon Athena Timestream コネクタにより、Amazon Athena で Amazon Timestream とやり取りすることをが可能です。また、Amazon Athena を通して時系列データにアクセスできます。AWS Glue Data Catalog は、オプションで補足メタデータのソースとして使用できます。

Amazon Timestream は、高速でスケーラブルなフルマネージド型の専用時系列データベースで、1 日あたり何兆もの時系列データポイントの保存と分析を容易にします。Timestream は、最新のデータをメモリに保持し、履歴データをユーザー定義のポリシーに基づいてコスト最適化ストレージ階層に移動することで、時系列データのライフサイクル管理にかかる時間とコストを節約します。

アカウントで Lake Formation を有効にしている場合、AWS Serverless Application Repository でデプロイした Athena フェデレーション Lambda コネクタの IAM ロールには、Lake Formation での AWS Glue Data Catalog への読み取りアクセス権が必要です。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションの Lambda 環境変数を使用して Timestream コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。
- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。
- glue_catalog – (オプション) クロスアカウントの AWS Glue カタログを指定するために、このオプションを使用します。デフォルトでは、コネクタは自身の AWS Glue アカウントからメタデータを取得しようとします。

でのデータベースとテーブルのセットアップAWS Glue

オプションで、AWS Glue Data Catalog を補足メタデータのソースとして使用できます。Timestream での使用のために AWS Glue テーブルを有効にするには、補足メタデータを供給する Timestream データベースに一致する名前の AWS Glue データベースとテーブルが必要です。

> **Note**
>
> 最高のパフォーマンスを得るには、 データベース名とテーブル名には小文字のみを使用してください。大文字と小文字が混在すると、コネクタは大文字と小文字を区別しない検索を実行するため、計算量が多くなります。

Timestream での使用のために AWS Glue テーブルを設定するには、AWS Glue でそのテーブルのプロパティを設定する必要があります。

補足メタデータのために AWS Glue を使用するには

1. AWS Glue コンソールでテーブルを編集して、次のテーブルプロパティを追加します。
   - timestream-metadata-flag — このプロパティは、補足メタデータのテーブルをコネクタが使用できることを Timestream コネクタに示します。`timestream-metadata-flag` プロパティがテーブルプロパティのリスト内に存在するのであれば、この `timestream-metadata-flag` に任意の値を指定することが可能です。
   - _view_template — 補足メタデータに AWS Glue を使用する場合は、このテーブルプロパティを使用して、任意の Timestream SQL をビューとして指定できます。Athena Timestream コネクタは、ビューからの SQL と Athena の SQL を使用してクエリを実行します。これは、Athena では利用できない Timestream SQL の機能を使用する場合に便利です。
2. このドキュメントに記載されているとおりに、AWS Glue 用として適切なデータ型を使用しているか確認してください。

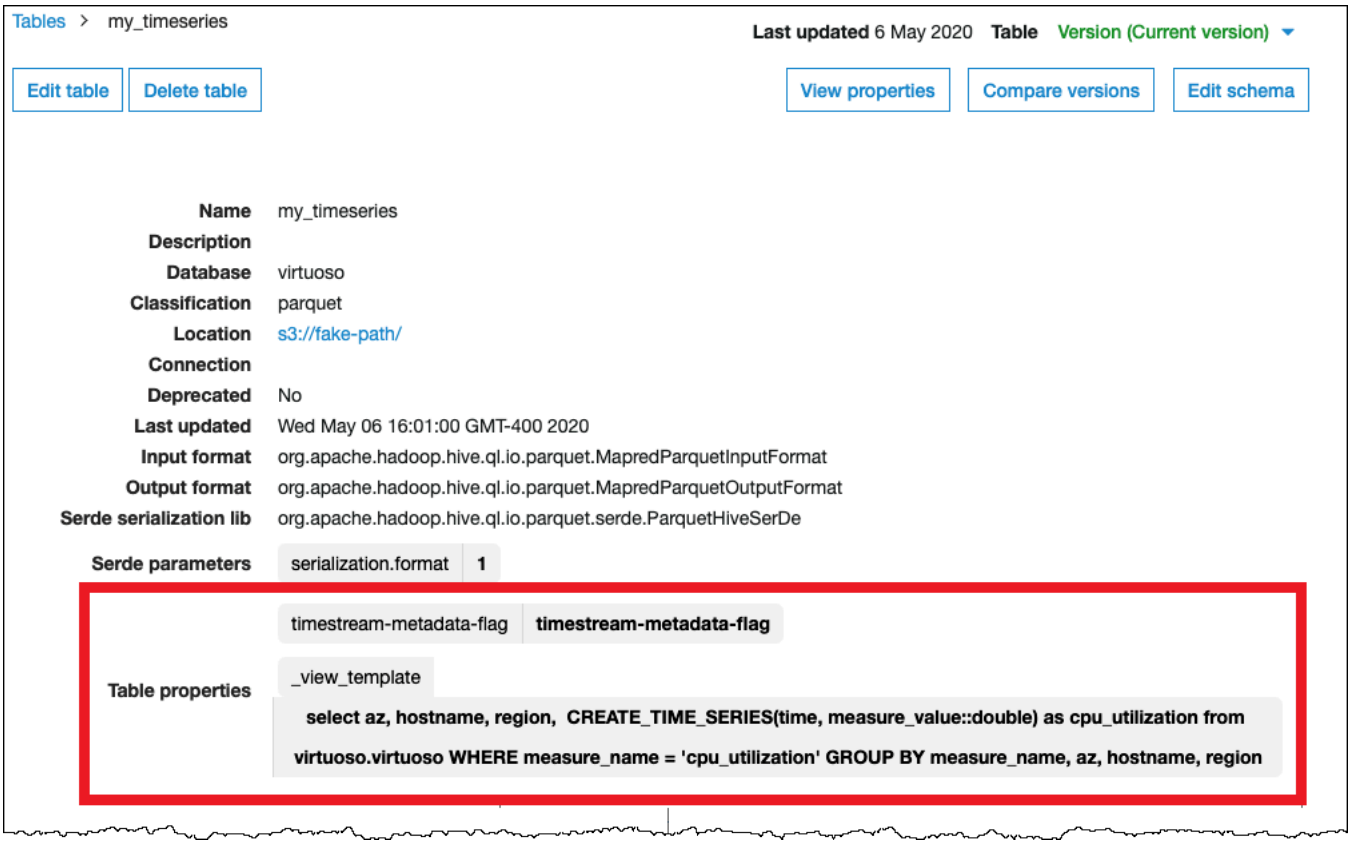
データ型

現在、Timestream コネクタは Timestream で使用可能なデータ型のサブセット (具体的にはスカラー値 `varchar`、`double`、および `timestamp`) のみをサポートしています。

`timeseries` データ型をクエリするには、Timestream `CREATE_TIME_SERIES` 関数を使用する AWS Glue テーブルプロパティでビューを設定する必要があります。また、任意の時系列の列のタイプとして構文 `ARRAY<STRUCT<time:timestamp,measure_value::double:double>>` を使用

するビューのスキーマを提供する必要があります。double をテーブルに適したスカラータイプに必ず置き換えてください。

次のイメージは、時系列でビューをセットアップするように設定された AWS Glue テーブルプロパティの例を示しています。



必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-timestream.yaml ファイルの Policies セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。
- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。
- AWS Glue Data Catalog — Timestream コネクタには、スキーマ情報を取得するために AWS Glue Data Catalog への読み込み専用アクセス権が必要です。
- CloudWatch Logs – コネクタは、ログを保存するために CloudWatch Logs にアクセスする必要があります。

- Timestream Access — Timestream クエリの実行用です。

パフォーマンス

LIMIT 句を使用して、インタラクティブなクエリのパフォーマンスを確保するために、返されるデータ (スキャンされたデータではない) を 256 MB 未満に制限することをお勧めします。

Athena Timestream コネクタは、クエリがスキャンするデータを減少させるために述語のプッシュダウンを実行します。LIMIT 句はスキャンされるデータ量を減少させますが、述語を提供しない場合、LIMIT 句を含む SELECT クエリは少なくとも 16 MB のデータをスキャンすることを想定する必要があります。列のサブセットを選択すると、クエリランタイムが大幅に短縮され、スキャンされるデータが減ります。Timestream コネクタは、同時実行によるスロットリングに強いです。

パススルークエリ

Timestream コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Timestream でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

次のクエリ例は、Timestream 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

Amazon Athena Timestream コネクタプロジェクトは、Apache-2.0 License でライセンスされています。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena TPC Benchmark DS (TPC-DS) コネクタ

Amazon Athena TPC-DS コネクタは、Amazon Athena で、Athena フェデレーションでのベンチマークや機能のテストに使用するランダムに生成された TPC Benchmark DS データのソースとやり取りすることを可能にします。Athena TPC-DS コネクタは、4 つのスケール係数のいずれかで TPC-DS 準拠のデータベースを生成します。Amazon S3 が基盤となるデータレイクのパフォーマンステストの代わりにこのコネクタを使用することはお勧めしません。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。

パラメータ

このセクションの Lambda 環境変数を使用して TPC-DS コネクタを設定します。

- spill_bucket – Lambda 関数の上限を超えたデータに対して、Amazon S3 バケットを指定します。
- spill_prefix – (オプション) 指定された `athena-federation-spill` という `spill_bucket` の、デフォルトのサブフォルダに設定します。このロケーションで、Amazon S3 のストレージライフサイクルを設定し、あらかじめ決められた日数または時間数以上経過したスピルを削除することをお勧めします。
- spill_put_request_headers – (オプション) スピリングに使用されるAmazon S3 の `putObject` リクエスト (例:`{"x-amz-server-side-encryption" : "AES256"}`) に関する、 JSON でエンコードされたリクエストヘッダーと値のマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。
- kms_key_id – (オプション) デフォルトでは、Amazon S3 に送信されるすべてのデータは、AES-GCM で認証された暗号化モードとランダムに生成されたキーを使用して暗号化されます。KMS が生成したより強力な暗号化キー (たとえば `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`) を Lambda 関数に使用させる場合は、KMS キー ID を指定します。

- disable_spill_encryption – (オプション) `True` に設定されている場合、スピルに対する暗号化を無効にします。デフォルト値は `False` です。この場合、S3 にスピルされたデータは、AES-GCM を使用して (ランダムに生成されたキー、または KMS により生成したキーにより) 暗号化されます。スピル暗号化を無効にすると、特にスピルされる先でサーバー側の暗号化を使用している場合に、パフォーマンスが向上します。

テストデータベースとテーブル

Athena TPC-DS コネクタは、4 つのスケール係数 `tpcds1`、`tpcds10`、`tpcds100`、`tpcds250` または `tpcds1000` のいずれかで TPC-DS 準拠のデータベースを生成します。

テーブルの概要

テストデータのテーブルと列の完全なリストを確認するには、`SHOW TABLES` または `DESCRIBE TABLE` クエリを実行してください。次のテーブルの概要は、便宜上提供されています。

1. call_center
2. catalog_page
3. catalog_returns
4. catalog_sales
5. カスタマー
6. customer_address
7. customer_demographics
8. date_dim
9. dbgen_version
10. household_demographics
11. income_band
12. インベントリ
13. item
14. プロモーション
15. 理由
16. ship_mode
17. ストア

18.store_returns

19.store_sales

20.time_dim

21.ウェアハウス

22.web_page

23.web_returns

24.web_sales

25.web_site

この生成されたスキーマとデータに対応する TPC-DS クエリについては、GitHub の athena-tpcds/src/main/resources/queries/ ディレクトリを参照してください。

クエリの例

次の SELECT クエリの例では、郡別の顧客層の tpcds カタログをクエリしています。

```
SELECT
  cd_gender,
  cd_marital_status,
  cd_education_status,
  count(*) cnt1,
  cd_purchase_estimate,
  count(*) cnt2,
  cd_credit_rating,
  count(*) cnt3,
  cd_dep_count,
  count(*) cnt4,
  cd_dep_employed_count,
  count(*) cnt5,
  cd_dep_college_count,
  count(*) cnt6
FROM
  "lambda:tpcds".tpcds1.customer c, "lambda:tpcds".tpcds1.customer_address ca,
 "lambda:tpcds".tpcds1.customer_demographics
WHERE
  c.c_current_addr_sk = ca.ca_address_sk AND
    ca_county IN ('Rush County', 'Toole County', 'Jefferson County',
                  'Dona Ana County', 'La Porte County') AND
    cd_demo_sk = c.c_current_cdemo_sk AND
    exists(SELECT *
```

```
            FROM "lambda:tpcds".tpcds1.store_sales, "lambda:tpcds".tpcds1.date_dim
            WHERE c.c_customer_sk = ss_customer_sk AND
              ss_sold_date_sk = d_date_sk AND
              d_year = 2002 AND
              d_moy BETWEEN 1 AND 1 + 3) AND
    (exists(SELECT *
            FROM "lambda:tpcds".tpcds1.web_sales, "lambda:tpcds".tpcds1.date_dim
            WHERE c.c_customer_sk = ws_bill_customer_sk AND
              ws_sold_date_sk = d_date_sk AND
              d_year = 2002 AND
              d_moy BETWEEN 1 AND 1 + 3) OR
      exists(SELECT *
             FROM "lambda:tpcds".tpcds1.catalog_sales, "lambda:tpcds".tpcds1.date_dim
             WHERE c.c_customer_sk = cs_ship_customer_sk AND
               cs_sold_date_sk = d_date_sk AND
               d_year = 2002 AND
               d_moy BETWEEN 1 AND 1 + 3))
GROUP BY cd_gender,
  cd_marital_status,
  cd_education_status,
  cd_purchase_estimate,
  cd_credit_rating,
  cd_dep_count,
  cd_dep_employed_count,
  cd_dep_college_count
ORDER BY cd_gender,
  cd_marital_status,
  cd_education_status,
  cd_purchase_estimate,
  cd_credit_rating,
  cd_dep_count,
  cd_dep_employed_count,
  cd_dep_college_count
LIMIT 100
```

必要な許可

このコネクタが必要とする IAM ポリシーの完全な詳細については、athena-tpcds.yaml ファイルの `Policies` セクションを参照してください。次のリストは、必要なアクセス権限をまとめたものです。

- Amazon S3 への書き込みアクセス – 大規模なクエリからの結果をスピルするために、コネクタは Amazon S3 内の口ケーションへの書き込みアクセス権限を必要とします。

- Athena GetQueryExecution – コネクタはこの権限を使用して、アップストリームの Athena クエリが終了した際に fast-fail を実行します。

パフォーマンス

Athena TPC-DS コネクタは、選択したスケールファクターに基づいてクエリの並列化を試みます。述語のプッシュダウンは Lambda 関数内で実行されます。

ライセンス情報

Amazon Athena TPC-DS コネクタプロジェクトは、Apache-2.0 License でライセンスされています。

追加リソース

このコネクタに関するその他の情報については、GitHub.com で対応するサイトを参照してください。

Amazon Athena Vertica コネクタ

Vertica は、クラウドまたはオンプレミスでデプロイできる列指向データベースプラットフォームで、エクサバイト規模のデータウェアハウスをサポートします。Athena からの Vertica データソースのクエリには、横串検索で Amazon Athena Vertica コネクタを使用できます。たとえば、Vertica 上のデータウェアハウスと Amazon S3 のデータレイクに対して分析クエリを実行できます。

前提条件

- Athena コンソールまたは AWS Serverless Application Repository を使用して AWS アカウントにコネクタをデプロイします。詳細については、「データソースコネクタのデプロイ」または「AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ」を参照してください。
- このコネクタを使用する際は、先に VPC とセキュリティグループをセットアップしておきます。詳細については、「データソースコネクタ用の VPC を作成する」を参照してください。

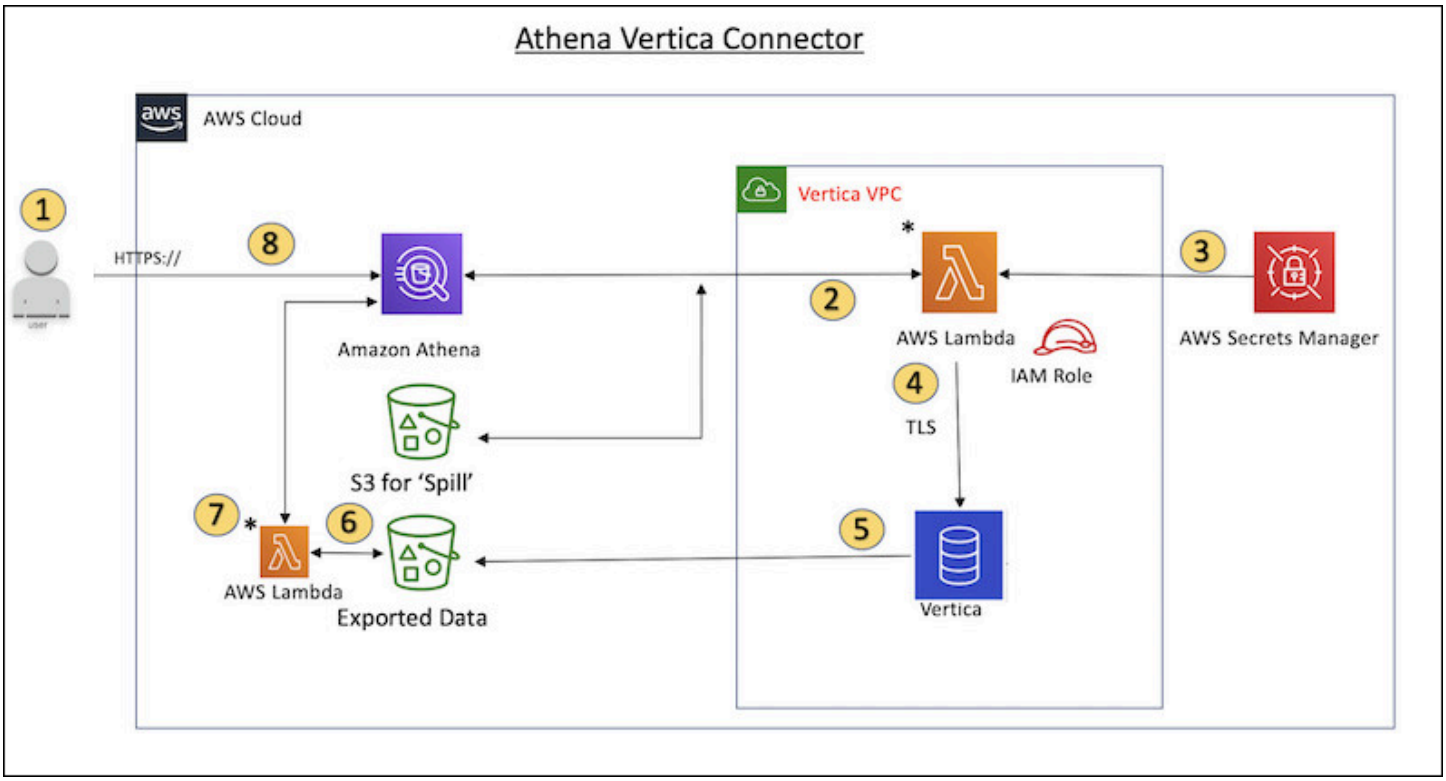
制限事項

- Athena Vertica コネクタでは、Amazon S3 からの Parquet ファイルの読み込みに Amazon S3 Select を使用するため、コネクタのパフォーマンスが低下する可能性があります。大きなテーブルに対してクエリを実行する場合は、CREATE TABLE AS (SELECT ...) クエリおよび SQL 述語を使用することをお勧めします。

- 現在、Athena 横串検索での既知の問題のため、コネクタにより Vertica ではクエリされたテーブルのすべての列が Amazon S3 にエクスポートされますが、Athena コンソールの結果にはクエリされた列のみが表示されます。
- DDL の書き込みオペレーションはサポートされていません。
- 関連性のある Lambda 上限値。詳細については、AWS Lambda デベロッパーガイドの Lambda のクォータを参照してください。

ワークフロー

次の図は、Vertica コネクタを使用するクエリのワークフローを示しています。



1. SQL クエリは、Vertica の 1 つ以上のテーブルに対して発行されます。
2. コネクタは SQL クエリを解析し、関連する部分を JDBC 接続経由で Vertica に送信します。
3. Vertica にアクセスするため、接続文字列には AWS Secrets Manager に保存されているユーザー名とパスワードを使用します。
4. コネクタは、次の例のように SQL クエリを Vertica の EXPORT コマンドでラップします。

```
EXPORT TO PARQUET (directory = 's3://DOC-EXAMPLE-BUCKET/folder_name',
   Compression='Snappy', fileSizeMB=64) OVER() as
SELECT
```

```
PATH_ID,
...
SOURCE_ITEMIZED,
SOURCE_OVERRIDE
FROM DELETED_OBJECT_SCHEMA.FORM_USAGE_DATA
WHERE PATH_ID <= 5;
```

5. Vertica が SQL クエリを処理し、結果セットを Amazon S3 バケットに送信します。スループットを向上させるため、Vertica は EXPORT オプションを使用して、複数の Parquet ファイルの書き込みオペレーションを並列化します。
6. Athena は Amazon S3 バケットをスキャンして、結果セット用に読み込むファイルの数を決定します。
7. Athena は Lambda 関数を複数回呼び出し、Amazon S3 Select を使用して結果セットから Parquet ファイルを読み込みます。複数回呼び出しを行うことで、Athena は Amazon S3 ファイルの読み込みを並列化し、1 秒あたり最大 100 GB のスループットを達成できます。
8. Athena は、Vertica から返されたデータをデータレイクからスキャンされたデータで処理し、その結果を返します。

用語

Vertica コネクタに関連する用語を次に示します。

- データベースインスタンス — Amazon EC2 にデプロイされた Vertica データベースの任意のインスタンス。
- ハンドラー – データベースインスタンスにアクセスする Lambda ハンドラー。ハンドラーには、メタデータ用とデータレコード用があります。
- メタデータハンドラー – データベースインスタンスからメタデータを取得する Lambda ハンドラー。
- レコードハンドラー – データベースインスタンスからデータレコードを取得する Lambda ハンドラー。
- 複合ハンドラー — データベースインスタンスからメタデータとデータレコードの両方を取得する Lambda ハンドラー。
- プロパティまたはパラメータ – ハンドラーがデータベース情報を抽出するために使用するデータベースプロパティ。これらのプロパティは Lambda の環境変数で設定します。
- 接続文字列 – データベースインスタンスへの接続を確立するために使用されるテキスト文字列。

- カタログ – Athena に登録された AWS Glue ではないカタログ。これは、connection_string プロパティに必須のプレフィックスです。

パラメータ

Amazon Athena Vertica コネクタは Lambda 環境変数を使用して、設定オプションを公開します。次の Lambda 環境変数を使用して、コネクタを設定できます。

- AthenaCatalogName — Lambda 関数名
- ExportBucket — Vertica のクエリ結果がエクスポートされる Amazon S3 バケット。
- SpillBucket — この関数がデータを提供できる Amazon S3 バケットの名前。
- SpillPrefix — この関数がデータを提供できる SpillBucket の場所のプレフィックス。
- SecurityGroupIds — Lambda 関数に適用する必要があるセキュリティグループに対応する 1 つ以上の ID (例: sg1、sg2、または sg3)。
- SubnetIds — Lambda 関数がデータソースへのアクセスに使用できるサブネットに対応する 1 つ以上のサブネット ID (例: subnet1、または subnet2)。
- SecretNameOrPrefix — この関数がアクセスできる Secrets Manager 内の名前セットの名前またはプレフィックス (例: vertica-*)。
- VerticaConnectionString — カタログ固有の接続が定義されていない場合にデフォルトで使用される Vertica 接続の詳細。オプションで、文字列は AWS Secrets Manager 構文を使用できます (例: ${secret_name})。
- VPC ID — Lambda 関数にアタッチする VPC ID。

接続文字列

次の形式の JDBC 接続文字列を使用して、データベースインスタンスに接続します。

```
vertica://jdbc:vertica://host_name:port/database?user=vertica-
username&password=vertica-password
```

単一接続ハンドラーの使用

次の単一接続のメタデータハンドラーとレコードハンドラーを使用して、単一の Vertica インスタンスに接続できます。

ハンドラーのタイプ	Class
複合ハンドラー	`VerticaCompositeHandler`
メタデータハンドラー	`VerticaMetadataHandler`
レコードハンドラー	`VerticaRecordHandler`

単一接続ハンドラーのパラメータ

パラメータ	説明
`default`	必須。デフォルトの接続文字列。

単一接続ハンドラーでは、1 つのデータベースインスタンスがサポートされます。また、`default` 接続文字列パラメータを指定する必要があります。他のすべての接続文字列は無視されます。

認証情報の提供

JDBC 接続文字列の中でデータベースのユーザー名とパスワードを指定するには、接続文字列のプロパティ、もしくは AWS Secrets Manager を使用します。

- 接続文字列 – ユーザー名とパスワードを、JDBC 接続文字列のプロパティとして指定できます。

> **Important**
>
> セキュリティ上のベストプラクティスとして、環境変数や接続文字列にハードコードされた認証情報を使用しないでください。ハードコードされたシークレットを AWS Secrets Manager に移動する方法については、「AWS Secrets Manager ユーザーガイド」の「ハードコードされたシークレットを AWS Secrets Manager に移動する」を参照してください。

- AWS Secrets Manager – Athena の横串検索機能を AWS Secrets Manager で使用するには、Secrets Manager に接続するためのインターネットアクセスまたは VPC エンドポイントが、Lambda 関数に接続されている VPC に必要です。

JDBC 接続文字列には、AWS Secrets Manager のシークレットの名前を含めることができます。コネクタは、このシークレット名を Secrets Manager の `username` および `password` の値に置き換えます。

Amazon RDS データベースインスタンスには、このサポートが緊密に統合されています。Amazon RDS を使用している場合は、AWS Secrets Manager と認証情報ローテーションの使用を強くお勧めします。データベースで Amazon RDS を使用していない場合は、認証情報を次の形式で JSON として保存します。

```
{"username": "${username}", "password": "${password}"}
```

シークレット名を含む接続文字列の例

次の文字列には、シークレット名 `${vertica-username}` および `${vertica-password}` が含まれています。

```
vertica://jdbc:vertica://host_name:port/database?user=${vertica-username}&password=
${vertica-password}
```

次の例のように、コネクタはシークレット名を使用し、シークレットを取得してユーザー名とパスワードを提供します。

```
vertica://jdbc:vertica://host_name:port/database?user=sample-user&password=sample-
password
```

現在、Vertica コネクタは `vertica-username` と `vertica-password` の JDBC プロパティを認識します。

スピルパラメータ

Lambda SDK は Amazon S3 にデータをスピルする可能性があります。同一の Lambda 関数によってアクセスされるすべてのデータベースインスタンスは、同じ場所にスピルします。

パラメータ	説明
`spill_bucket`	必須。スピルバケット名。

パラメータ	説明
spill_prefix	必須。スピルバケットのキープレフィックス
spill_put_request_headers	(オプション) スピルに使用される Amazon S3 の putObject リクエスト (例:{"x-amz-server-side-encryption" : "AES256"}) における、リクエストヘッダーと値に関する JSON でエンコードされたマッピング。利用可能な他のヘッダーについては、「Amazon Simple Storage Service API リファレンス」の「PutObject」を参照してください。

サポートされるデータ型

次の表は、Vertica コネクタ用にサポートされているデータ型です。

ブール値
BigInt
ショート
整数
Long
浮動小数点数
ダブル
日付
Varchar
バイト
BigDecimal
Varchar としての TimeStamp

パフォーマンス

Lambda 関数は射影プッシュダウンを実行して、クエリがスキャンするデータを削減します。LIMIT 句はスキャンされるデータ量を減らしますが、述語を提供しない場合、LIMIT 句を含む SELECT クエリは少なくとも 16 MB のデータをスキャンすることを想定する必要があります。Vertica コネクタは、同時実行によるスロットリングに強いです。

パススルークエリ

Vertica コネクタは、パススルークエリをサポートします。パススルークエリは、テーブル関数を使用して、実行のためにクエリ全体をデータソースにプッシュダウンします。

Vertica でパススルークエリを使用するには、以下の構文を使用できます。

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

以下のクエリ例は、Vertica 内のデータソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

ライセンス情報

このコネクタを使用することにより、pom.xml ファイル内のリストにある、サードパーティのコンポーネントが使用されることを承認し、 GitHub.com にある LICENSE.txt ファイルに記載されている、個別のサードパーティライセンスの使用条件に同意したとみなされます。

追加リソース

最新の JDBC ドライバーのバージョン情報については、GitHub.com で Vertica コネクタ用の pom.xml ファイルを参照してください。

このコネクタの追加情報については、GitHub.com の対応するサイトおよび AWS ビッグデータブログの「Athena Federated Query SDK を使用して Amazon Athena で Vertica データソースのクエリを実行する」を参照してください。

データソースコネクタのデプロイ

横串検索を作成するための準備は、Lambda 関数データソースコネクタのデプロイと、Lambda 関数のデータソースへの接続の 2 つの部分から成るプロセスです。このプロセスでは、Lambda 関数に後ほど Athena コンソールで選択できる名前を付けます。コネクタには SQL クエリで参照できる名前を付けます。

> **Note**
>
> AWS Secrets Manager で Athena 横串検索機能を使用するには、Secrets Manager に Amazon VPC プライベートエンドポイントを設定する必要があります。詳細については、「AWS Secrets Manager ユーザーガイド」の「Secrets Manager VPC プライベートエンドポイントを作成する」を参照してください。

トピック

Athena コンソールの使用

データソースコネクタを選択し、名前を付けてデプロイするには、統合されたプロセスで Athena コンソールと Lambda コンソールを使用します。

データソースコネクタをデプロイする

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. ナビゲーションペインで、[Data source] (データソース) をクリックします。
4. [Data sources] (データソース) タブで [Connect data source] (データソースを接続する) を選択します。
5. [Data source selection] (データソースの選択) については、次のガイドラインを考慮して、Athena でクエリを実行するデータソースを選択します。

   - データソースに対応するフェデレーションクエリのオプションを選択します。Athena には、MySQL、Amazon DocumentDB、PostgreSQL などのソースに設定できる事前構築されたデータソースコネクタがあります。
   - このページで Apache Hive メタストアまたは他のフェデレーションクエリのデータソースオプションを使用していない場合、Amazon S3 内のデータをクエリするには [S3 - AWS Glue Data Catalog] を選択します。Athena では、AWS Glue Data Catalog を使用して Amazon S3 にデータソースのメタデータとスキーマ情報を格納します。これはデフォルト (非フェデレーション) のオプションです。詳細については、「AWS Glue を使用した Amazon S3 内のデータソースへの接続」を参照してください。
   - [S3 - Apache Hive metastore] (S3 - Apache Hive メタストア) を選択して、Apache Hive メタストアを使用する Amazon S3 内のデータセットをクエリします。このオプションの詳細については、「Apache Hive メタストアへの Athena の接続」を参照してください。
   - Athena で使用する独自のデータソースコネクタを作成する場合、[Custom data source] (カスタムデータソース) を選択します。データソースコネクタの書き込みについては、「Athena Query Federation SDK を使用したデータソースコネクタの開発」を参照してください。

このチュートリアルでは、フェデレーションデータソースとして [Amazon CloudWatch Logs] を選択します。

6. [Next] を選択します。
7. [Data source details] (データソースの詳細) ページの [Data source name] (データソース名) に、Athena からデータソースをクエリする際に SQL ステートメントで使用する名前 (例えば `CloudWatchLogs`) を入力します。名前は最大 127 文字で、アカウント内で一意である必要があります。この設定は、作成後に変更することはできません。有効な文字は、a ~ z、A ~ Z、0 ~ 9、_ (アンダースコア)、@ (at マーク)、および - (ハイフン) です。`awsdatacatalog`、`hive`、`jmx`、および `system` の名前は Athena によって予約されており、データソース名には使用できません。
8. Lambda 関数を使用する場合、[Lambda 関数の作成] を選択します。選択したコネクタの機能ページが AWS Lambda コンソールで開きます。このページには、コネクタに関する詳細情報が表示されます。
9. [アプリケーションの設定] で、各アプリケーション設定の説明をよく読み、要件に対応する値を入力します。

   表示されるアプリケーション設定は、データソースのコネクタによって異なります。最低限必要な設定は次のとおりです。

   - AthenaCatalogName – ターゲットとなるデータソースを示す小文字の Lambda 関数の名前です (例えば、`cloudwatchlogs`)。
   - [SpillBucket] – Lambda 関数のレスポンスサイズ制限を超えるデータを保存するための、アカウント内の Simple Storage Service (Amazon S3) バケット。

   > **Note**
   >
   > スピルしたデータは、その後の実行では再利用されることはなく、12 時間後には安全に削除できます。Athena はこれらのデータを削除しません。これらのオブジェクトを管理するには、Simple Storage Service (Amazon S3) スピルバケットから古いデータを削除するオブジェクトライフサイクルポリシーを追加することを検討してください。詳細については、Amazon S3 ユーザーガイドの「ストレージのライフサイクルの管理」をご参照ください。

10. [I acknowledge that this app creates custom IAM roles and resource policies] (このアプリがカスタム IAM ロールとリソースポリシーを作成することを承認します) を選択します。詳細については、[Info] (情報) リンクを選択してください。
11. [デプロイ] を選択します。デプロイが完了すると、Lambda 関数が Lambda コンソールの [リソース] セクションに表示されます。

データソースへの接続

データソースコネクタをアカウントにデプロイしたら、Athena を接続できます。

アカウントにデプロイしたコネクタを使用して Athena をデータソースに接続するには

1. Athena コンソールの [Connect data sources] (データソースを接続) ページに戻ります。
2. [接続の詳細] セクションで、[Lambda 関数を選択または入力] 検索ボックスの横にある更新アイコンを選択します。
3. Lambda コンソールで作成した関数の名前を選択します。Lambda 関数の ARN が表示されます。
4. (オプション) [Tags] (タグ) でキーと値のペアを追加して、このデータソースに関連付けます。タグの詳細については、Athena リソースへのタグ付けを参照してください。
5. [Next] を選択します。
6. [Review and create] (確認と作成) ページで、データソースの詳細について確認し、[Add data source] (データソースの追加) を選択します。
7. データソースのページの [データソースの詳細] セクションには、新しいコネクタに関する情報が表示されます。これで、Athena クエリでコネクタを使用できるようになりました。

   データコネクタを使用したクエリについては、「フェデレーティッドクエリの実行」を参照してください。

AWS Serverless Application Repository を使用したデータソースコネクタのデプロイ

データソースコネクタをデプロイするには、Athena コンソールではなく AWS Serverless Application Repository を使用します。AWS Serverless Application Repository により、使用する対象のコネクタを検索し、コネクタに必要なパラメータを指定した上で、そのコネクタをアカウントにデプロイします。コネクタのデプロイが完了したら、Athena コンソールを使用して Athena がデータソースを使用できるようにします。

アカウントへのコネクタのデプロイ

AWS Serverless Application Repository を使用してデータソースコネクタをアカウントにデプロイするには

1. AWS Management Console にサインインし、サーバーレスアプリケーションリポジトリを開きます。
2. ナビゲーションペインで、[Available applications] (利用可能なアプリケーション) を選択します。
3. [Show apps that create custom roles or resource policies] (カスタム IAM ロールまたはリソースポリシーを作成するアプリを表示する) オプションを選択します。
4. 検索ボックスに、コネクタの名前を入力します。事前構築された Athena データコネクタのリストについては、「使用可能なデータソースコネクタ」を参照してください。
5. コネクタの名前を選択します。コネクタを選択すると、Lambda 関数の [Application details] (アプリケーションの詳細) ページが AWS Lambda コンソールに表示されます。
6. 詳細ページの右側の [Application settings] (アプリケーションの設定) で、必要な情報を入力します。最低限必要な設定は次のとおりです。Athena で構築されるデータコネクタのその他の設定可能なオプションについては、GitHub の「Available connectors」(使用可能なコネクタ) の対応するトピックを参照してください。

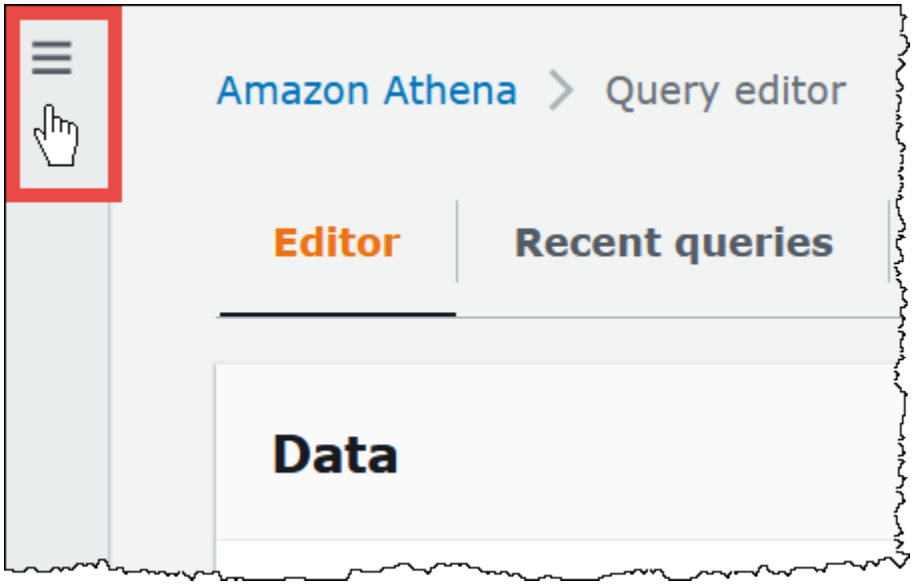
   - AthenaCatalogName — ターゲットとなるデータソースを示す小文字の Lambda 関数の名前 (例えば、`cloudwatchlogs`)。
   - SpillBucket — アカウント内の Amazon S3 バケットを指定して、Lambda 関数のレスポンスサイズ制限を超える大規模なレスポンスペイロードからデータを受信します。
7. [I acknowledge that this app creates custom IAM roles and resource policies] (このアプリがカスタム IAM ロールとリソースポリシーを作成することを承認します) を選択します。詳細については、[Info] (情報) リンクを選択してください。
8. [Application settings] (アプリケーションの設定) セクションの右下で [Deploy] (デプロイ) を選択します。デプロイが完了すると、Lambda 関数が Lambda コンソールの [リソース] セクションに表示されます。

コネクタを Athena で使用できるようにする

Athena コンソールを使用して、データソースコネクタを Athena で使用できるようにします。

データソースコネクタを Athena で使用できるようにするには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。



3. ナビゲーションペインで、[Data source] (データソース) をクリックします。
4. [Data sources] (データソース) タブで [Connect data source] (データソースを接続する) を選択します。
5. [データソースを選択] で、AWS Serverless Application Repository でコネクタを作成したデータソースを選択します。このチュートリアルでは、フェデレーションデータソースとして Amazon CloudWatch Logs を選択します。
6. [Next] を選択します。
7. [Data source details] (データソースの詳細) ページの [Data source name] (データソース名) に、Athena からデータソースをクエリする際に SQL ステートメントで使用する名前 (例えば `CloudWatchLogs`) を入力します。名前は最大 127 文字で、アカウント内で一意である必要があります。この設定は、作成後に変更することはできません。有効な文字は、a ~ z、A ~ Z、0 ~ 9、_ (アンダースコア)、@ (at マーク)、および - (ハイフン) です。`awsdatacatalog`、`hive`、`jmx`、および `system` の名前は Athena によって予約されており、データソース名には使用できません。
8. [接続の詳細] セクションで、[Lambda 関数を選択または入力] ボックスを使って、先ほど作成した関数の名前を選択します。Lambda 関数の ARN が表示されます。
9. (オプション) [Tags] (タグ) でキーと値のペアを追加して、このデータソースに関連付けます。タグの詳細については、Athena リソースへのタグ付けを参照してください。

10. [Next] を選択します。
11. [Review and create] (確認と作成) ページで、データソースの詳細について確認し、[Add data source] (データソースの追加) を選択します。
12. データソースのページの [データソースの詳細] セクションには、新しいコネクタに関する情報が表示されます。これで、Athena クエリでコネクタを使用できるようになりました。

    データコネクタを使用したクエリについては、「フェデレーティッドクエリの実行」を参照してください。

データソースコネクタ用の VPC を作成する

一部の Athena データソースコネクタには VPC とセキュリティグループが必要です。このトピックでは、サブネットと VPC 用のセキュリティグループを持つ VPC を作成する方法について説明します。このプロセスの一環として、作成した VPC、サブネット、およびセキュリティグループの ID を取得します。これらの ID は、コネクタを Athena で使用するように構成するときに必要になります。

Athena データソースコネクタ用の VPC を作成するには

1. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. ナビゲーションペインで、[New VPC Experience] (新しい VPC エクスペリエンス) が選択されていることを確認します。
3. [Create VPC (VPC の作成)] を選択します。
4. [VPC Settings] (VPC 設定) の [Resources to create] (作成するリソース) で、[VPC and more] (VPC など) を選択します。
5. [Auto-generate] (自動生成) で、VPC 内のすべてのリソースの名前タグを生成する際に使用される値を入力します。
6. [Create VPC (VPC の作成)] を選択します。
7. [View VPC] (VPC を表示) を選択します。
8. [Details] (詳細) セクションの [VPC ID] に、後で参照できるように VPC ID をコピーします。

これで、先ほど作成した VPC のサブネット ID を取得する準備が整いました。

VPC サブネット ID を取得するには

1. VPC コンソールのナビゲーションペインで、[Subnets] (サブネット) を選択します。
2. 作成したサブネットに対応する名前を選択します。
3. [Details] (詳細) セクションの [Subnet ID] (サブネット ID) に、後で参照できるようにサブネット ID をコピーします。

次に、VPC 用のセキュリティグループを作成します。

VPC 用のセキュリティグループを作成するには

1. VPC コンソールのナビゲーションペインで、[Security] (セキュリティ)、[Security Groups] (セキュリティグループ) を選択します。
2. [Create Security Group] を選択します。
3. [Create security group] (セキュリティグループの作成) ページで、以下の情報を入力します。
   - [Security group name] (セキュリティグループ名) に、セキュリティグループの名前を入力します。
   - [Description] (説明) に、セキュリティグループの説明を入力します。このフィールドは必須です。
   - [VPC] に、データソースコネクタ用に作成した VPC の VPC ID を入力します。
   - [Inbound rules] (インバウンドルール) と [Outbound rules] (アウトバウンドルール) で、必要なインバウンドルールとアウトバウンドルールを追加します。
4. [Create Security Group] を選択します。
5. セキュリティグループの [Details] (詳細) ページに、後で参照できるように [Security group ID] (セキュリティグループ ID) をコピーします。

クロスアカウントのフェデレーションクエリの有効化

フェデレーションクエリを使用すると、AWS Lambda にデプロイされたデータソースコネクタを使用して Amazon S3 以外のデータソースをクエリできます。クロスアカウントのフェデレーションクエリ機能を使用すると、Lambda 関数とクエリ対象のデータソースを異なるアカウントに配置できます。

データ管理者は、データコネクタをデータアナリストのアカウントと共有することでクロスアカウントの串刺検索を有効化できます。また、データアナリストは、データ管理者から共有された Lambda

ARN を使用してアカウントに追加することでクロスアカウントの串刺検索を有効化できます。元のアカウントのコネクタの設定を変更すると、更新された設定は自動的に他のユーザーのアカウントのコネクタの共有インスタンスに適用されます。

考慮事項と制約事項

- クロスアカウントのフェデレーションクエリ機能は、Lambda ベースのデータソースを使用する Hive 以外のメタストアデータのコネクタで使用できます。
- AWS Glue Data Catalog データソース型では、この機能は使用できません。AWS Glue Data Catalog へのクロスアカウントアクセスの詳細については、「AWS Glue データカタログへのクロスアカウントアクセス」を参照してください。
- コネクタの Lambda 関数からの応答が Lambda 応答サイズの制限である 6 MB を超えると、Athena は自動的に応答を暗号化してバッチ処理し、設定した Amazon S3 バケットに流出します。Athena が流出したデータを読み取るには、Athena クエリを実行するエンティティが流出場所にアクセスできる必要があります。クエリが完了するとデータが不要になるため、流出場所からオブジェクトを削除するように Amazon S3 ライフサイクルポリシーを設定することをお勧めします。
- AWS リージョン をまたいだフェデレーションクエリの使用はサポートされていません。

必要なアクセス許可

- データ管理者のアカウント A がデータアナリストのアカウント B と Lambda 関数を共有するには、アカウント B に Lambda の呼び出し関数と流出バケットのアクセスが必要です。したがって、アカウント A では、Lambda 関数へのリソースベースのポリシーと Amazon S3 内の流出バケットへのプリンシパルアクセスを追加する必要があります。
  1. 次のポリシーは、アカウント A の Lambda 関数で Lambda の呼び出し関数のアクセス許可をアカウント B に付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountInvocationStatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": ["arn:aws:iam::account-B-id:user/username"]
      },
      "Action": "lambda:InvokeFunction",
```

```
      "Resource": "arn:aws:lambda:aws-region:account-A-id:function:lambda-function-
name"
    }
  ]
}
```

2. 次のポリシーは、アカウント B のプリンシパルに流出バケットへのアクセスを許可します。

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": ["arn:aws:iam::account-B-id:user/username"]
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
             ],
            "Resource": [
                "arn:aws:s3:::spill-bucket",
                "arn:aws:s3:::spill-bucket/*"
            ]
        }
     ]
 }
```

3. Lambda 関数がフェデレーション SDK によって提供されるデフォルトの暗号化ではなく、AWS KMS キーを使用して流出バケットを暗号化している場合、次の例のようにアカウント A の AWS KMS キーポリシーで、アカウント B のユーザーにアクセス権を付与する必要があります。

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal":
    {
       "AWS": ["arn:aws:iam::account-B-id:user/username"]
    },
    "Action": [ "kms:Decrypt" ],
    "Resource": "*" // Resource policy that gets placed on the KMS key.
```

```
}
```

- アカウント A のコネクタをアカウント B と共有するには、AWS Security Token Service の AssumeRole API アクションを呼び出すことでアカウント A が引き受ける `AthenaCrossAccountCreate-account-A-id` というロールをアカウント B で作成する必要があります。

  `CreateDataCatalog` アクションを許可する次のポリシーは、アカウント B で作成され、アカウント B がアカウント A に対して作成する `AthenaCrossAccountCreate-account-A-id` ロールに追加される必要があります。

```
{
 "Effect": "Allow",
 "Action": "athena:CreateDataCatalog",
 "Resource": "arn:aws:athena:*:account-B-id:datacatalog/*"
}
```

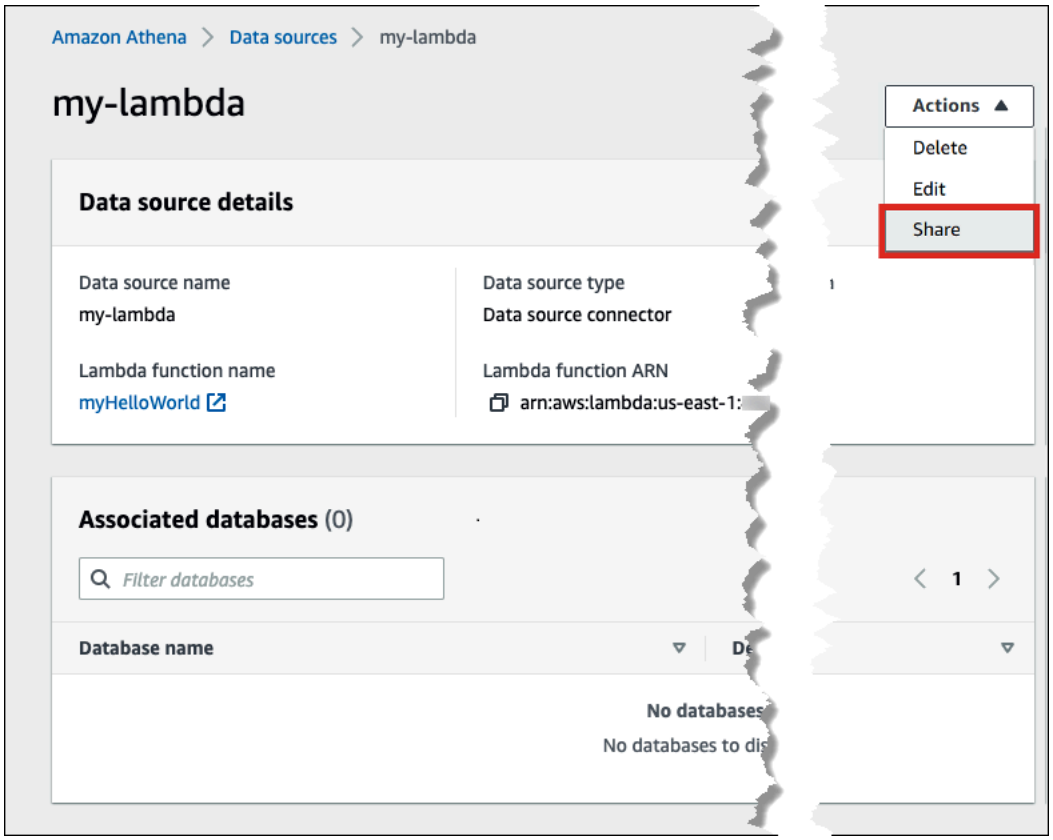
アカウント A のデータソースをアカウント B と共有する

アクセス許可を設定したら、Athena コンソールの [Data sources] (データソース) ページで、アカウント (アカウント A) のデータコネクタを別のアカウント (アカウント B) と共有できます。アカウント A は、コネクタの完全な制御と所有権を保持しています。アカウント A がコネクタの設定を変更すると、更新された設定がアカウント B の共有されたコネクタに適用されます。

アカウント A の Lambda データソースをアカウント B と共有するには

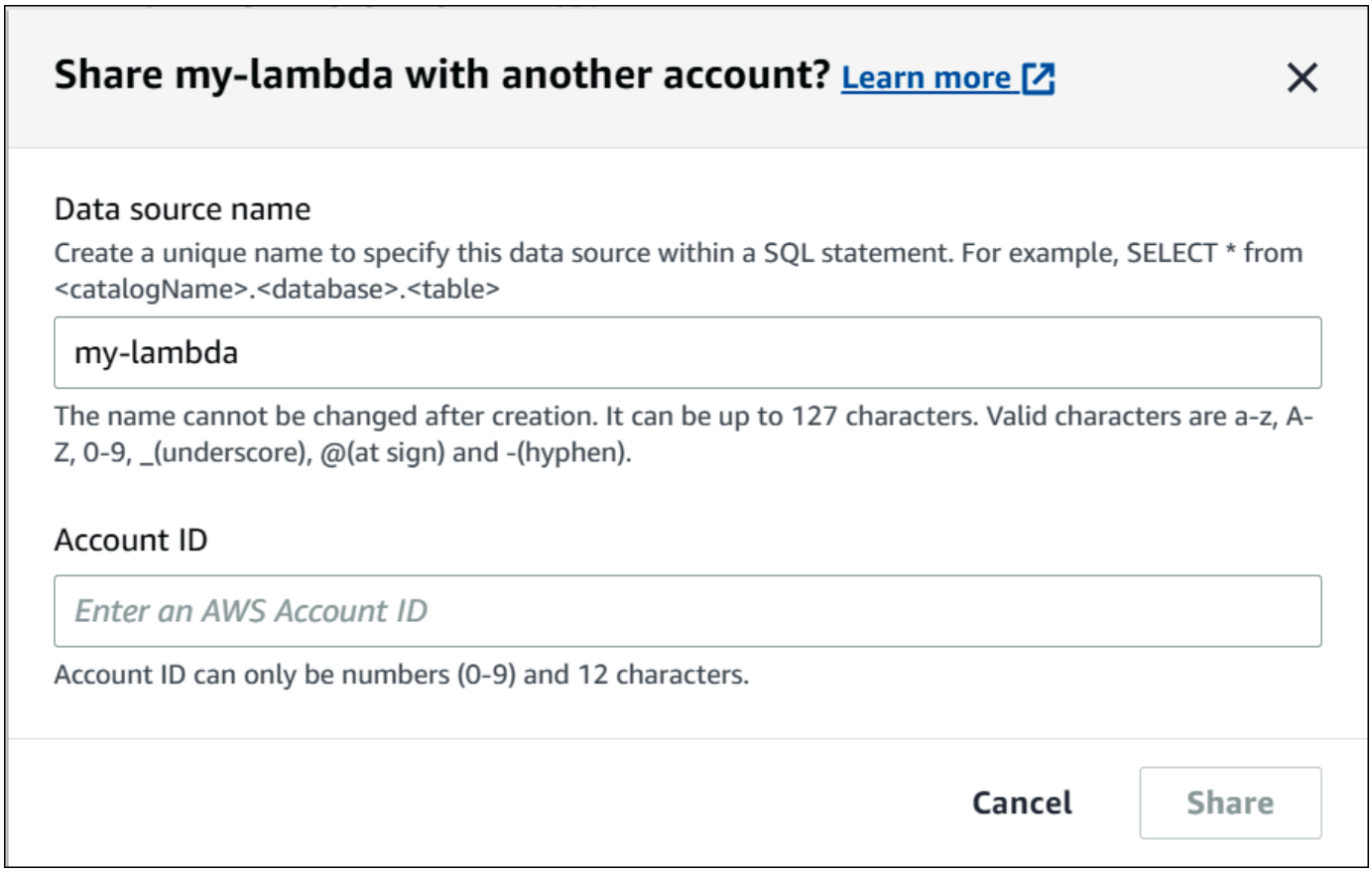
1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. [Data sources] (データソース) を選択します。
4. [Data sources] (データソース) ページで、共有するコネクタのリンクを選択します。
5. Lambda データソースの詳細ページで、右上隅の [Share] (共有) オプションを選択します。



6. [Share ***Lambda-name*** with another account] (別のアカウントと Lambda-name を共有) ダイアログボックスで、必要な情報を入力します。

- [Data source name] (データソース名) については、コピーしたデータソースの名前を他のアカウントで表示したいとおりに入力します。
- [Account ID] (アカウント ID) については、データソースを共有するアカウントの ID を入力します (この場合、アカウント B)。



7. [共有] を選択します。指定した共有のデータコネクタが、アカウント B で作成されます。アカウント A のコネクタに対する設定の変更は、アカウント B のコネクタに適用されます。

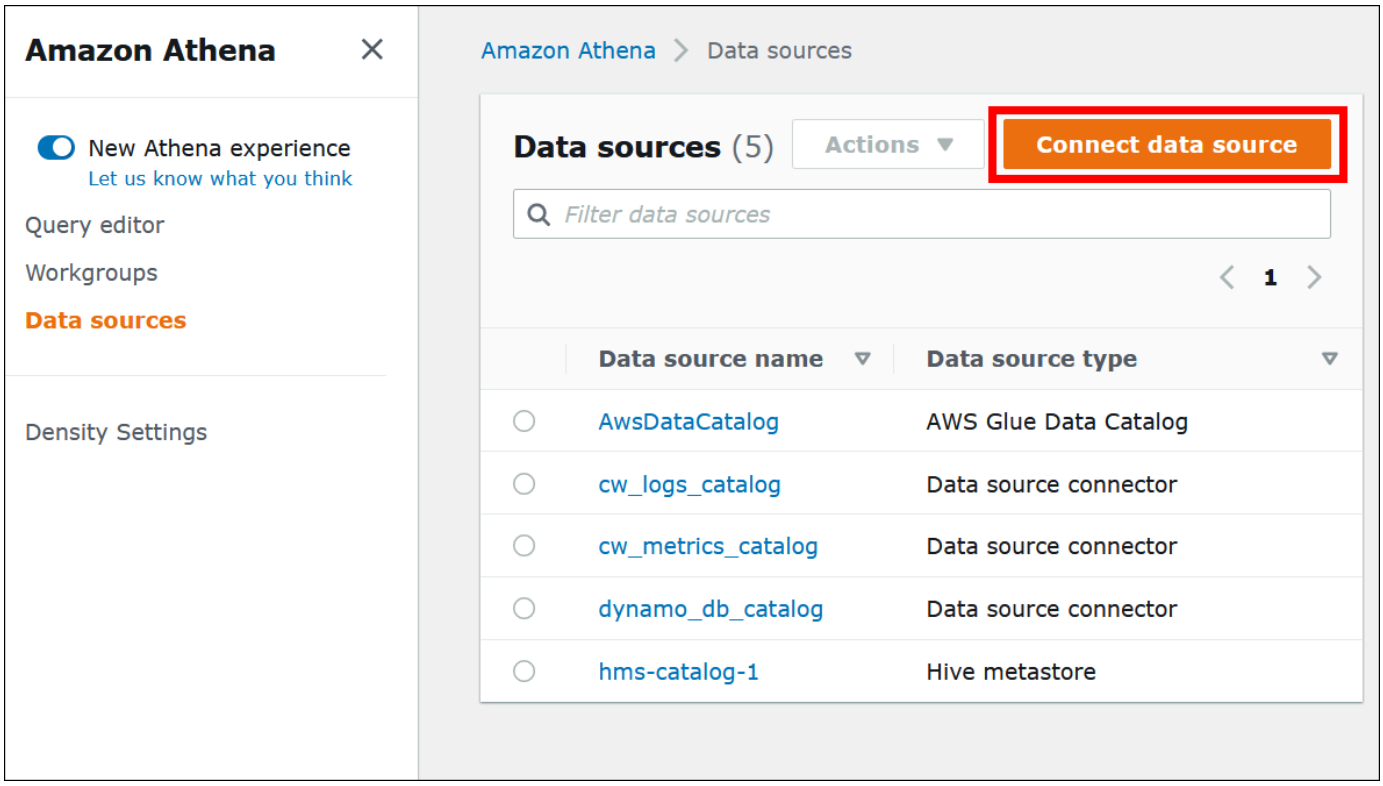
アカウント A からアカウント B への共有データソースの追加

データアナリストとして、データ管理者からアカウントに追加するコネクタの ARN が与えられる場合があります。Athena コンソールの [Data sources] (データソース) ページで、管理者から提供された Lambda ARN をアカウントに追加します。

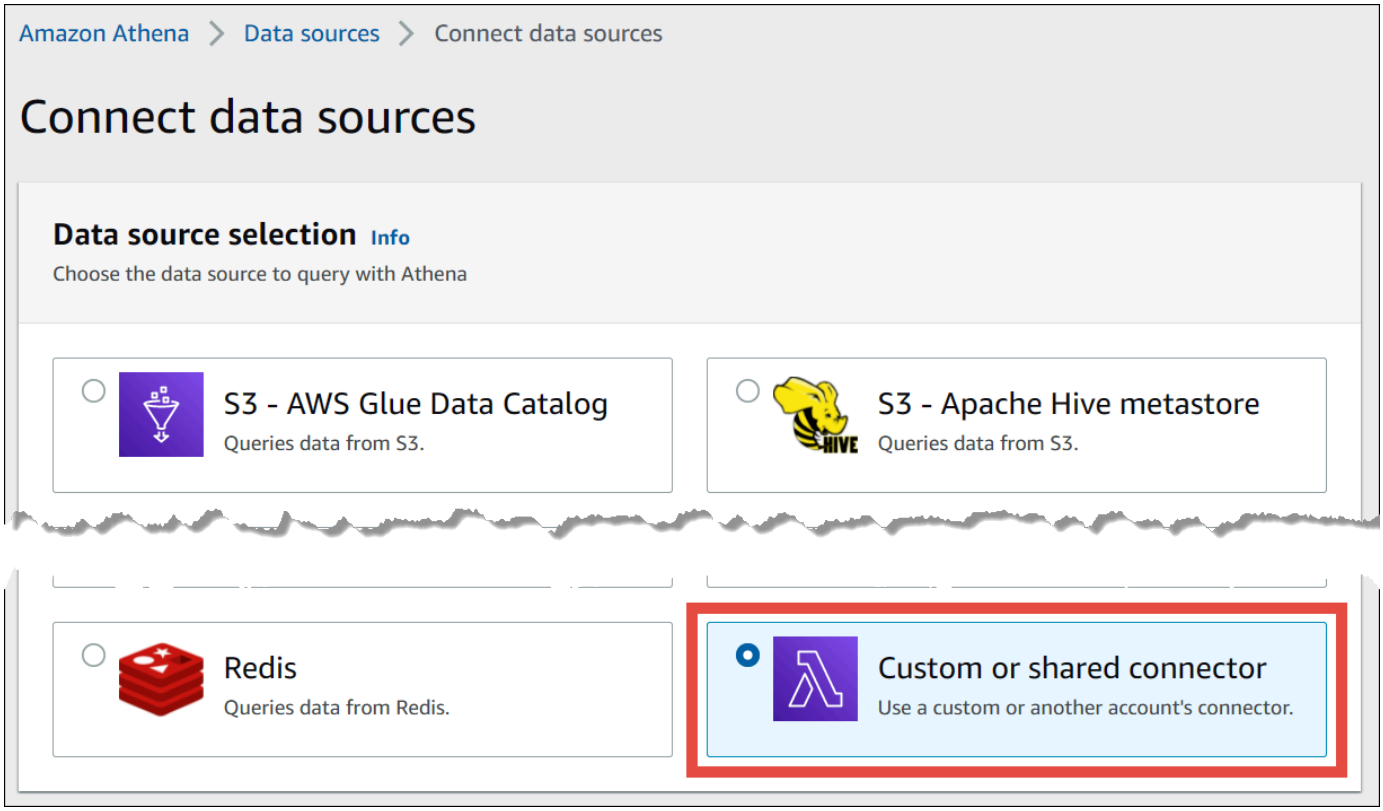
共有されたデータコネクタの Lambda ARN をアカウントに追加するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。

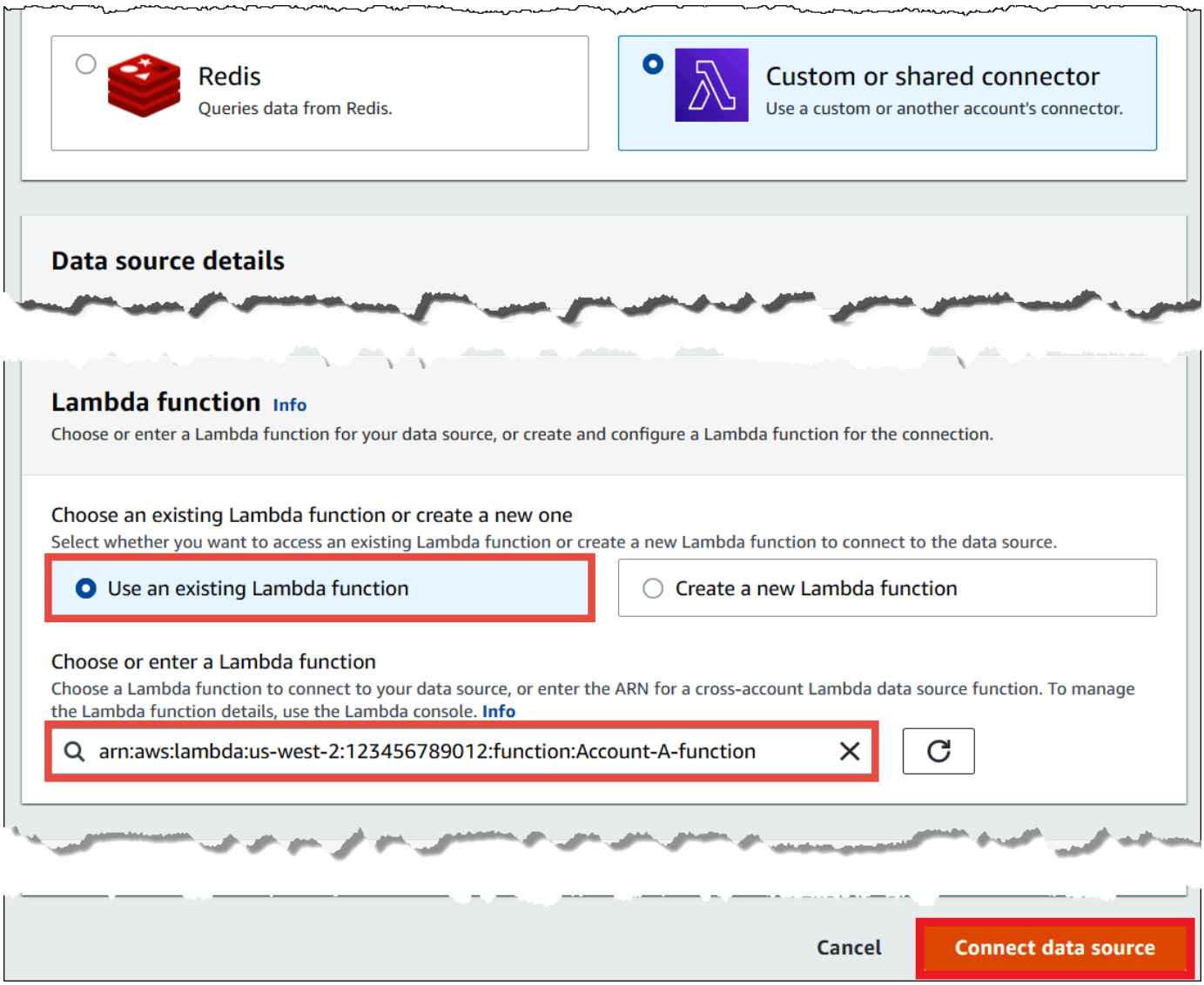
2. 新しいコンソールエクスペリエンスを使用しており、ナビゲーションペインが表示されない場合は、左側の展開メニューを選択します。
3. [Data sources] (データソース) を選択します。
4. [Data sources] (データソース) タブで [Connect data source] (データソースを接続する) をクリックします。



5. [Custom or shared connector] (カスタムまたは共有されたコネクタ) 選択します。



6. [Lambda function] (Lambda 関数) セクションで、[Use an existing Lambda function] (既存の Lambda 関数の使用) オプションが選択されていることを確認します。



7. [Choose or enter a Lambda function] (Lambda 関数の選択または入力) については、アカウント A の Lambda ARN を入力します。
8. [Connect data source] (データソースを接続する) をクリックします。

トラブルシューティング

アカウント A にアカウント B のロールを引き受けるアクセス許可がないというエラーメッセージが表示された場合は、アカウント B で作成されたロールの名前のスペルが正しいこと、適切なポリシーがアタッチされていることを確認してください。

データソースコネクタの更新

最新バージョンを使用して新機能と機能強化を利用するために、Athena ではデータソースコネクタを定期的に更新することを推奨しています。まず、最新バージョン番号を調べる必要があります。

最新バージョンの Athena Query Federation を確認する

Athena データソースコネクタの最新バージョン番号は、最新の Athena Query Federation バージョンに対応しています。特定のケースでは、GitHub のリリースが AWS Serverless Application Repository (SAR) で入手可能なリリースよりも少し新しい可能性があります。

最新の Athena Query Federation バージョン番号を確認するには

1. GitHub URL (https://github.com/awslabs/aws-athena-query-federation/releases/latest) を確認してください。
2. メインページの見出し次の形式のようになっており、そこからリリース番号を書き留めます。

   Release v *#.## X ##.####* of Athena Query Federation

   たとえば、Release v2023.8.3 of Athena Query Federation のリリース番号は 2023.8.3 です。

リソース名の検索とメモ

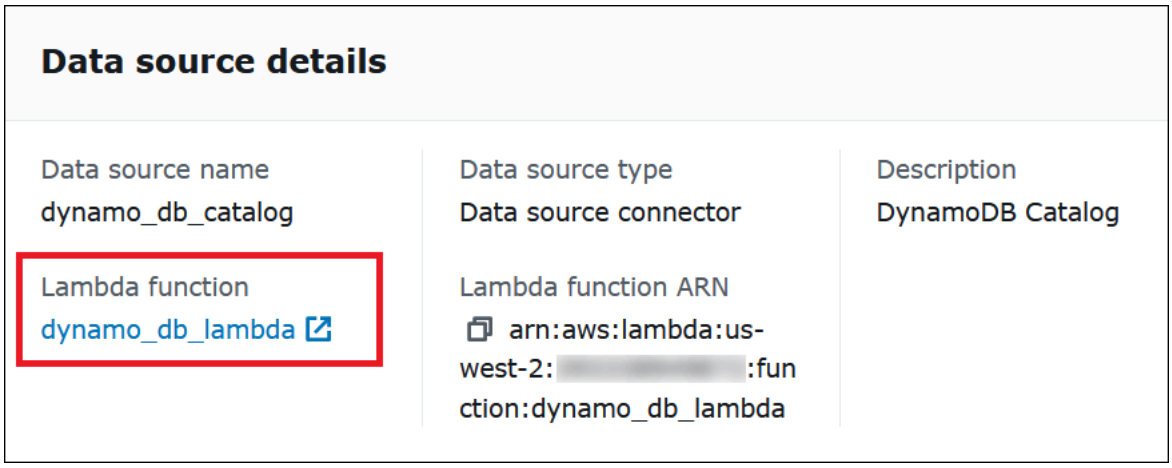
アップグレードの準備として、次の情報を確認して書き留めておく必要があります:

1. コネクタの Lambda 関数名。
2. Lambda 関数の環境変数。
3. コネクタの Lambda 関数を管理する Lambda アプリケーション名。

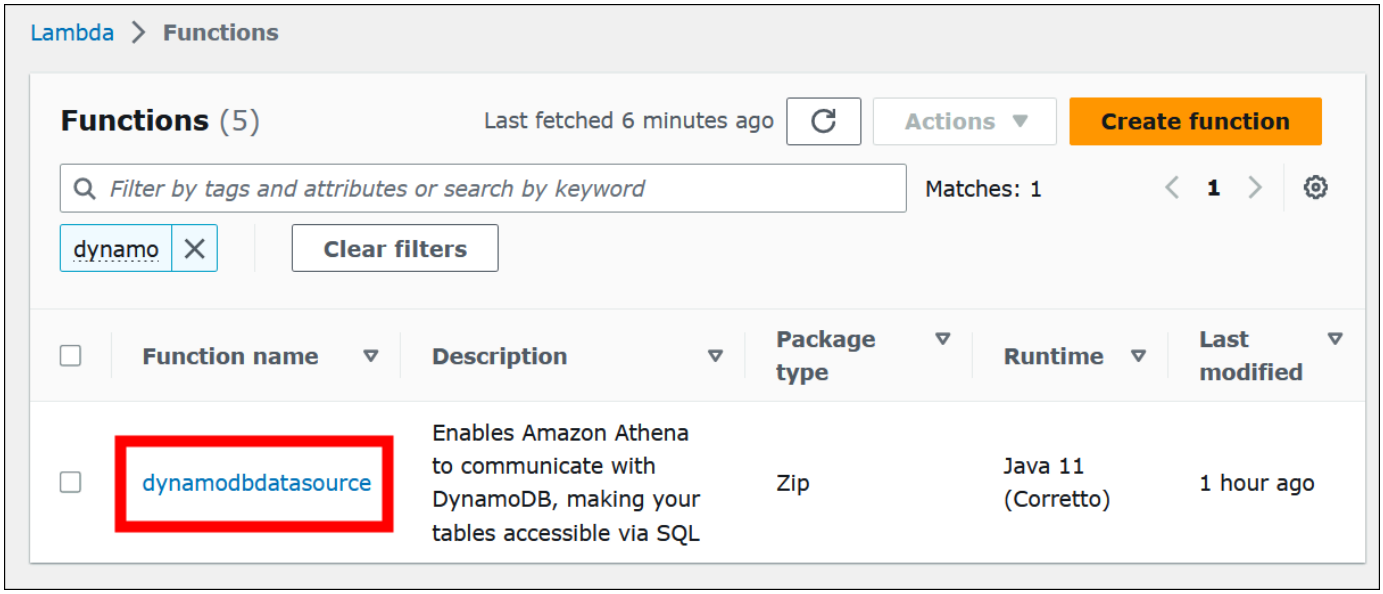
Athena コンソールからリソース名を検索するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. ナビゲーションペインで、[Data source] (データソース) をクリックします。
4. [データソース名] 列で、コネクタのデータソースへのリンクを選択します。
5. [データソースの詳細] セクションの [Lambda 関数] で、Lambda 関数へのリンクを選択します。



6. [関数] ページの [関数名] 列で、コネクタの関数名を見つけて書き留めます。



7. 関数名のリンクを選択します。
8. [関数の概要] セクションで、[構成] タブを選択します。
9. 左側のペインで、[環境変数] をクリックします。
10. 「環境変数」セクションで、キーとそれに対応する値を見つけて書き留めます。
11. ページの上部までスクロールします。
12. メッセージ [この関数はアプリケーションに属しています。] [管理するにはここをクリックしてください] が表示されたら、[ここをクリック] リンクを選択してください。
13. 「serverlessrepo-*your_application_name*」ページで、serverlessrepo を除いたアプリケーション名を見つけて書き留めてください。たとえば、アプリケーション名が serverlessrepo-DynamoDbTestApp の場合、アプリケーション名は DynamoDbTestApp です。
14. アプリケーションの Lambda コンソールページを開いたままにして、[使用しているコネクタのバージョンを確認] のステップに進んでください。

使用しているコネクタのバージョンを確認

使用しているコネクタのバージョンを確認するには、次のステップに従います。

使用しているコネクタのバージョンを確認するには

1. Lambda アプリケーションの Lambda コンソールページで、[デプロイ] タブを選択します。
2. [デプロイ] タブで、[SAM テンプレート] を展開します。
3. CodeUri を検索します。

4. CodeUri 下にある[キー] フィールドで、次の文字列を探します:

```
applications-connector_name-
versions-year.week_of_year.iteration_of_week/hash_number
```

次の例は、CloudWatch コネクタの文字列を示しています:

```
applications-AthenaCloudwatchConnector-versions-2021.42.1/15151159...
```

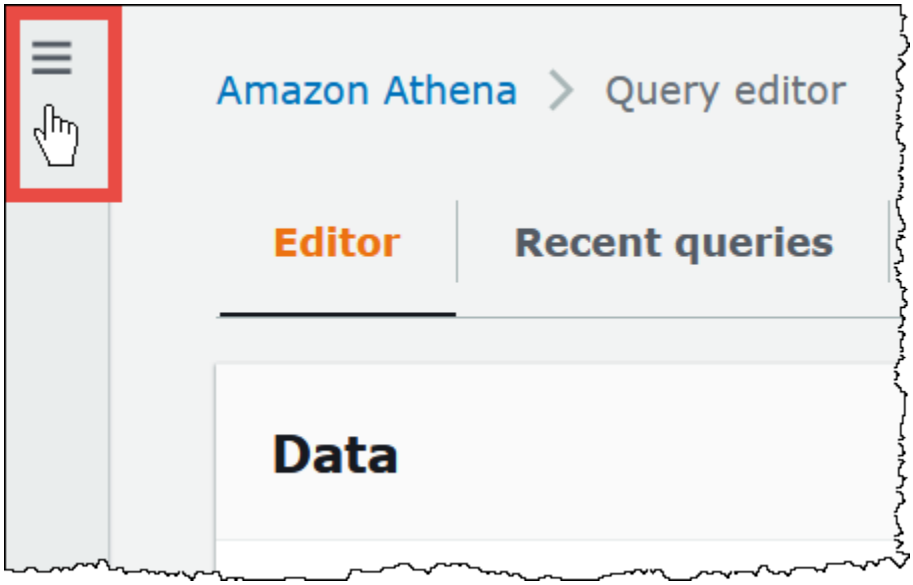
5. *#.#### X #.##########*の値を記録します (たとえば、2021.42.1)。これはお使いのコネクタのバージョンです。

新しいバージョンのコネクタをデプロイする

新しいバージョンのコネクタをデプロイするには、次のステップに従います。

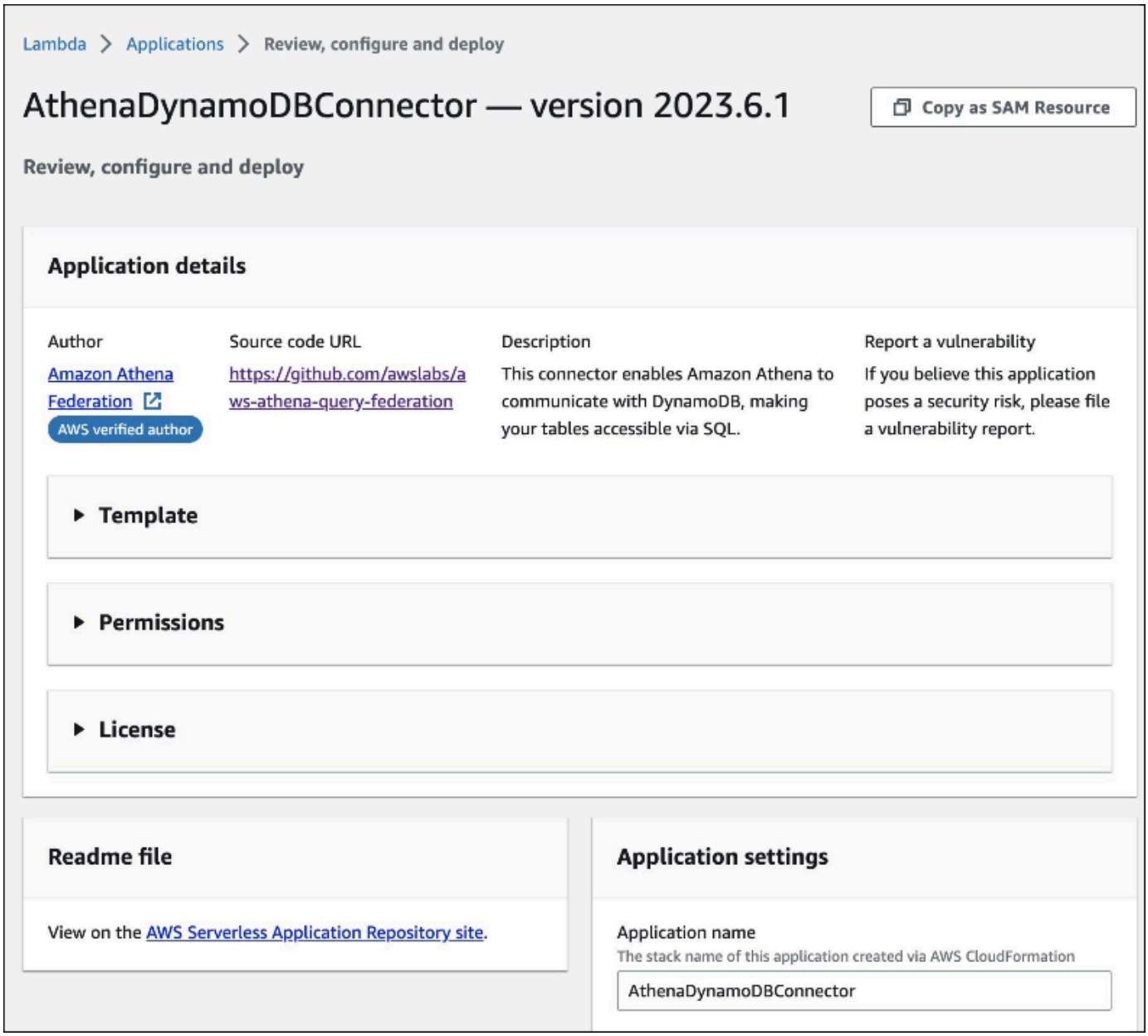
新しいバージョンのコネクタをデプロイするには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。



3. ナビゲーションペインで、[Data source] (データソース) をクリックします。
4. [Data sources] (データソース) タブで [Connect data source] (データソースを接続する) を選択します。
5. アップグレードするデータソースを選択し、[次へ] を選択します。

6. [接続の詳細] セクションで、[Lambda 関数の作成] を選択します。これにより、更新したアプリケーションをデプロイできる Lambda コンソールが開きます。



7. 実際には新しいデータソースを作成していないため、[Athena コンソール] タブを閉じてかまいません。

8. コネクタの Lambda コンソールページで、次のステップを実行します:

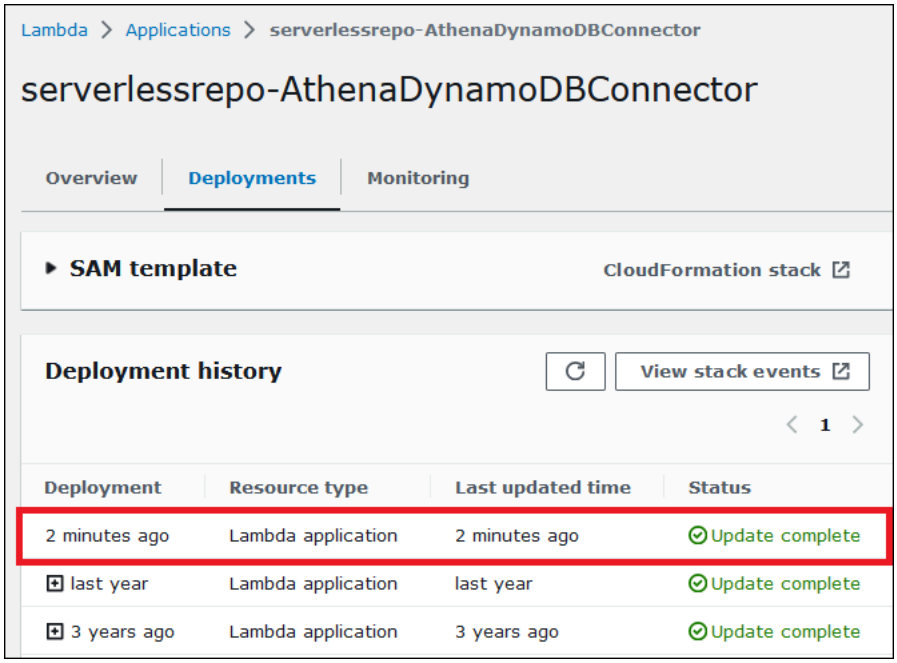
   a. アプリケーション名から serverlessrepo- プレフィックスを削除したことを確認し、アプリケーション名を [アプリケーション名] フィールドにコピーします。

   b. Lambda 関数名を [Athena カタログ名] フィールドにコピーします。一部のコネクタでは、このフィールドを [Lambda 関数名] と呼んでいます。

c. 記録した環境変数を対応するフィールドにコピーします。

9. [このアプリがカスタム IAM ロールとリソースポリシーを作成することを承認します] を選択した後、[デプロイ] を選択します。
10. アプリケーションが更新されたことを確認するには、[デプロイ] タブを選択します。

    [デプロイ履歴] セクションには、更新が完了したことが表示されます。



11. 新しいバージョン番号を確認するには、以前と同様に [SAM テンプレート] を展開して CodeUri を検索し、コネクタのバージョン番号を [キー] フィールドで確認します。

これで、更新したコネクタを使用して Athena フェデレーティッドクエリを作成できるようになりました。

フェデレーティッドクエリの実行

1 つ、または複数のデータコネクタを設定してアカウントにデプロイしたら、それらを Athena クエリで使用できます。

単一データソースのクエリ

このセクションの例では、Amazon Athena CloudWatch コネクタ が構成され、アカウントにデプロイされていることを前提としています。他のコネクタを使用する場合も、同じ方法でクエリを実行します。

CloudWatch コネクタを使用する Athena クエリを作成する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. Athena クエリエディタで、FROM 句に次の構文を使用する SQL クエリを作成します。

```
MyCloudwatchCatalog.database_name.table_name
```

例

以下の例では、/var/ecommerce-engine/order-processor CloudWatch Logs ロググループの all_log_streams ビューへの接続に Athena CloudWatch コネクタを使用します。all_log_streams ビューは、ロググループ内のすべてのログストリームのビューです。このクエリ例では、返される行数を 100 に制限します。

```
SELECT *
FROM "MyCloudwatchCatalog"."/var/ecommerce-engine/order-processor".all_log_streams
LIMIT 100;
```

次の例では、前の例と同じビューの情報を解析します。この例では、注文 ID とログレベルを抽出し、レベル INFO のメッセージをフィルタリングします。

```
SELECT
    log_stream as ec2_instance,
    Regexp_extract(message '.*orderId=(\d+) .*', 1) AS orderId,
    message AS order_processor_log,
    Regexp_extract(message, '(.*):.*', 1) AS log_level
FROM MyCloudwatchCatalog."/var/ecommerce-engine/order-processor".all_log_streams
WHERE Regexp_extract(message, '(.*):.*', 1) != 'INFO'
```

複数のデータソースのクエリ

さらに複雑な例として、次のデータソースを使用して顧客の購入に関するデータを保存する電子商取引会社があると考えてみましょう。

- 製品カタログデータを保存ための Amazon RDS for MySQL
- E メールアドレスや配送先住所などの顧客アカウントデータを保存するための Amazon DocumentDB
- 注文の発送と追跡データを保存するための Amazon DynamoDB

この電子商取引アプリケーションのデータアナリストが、一部の地域で地域の気象条件により配送時間に影響が出ていることを知ったとしましょう。アナリストは、遅延している注文の数や、影響を受けている顧客のいる場所、最も影響を受けている製品を知りたいと考えます。情報のソースを個別に調査することはせずに、アナリストは Athena を使用して、データを 1 つのフェデレーションクエリに結合します。

Example

```
SELECT
     t2.product_name AS product,
     t2.product_category AS category,
     t3.customer_region AS region,
     count(t1.order_id) AS impacted_orders
FROM my_dynamodb.default.orders t1
JOIN my_mysql.products.catalog t2 ON t1.product_id = t2.product_id
JOIN my_documentdb.default.customers t3 ON t1.customer_id = t3.customer_id
WHERE
     t1.order_status = 'PENDING'
     AND t1.order_date between '2022-01-01' AND '2022-01-05'
GROUP BY 1, 2, 3
ORDER BY 4 DESC
```

フェデレーションされたビューのクエリ

フェデレーションされたソースをクエリする場合、ビューを使用して基になるデータソースを難読化したり、データにクエリする他のアナリストに複雑な結合が表示されないようにすることができます。

考慮事項と制約事項

- フェデレーションされたビューには Athena バージョン 3 が必要です。
- フェデレーションされたビューは、基になるデータソースではなく、AWS Glue に格納されます。
- フェデレーションされたカタログで作成されたビューでは、次の例にあるように、完全に修飾されている名前構文を使用する必要があります。

  ```
  "ddbcatalog"."default"."customers"
  ```

- フェデレーションされたソースでクエリを実行するユーザーには、フェデレーションされたソースをクエリする権限が必要です。

- フェデレーションされたビューには athena:GetDataCatalog 権限が必要です。詳細については、「Athena Federated Query を許可する IAM 許可ポリシーの例」を参照してください。

例

次の例では、フェデレーションされたデータソースに保存されているデータに基づいて、customers と呼ばれるビューを作成しています。

Example

```
CREATE VIEW customers AS
SELECT *
FROM my_federated_source.default.table
```

次のクエリ例は、基になるフェデレーションされたデータソースではなく、customers ビューを参照するクエリを示しています。

Example

```
SELECT id, SUM(order_amount)
FROM customers
GROUP by 1
ORDER by 2 DESC
LIMIT 50
```

次の例では、フェデレーションされたデータソースからのデータと Amazon S3 データソースからのデータを組み合わせた、order_summary と呼ばれるビューを作成しています。Athena ですでに作成済みのフェデレーションされたソースからは、ビューは person および profile テーブルを使用します。Amazon S3 からのデータでは、ビューは purchase および payment テーブルを使用します。Amazon S3 を参照する場合、ステートメントではキーワード awsdatacatalog を使用します。フェデレーションされたデータソースは完全に修飾された名前構文である *federated_source_name*.*federated_source_database*.*federated_source_table* を使用していることに注目してください。

Example

```
CREATE VIEW default.order_summary AS
SELECT *
FROM federated_source_name.federated_source_database."person" p
```

```
    JOIN federated_source_name.federated_source_database."profile" pr ON pr.id = p.id
    JOIN awsdatacatalog.default.purchase i ON p.id = i.id
    JOIN awsdatacatalog.default.payment pay ON pay.id = p.id
```

追加リソース

- 元のソースから分離され、マルチユーザーモデルでのオンデマンドの分析に使用できるフェデレーティッドビューの例については、AWS Big Data Blog の「Extend your data mesh with Amazon Athena and federated views」を参照してください。
- Athena のビューを使用するための詳細については、「ビューの使用」を参照してください。

フェデレーテッドパススルークエリの実行

Athena では、データソース自体のクエリ言語を使用してフェデレーテッドデータソースに対するクエリを実行したり、実行のためにクエリ全体をデータソースにプッシュダウンしたりすることができます。これらのクエリはパススルークエリと呼ばれます。パススルークエリを実行するには、Athena クエリでテーブル関数を使用します。データソースで実行するパススルークエリは、テーブル関数に対する引数の 1 つに含めます。パススルークエリは、Athena SQL を使用して分析できるテーブルを返します。

サポートされるコネクタ

以下の Athena データソースコネクタがパススルークエリをサポートしています。

- Azure Data Lake Storage
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- CloudWatch
- Db2
- Db2 iSeries
- DocumentDB
- DynamoDB
- HBase
- Google BigQuery

- Hortonworks
- MySQL
- Neptune
- OpenSearch
- Oracle
- PostgreSQL
- Redshift
- SAP HANA
- Snowflake
- SQL Server
- Teradata
- Timestream
- Vertica

考慮事項と制約事項

Athena でパススルークエリを使用するときは、以下の点を考慮してください。

- クエリパススルーがサポートされるのは、Athena SELECT ステートメントまたは読み取り操作のみです。
- パススルークエリは、外部クエリ (つまり、テーブル関数を呼び出すクエリ) のカタログのコンテキスト内で実行される必要があります。
- クエリのパフォーマンスは、データソースの設定に応じて異なる場合があります。
- ビューでは、パススルークエリがサポートされません。

構文

以下は、一般的な Athena クエリパススルー構文です。

```
SELECT * FROM TABLE(system.function_name(arg1 => 'arg1Value'[, arg2 =>
 'arg2Value', ...]))
```

ほとんどのデータソースでは、query が最初で唯一の引数であり、その後にアロー演算子 => とクエリ文字列が続きます。

```
SELECT * FROM TABLE(system.query(query => 'query string'))
```

オプションの名前付き引数 query とアロー演算子 => は、簡素化のために省略することが可能です。

```
SELECT * FROM TABLE(system.query('query string'))
```

データソースがクエリ文字列以上のものを必要とする場合は、データソースが期待する順序で名前付き引数を使用してください。たとえば、*arg1* => '*arg1Value*' 式には最初の引数とその値が含まれます。*arg1* という名前はデータソースに固有のもので、コネクタによって異なる場合があります。

```
SELECT * FROM TABLE(
        system.query(
            arg1 => 'arg1Value',
            arg2 => 'arg2Value',
            arg3 => 'arg3Value'
        ));
```

特定のコネクタで使用する正確な構文については、個々のコネクタのページを参照してください。

引用符の使用

引数値 (渡すクエリ文字列を含む) は、以下の例にあるように、一重引用符で囲む必要があります。

```
SELECT * FROM TABLE(system.query(query => 'SELECT * FROM testdb.persons LIMIT 10'))
```

クエリ文字列が二重引用符で囲まれていると、クエリは失敗します。以下のクエリは、COLUMN_NOT_FOUND: line 1:43: Column 'select * from testdb.persons limit 10' cannot be resolved というエラーメッセージで失敗します。

```
SELECT * FROM TABLE(system.query(query => "SELECT * FROM testdb.persons LIMIT 10"))
```

一重引用符をエスケープするには、元の引用符に一重引用符を追加します (terry's_group を terry''s_group にするなど)。

例

以下のクエリ例は、データソースにクエリをプッシュダウンします。クエリは customer テーブル内のすべての列を選択し、結果を 10 個に制限します。

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10;'
        ))
```

以下のステートメントは同じクエリを実行しますが、オプションの名前付き引数 query とアロー演算子 => を削除します。

```
SELECT * FROM TABLE(
        system.query(
            'SELECT * FROM customer LIMIT 10;'
        ))
```

Athena とフェデレーションテーブル名修飾子

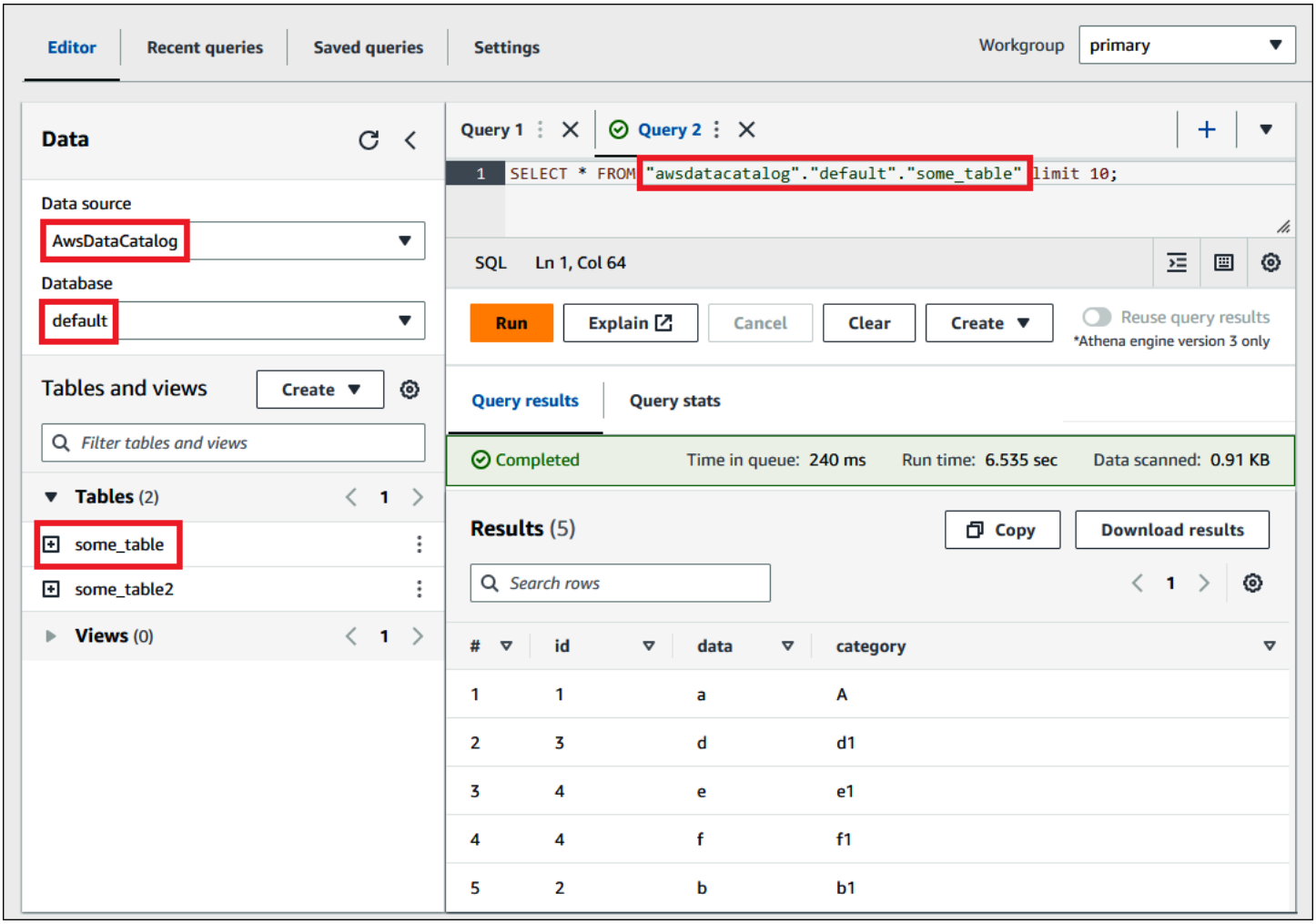
Athena はデータオブジェクトの階層を指すのに次の用語を使います。

- データソース — データベースのグループ
- データベース — テーブルのグループ
- テーブル — 行または列のグループとして整理されたデータ

これらのオブジェクトは、次のように代替だが同等の名前で呼ばれることもあります。

- データソースはカタログと呼ばれることもあります。
- データベースはスキーマと呼ばれることもあります。

Athena コンソールの次のクエリ例では、awsdatacatalog データソース、default データベース、および some_table テーブルを使用しています。



フェデレーションデータソースの用語

フェデレーションデータソースをクエリする場合、基になるデータソースが Athena と同じ用語を使用していない可能性があることに注意してください。フェデレーションクエリを作成する場合、この区別を念頭に置いてください。以下のセクションでは、Athena のデータオブジェクト用語がフェデレーションデータソースのデータオブジェクト用語とどのように対応するかを説明します。

Amazon Redshift

Amazon Redshift データベースは、Redshift テーブルのグループを含む Redshift スキーマのグループです。

Athena	Redshift
共有データソース	Redshift コネクタの Lambda 関数は、Redshift `database` を指すように設定されています。

Athena	Redshift
`data_source.database.table`	`database.schema.table`

クエリの例

```
SELECT * FROM
Athena_Redshift_connector_data_source.Redshift_schema_name.Redshift_table_name
```

このコネクタの詳細については、「Amazon Athena Redshift コネクタ」を参照してください。

Cloudera Hive

Cloudera Hive サーバーまたはクラスターは、Cloudera Hive テーブルのグループを含む Cloudera Hive データベースのグループです。

Athena	[Hive]
Cloudera Hive データソース	Cloudera Hive コネクタの Lambda 関数は Cloudera Hive `server` を指すように設定されています。
`data_source.database.table`	`server.database.table`

クエリの例

```
SELECT * FROM
Athena_Cloudera_Hive_connector_data_source.Cloudera_Hive_database_name.Cloudera_Hive_table_name
```

このコネクタの詳細については、「Amazon Athena Cloudera Hive コネクタ」を参照してください。

Cloudera Impala

Impala サーバーまたはクラスターは、Impala テーブルのグループを含む Impala データベースのグループです。

Athena	Impala
Impala データソース	Impala コネクタの Lambda 関数は Impala `server` を指すように構成されています。
`data_source.database.table`	`server.database.table`

クエリの例

```
SELECT * FROM
Athena_Impala_connector_data_source.Impala_database_name.Impala_table_name
```

このコネクタの詳細については、「Amazon Athena Cloudera Impala コネクタ」を参照してください。

MySQL

MySQL サーバーは MySQL テーブルのグループを含む MySQL データベースのグループです。

Athena	MySQL
MySQL データソース	MySQL コネクタの Lambda 関数は MySQL `server` を指すように設定されています。
`data_source.database.table`	`server.database.table`

クエリの例

```
SELECT * FROM
Athena_MySQL_connector_data source.MySQL_database_name.MySQL_table_name
```

このコネクタの詳細については、「Amazon Athena MySQL コネクタ」を参照してください。

Oracle

Oracle サーバー (またはデータベース) は、Oracle テーブルのグループを含む Oracle スキーマのグループです。

Athena	Oracle
Oracle データソース	Oracle コネクタの Lambda 関数は Oracle `server` を指すように設定されています。
`data_source.database.table`	`server.schema.table`

クエリの例

```
SELECT * FROM
Athena_Oracle_connector_data_source.Oracle_schema_name.Oracle_table_name
```

このコネクタの詳細については、「Amazon Athena Oracle コネクタ」を参照してください。

Postgres

Postgres サーバー (またはクラスター) は Postgres データベースのグループです。Postgres データベースは Postgres テーブルのグループを含む Postgres スキーマのグループです。

Athena	Postgres
Postgres データソース	Postgres コネクタの Lambda 関数は Postgres `server` と`database`を指すように設定されています。
`data_source.database.table`	`server.database.schema.table`

クエリの例

```
SELECT * FROM
Athena_Postgres_connector_data_source.Postgres_schema_name.Postgres_table_name
```

このコネクタの詳細については、「Amazon Athena PostgreSQL コネクタ」を参照してください。

Athena Query Federation SDK を使用したデータソースコネクタの開発

独自のデータソースコネクタを記述するには、Athena Query Federation SDK を使用できます。Athena Query Federation SDK は、Athena がそのクエリ実行プラインの一部をユーザーが記述

してデプロイするコードに委託するために使用できる、インターフェイスとワイヤプロトコルの一式を定義します。SDK には、コネクタスイートとコネクタの例が含まれています。

また、ユーザー独自の使用のために、Amazon Athena の事前構築されたコネクタをカスタマイズすることもできます。GitHub からのソースコードのコピーを変更し、コネクタ公開ツールを使用して独自の AWS Serverless Application Repository パッケージを作成することができます。この方法でコネクタをデプロイした後は、それを Athena クエリで使用することができます。

SDK をダウンロードする方法と、独自のコネクタを記述するための詳細な手順については、GitHub の「Example Athena connector」(Example Athena コネクタ) を参照してください。

Apache Spark 用の Athena データソースコネクタ

一部の Athena データソースコネクタは、Spark DSV2 コネクタとして利用できます。Spark DSV2 コネクタ名には `-dsv2` サフィックスが付いています (例: `athena-dynamodb-dsv2`)。

現在利用可能な DSV2 コネクタ、その Spark `.format()` クラス名、および対応する Amazon Athena Federated Query ドキュメントへのリンクを次に示します。

DSV2 コネクタ	Spark .format() クラス名	ドキュメント
athena-cloudwatch-dsv2	`com.amazonaws.athena.connectors.dsv2.cloudwatch.CloudwatchTableProvider`	CloudWatch
athena-cloudwatch-metrics-dsv2	`com.amazonaws.athena.connectors.dsv2.cloudwatch.metrics.CloudwatchMetricsTableProvider`	CloudWatch メトリクス
athena-aws-cmdb-dsv2	`com.amazonaws.athena.connectors.dsv2.aws.cmdb.AwsCmdbTableProvider`	CMDB
athena-dy	`com.amazonaws.athena.connectors.dsv2`	DynamoDB

DSV2 コネクタ	Spark .format() クラス名	ドキュメント
namodb-dsv2	`.dynamodb.DDBTableProvider`	

DSV2 コネクタ用の `.jar` ファイルをダウンロードするには、Amazon Athena Query Federation DSV2 の GitHub ページにアクセスし、「リリース」、「*<#####>* のリリース」、「アセット」セクションを参照してください。

Spark に対する jar の指定

Spark で Athena DSV2 コネクタを使用するには、使用している Spark 環境にコネクタの `.jar` ファイルを送信します。次のセクションでは、特定のケースについて説明します。

Athena for Spark

Amazon Athena for Apache Spark にカスタム `.jar` ファイルとカスタム設定を追加するための情報については、「JAR ファイルとカスタム Spark 設定を追加する」を参照してください。

一般的な Spark

コネクタ `.jar` ファイルを Spark に渡すには、次の例のように、`spark-submit` コマンドを使用し、`--jars` オプションで `.jar` ファイルを指定します。

```
spark-submit \
  --deploy-mode cluster \
  --jars https://github.com/awslabs/aws-athena-query-federation-dsv2/releases/
download/some_version/athena-dynamodb-dsv2-some_version.jar
```

Amazon EMR Spark

Amazon EMR で `--jars` パラメータを指定して `spark-submit` コマンドを実行するには、Amazon EMR Spark クラスターにステップを追加する必要があります。Amazon EMR での `spark-submit` の使用方法の詳細については、「Amazon EMR リリースガイド」の「Spark ステップを追加する」を参照してください。

AWS Glue ETL Spark

AWS Glue ETL の場合、`.jar` ファイルの GitHub.com URL を `aws glue start-job-run` コマンドの `--extra-jars` 引数に渡すことができます。AWS Glue ドキュメントでは `--extra-jars` パ

ラメータが Amazon S3 パスを取るように説明されていますが、パラメータは HTTPS URL を取ることもできます。詳細については、「AWS Glue デベロッパーガイド」の「ジョブパラメータリファレンス」を参照してください。

Spark でのコネクタに対するクエリの実行

Apache Spark で既存の Athena フェデレーテッドクエリと同等のクエリを送信するには、`spark.sql()` 関数を使用します。たとえば、Apache Spark で使用する次の Athena クエリがあるとします。

```
SELECT somecola, somecolb, somecolc
FROM ddb_datasource.some_schema_or_glue_database.some_ddb_or_glue_table
WHERE somecola > 1
```

Amazon Athena DynamoDB DSV2 コネクタを使用して Spark で同じクエリを実行するには、次のコードを使用します。

```
dynamoDf = (spark.read
    .option("athena.connectors.schema", "some_schema_or_glue_database")
    .option("athena.connectors.table", "some_ddb_or_glue_table")
    .format("com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider")
    .load())

dynamoDf.createOrReplaceTempView("ddb_spark_table")

spark.sql('''
SELECT somecola, somecolb, somecolc
FROM ddb_spark_table
WHERE somecola > 1
''')
```

パラメータの指定

DSV2 バージョンの Athena データソースコネクタは、対応する Athena データソースコネクタと同じパラメータを使用します。パラメータの情報については、対応する Athena データソースコネクタのドキュメントを参照してください。

PySpark コードでは、次の構文を使用してパラメータを設定します。

```
spark.read.option("athena.connectors.conf.parameter", "value")
```

たとえば、次のコードは Amazon Athena DynamoDB コネクタ `disable_projection_and_casing` パラメータを `always` に設定します。

```
dynamoDf = (spark.read
    .option("athena.connectors.schema", "some_schema_or_glue_database")
    .option("athena.connectors.table", "some_ddb_or_glue_table")
    .option("athena.connectors.conf.disable_projection_and_casing", "always")
    .format("com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider")
    .load())
```

データカタログにアクセスするための IAM ポリシー

データカタログへのアクセスを制御するには、リソースレベルの IAM アクセス許可、またはアイデンティティベースの IAM ポリシーを使用します。

以下の手順は、Athena に固有の手順です。

IAM 固有の情報については、このセクションの最後に表示されているリンク先を参照してください。JSON データカタログポリシーの例については、「データカタログポリシーの例」を参照してください。

IAM コンソールのビジュアルエディタを使用してデータカタログポリシーを作成する

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[Policies] (ポリシー)、[Create policy] (ポリシーの作成) の順にクリックします。
3. [Visual editor] (ビジュアルエディタ) タブで、[Choose a service] (サービスの選択) をクリックします。次に、ポリシーに追加する Athena を選択します。
4. [Select actions] (アクションの選択) を選択し、ポリシーに追加するアクションを選択します。ビジュアルエディタが Athena で利用できるアクションを表示します。詳細については、「サービス承認リファレンス」の「Amazon Athena 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。
5. [Add actions] (アクションの追加) をクリックして特定のアクションを入力、またはワイルドカード (*) を使用して複数のアクションを指定します。

   デフォルトでは、作成しているポリシーが選択するアクションを許可します。Athena 内の `datacatalog` リソースに対するリソースレベルのアクセス許可をサポートするアクションを 1 つ、または複数選択すると、エディタが `datacatalog` リソースをリストします。

6. [Resources] (リソース) をクリックして、ポリシーの特定のデータカタログを指定します。JSON データカタログポリシーの例については、「データカタログポリシーの例」を参照してください。
7. 以下のように datacatalog リソースを指定します。

```
arn:aws:athena:<region>:<user-account>:datacatalog/<datacatalog-name>
```

8. [Review policy] (ポリシーの確認) をクリックして、作成するポリシーの [Name] (名前) と [Description (説明) (オプション) を入力します。ポリシー概要を確認して、意図したアクセス許可を付与したことを確認します。
9. [Create Policy] (ポリシーの作成) をクリックして、新しいポリシーを保存します。
10. このアイデンティティベースのポリシーをユーザー、グループ、またはロールにアタッチし、それらのユーザー、グループ、またはロールがアクセスできる datacatalog リソースを指定します。

詳細については、サービス認証リファレンスと IAM ユーザーガイドで以下のトピックを参照してください。

- Amazon Athena のアクション、リソース、および条件キー
- ビジュアルエディタでのポリシーの作成
- IAM ポリシーの追加と削除
- リソースへのアクセスの制御

JSON データカタログポリシーの例については、「データカタログポリシーの例」を参照してください。

AWS Glue のアクセス許可および AWS Glue クローラーのアクセス許可の詳細については、「AWS Glue デベロッパーガイド」の「AWS Glue 用の IAM アクセス許可のセットアップ」および「クローラーの前提条件」を参照してください。

Amazon Athena アクションの完全なリストについては、Amazon Athena API Reference で API アクション名を参照してください。

データカタログポリシーの例

このセクションには、データカタログに対するさまざまなアクションを有効にするために使用できるポリシーの例が含まれています。

データカタログは、Athena によって管理されている IAM リソースです。そのため、データカタログポリシーで datacatalog を入力として取るアクションを使用する場合、データカタログの ARN を次のように指定する必要があります。

```
"Resource": [arn:aws:athena:<region>:<user-account>:datacatalog/<datacatalog-name>]
```

<datacatalog-name> は、データカタログの名前です。例えば、test_datacatalog という名前のデータカタログの場合は、次のようにリソースとして指定します。

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:datacatalog/test_datacatalog"]
```

Amazon Athena アクションの完全なリストについては、Amazon Athena API Reference で API アクション名を参照してください。IAM ポリシーの詳細については、「IAM ユーザーガイド」で「ビジュアルエディタでのポリシーの作成」を参照してください。ワークグループ用の IAM ポリシーの作成に関する詳細については、「 データカタログにアクセスするための IAM ポリシー」を参照してください。

- Example Policy for Full Access to All Data Catalogs
- Example Policy for Full Access to a Specified Data Catalog
- Example Policy for Querying a Specified Data Catalog
- Example Policy for Management Operations on a Specified Data Catalog
- Example Policy for Listing Data Catalogs
- Example Policy for Metadata Operations on Data Catalogs

Example すべてのデータカタログへの完全なアクセスのためのポリシー例

以下のポリシーは、アカウントに存在している可能性があるすべてのデータカタログリソースへの完全なアクセスを許可します。アカウントの他のすべてのユーザーのデータカタログを管理する必要があるユーザーにこのポリシーを使用することをお勧めします。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:*"
         ],
```

```
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Example 指定されたデータカタログへの完全なアクセスのためのポリシー例

次のポリシーでは、datacatalogA という名前の 1 つの特定のデータカタログリソースへのフルアクセスを許可します。特定のデータカタログを完全に制御しているユーザーにこのポリシーを使用できます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:ListDataCatalogs",
            "athena:ListWorkGroups",
            "athena:GetDatabase",
            "athena:ListDatabases",
            "athena:ListTableMetadata",
            "athena:GetTableMetadata"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "athena:StartQueryExecution",
            "athena:GetQueryResults",
            "athena:DeleteNamedQuery",
            "athena:GetNamedQuery",
            "athena:ListQueryExecutions",
            "athena:StopQueryExecution",
            "athena:GetQueryResultsStream",
            "athena:ListNamedQueries",
            "athena:CreateNamedQuery",
            "athena:GetQueryExecution",
            "athena:BatchGetNamedQuery",
            "athena:BatchGetQueryExecution",
```

```
            "athena:DeleteWorkGroup",
            "athena:UpdateWorkGroup",
            "athena:GetWorkGroup",
            "athena:CreateWorkGroup"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:workgroup/*"
         ]
      },
      {
        "Effect":"Allow",
        "Action":[
            "athena:CreateDataCatalog",
            "athena:DeleteDataCatalog",
            "athena:GetDataCatalog",
            "athena:GetDatabase",
            "athena:GetTableMetadata",
            "athena:ListDatabases",
            "athena:ListTableMetadata",
            "athena:UpdateDataCatalog"
        ],
        "Resource":"arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
      }
   ]
}
```

Example 指定されたデータカタログをクエリするためのポリシー例

次のポリシーでは、ユーザーは指定した datacatalogA に対してクエリを実行することを許可されています。ユーザーは、データカタログの更新や削除など、データカタログ自体の管理タスクを実行することはできません。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:StartQueryExecution"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:workgroup/*"
         ]
```

```
      },
      {
         "Effect":"Allow",
         "Action":[
            "athena:GetDataCatalog"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
         ]
      }
   ]
}
```

Example 指定されたデータカタログに対する管理オペレーションのためのポリシー例

次のポリシーでは、ユーザーはデータカタログ datacatalogA の作成、削除、詳細の取得、および更新を許可されています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:UpdateDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
            ]
        }
    ]
}
```

Example データカタログをリスト化するためのポリシー例

以下のポリシーは、すべてのユーザーがすべてのデータカタログをリストアップすることを許可します。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListDataCatalogs"
            ],
            "Resource": "*"
        }
    ]
}
```

Example データカタログでのメタデータオペレーションのためのポリシー例

以下のポリシーは、データカタログに対するメタデータオペレーションを許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetDatabase",
                "athena:GetTableMetadata",
                "athena:ListDatabases",
                "athena:ListTableMetadata"
            ],
            "Resource": "*"
        }
    ]
}
```

データソースの管理

Athena コンソールの [Data Sources] (データソース) ページを使用して、作成したデータソースを管理できます。

データソースを表示する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. ナビゲーションペインで、[Data source] (データソース) をクリックします。
4. データソースのリストから、表示するデータソースの名前を選択します。

> **Note**
>
> [Data source name] (データソース名) 列の項目は、ListDataCatalogs API アクションと list-data-catalogs CLI コマンドの出力に対応しています。

データソースを編集する

1. [Data sources] (データソース) ページで、以下のいずれかの作業を行います。
   - カタログ名の横にあるボタンをクリックした後、[Actions] (アクション)、[Edit] (編集) の順にクリックします。
   - データソースの名前を選択します。その後、詳細ページで、[Actions] (アクション)、[Edit] (編集) の順にクリックします。
2. [Edit] (編集) ページでは、データソースのために異なる Lambda 関数を選択することや、説明の変更、またはカスタムタグの追加などが行えます。タグの詳細については、Athena リソースへのタグ付けを参照してください。
3. [Save] を選択します。
4. [AwsDataCatalog] データソースを編集するには、[AwsDataCatalog] のリンクをクリックして、詳細ページを開きます。次にその詳細ページで、カタログを編集する AWS Glue コンソールへのリンクをクリックします。

データソースを共有するには

データソース共有に関する情報ついては、以下のリンクをご覧ください。

- Hive 以外の Lambda ベースのデータソースについては、「クロスアカウントのフェデレーションクエリの有効化」を参照してください。
- AWS Glue Data Catalog を使用する場合は、「AWS Glue データカタログへのクロスアカウントアクセス」を参照してください。

データソースを削除するには

1. [Data sources] (データソース) ページで、以下のいずれかの作業を行います。
   - カタログ名の横にあるボタンをクリックした上で、[Actions] (アクション)、[Delete] (削除) の順にクリックします。
   - データソースの名前を選択し、次に詳細ページで [Actions] (アクション)、[Delete] (削除) の順にクリックします。

   > **Note**
   >
   > AwsDataCatalog はアカウントにおけるデフォルトのデータソースなので、これを削除することはできません。

   データソース、それに対応するデータカタログ、テーブルを削除すると警告が表示され、クエリエディタからはビューが削除されます。このデータソースを使用している保存済みのクエリは、Athena で実行できなくなります。
2. 削除を確認する場合は、データソースの名前を入力した上で [Delete] (削除) をクリックします。

Athena で Amazon DataZone を使用する

「Amazon DataZone」を使用して、組織の境界を越えてデータを大規模に共有、検索、検出できます。DataZone は、Athena、AWS Glue、AWS Lake Formation などの AWS 分析サービス全体のエクスペリエンスを簡素化します。例えば、さまざまなデータソースに数ペタバイトのデータがある場合、Amazon DataZone を使用して、人、データ、ツールをビジネスユースケースに基づいてグループ化できます。詳細については、「Amazon DataZone とは?」を参照してください。

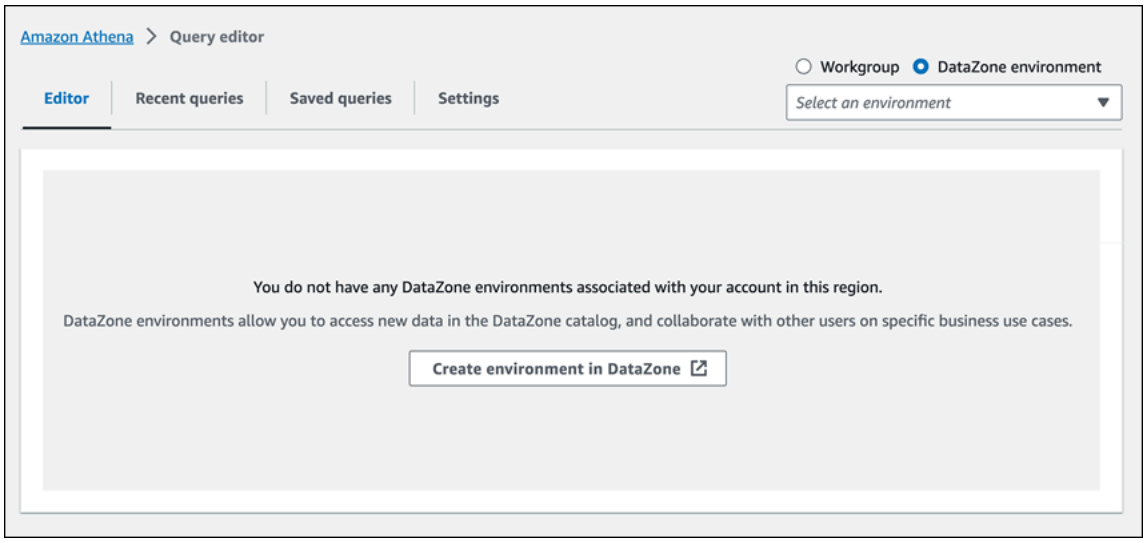
Athena では、クエリエディターを使用して DataZone 環境にアクセスしてクエリを実行できます。DataZone 環境では、DataZone プロジェクトとドメインの組み合わせが指定されます。Athena コンソールから DataZone 環境を使用するとき、DataZone 環境の IAM ロールを引き継ぎ、その環境に属するデータベースおよびテーブルのみが表示されます。許可は DataZone で指定したロールによって決まります。

Athena では、クエリエディタページの [DataZone 環境] セレクターを使用して DataZone 環境を選択できます。

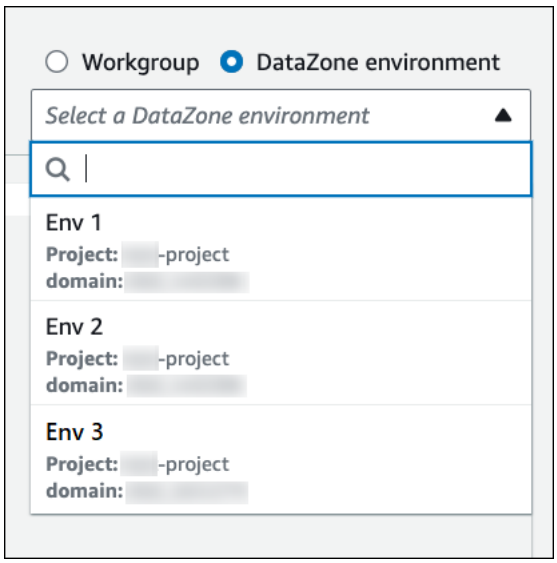
Athena で DataZone 環境を開く方法

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. Athena コンソールの右上にある [ワークグループ] の横で、[DataZone 環境] を選択します。

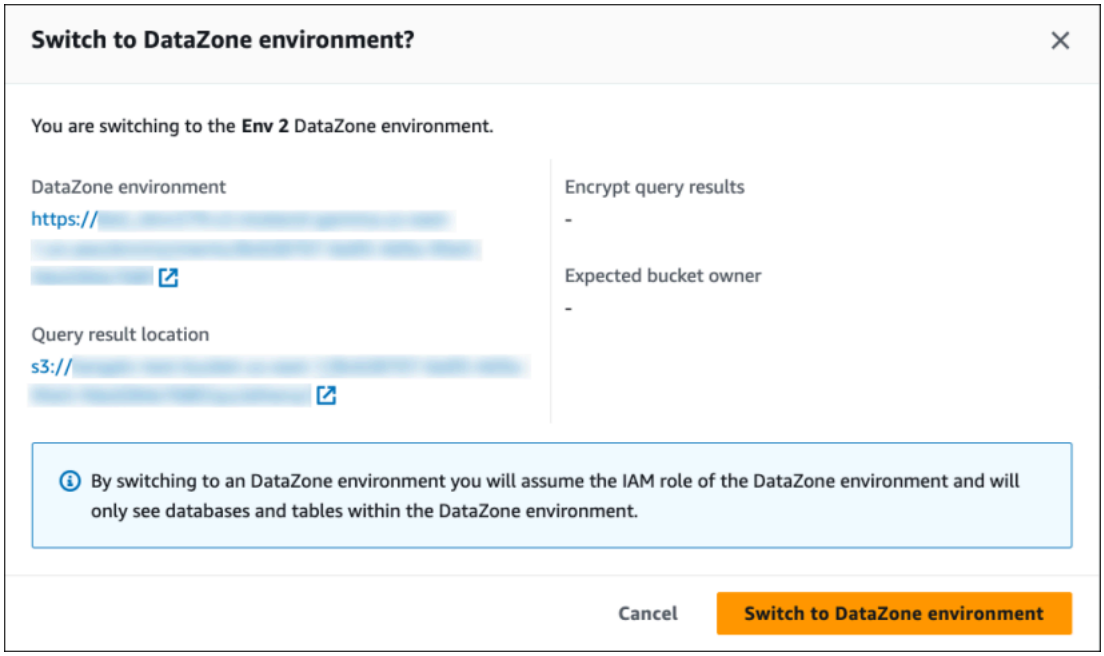
> **Note**
>
> [DataZone 環境] オプションは、DataZone で利用可能なドメインが 1 つ以上ある場合にのみ表示されます。



3. [DataZone 環境] セレクタを使用して DataZone 環境を選択します。



4. [DataZone 環境に切り替え] ダイアログボックスで目的の環境が表示されていることを確認し、[DataZone 環境に切り替え] を選択します。



DataZone および Athena の開始方法の詳細については、「Amazon DataZone ユーザーガイド」の「入門」チュートリアルを参照してください。

ODBC および JDBC ドライバーを使用した Amazon Athena への接続

ビジネスインテリジェンスツールを使用してデータを探索し、視覚化するには、ODBC (Open Database Connectivity) ドライバーまたは JDBC (Java Database Connectivity) ドライバーをダウンロード、インストール、および設定します。

トピック

- JDBC を使用した Amazon Athena への接続
- ODBC を使用した Amazon Athena への接続

以下の「AWS ナレッジセンター」と「AWS ビッグデータブログのトピック」も参照してください。

- JDBC ドライバーを使用して Athena に接続するときに、独自の IAM ロール認証情報を使用する、または別の IAM ロールに切り替える方法を教えてください。
- ADFS と AWS の信頼の確立、およびアクティブディレクトリ認証情報を使用した Amazon Athena と ODBC ドライバーの接続

JDBC を使用した Amazon Athena への接続

Amazon Athena には、バージョン 2.x と 3.x の 2 つの JDBC ドライバーが用意されています。Athena JDBC 3.x ドライバーは、より優れたパフォーマンスと互換性を提供する新世代ドライバーです。JDBC 3.x ドライバーは、Amazon S3 からの直接的なクエリ結果の読み取りをサポートするので、大規模なクエリ結果を使用するアプリケーションのパフォーマンスが向上します。新しいドライバーではサードパーティーとの依存関係も少なくなっているため、BI ツールやカスタムアプリケーションとの統合が容易になります。ほとんどの場合、新しいドライバーは既存の設定をまったく変更しない、または最小限の変更を行うだけで使用できます。

- JDBC 3.x ドライバーをダウンロードするには、「Athena JDBC 3.x ドライバー」を参照してください。
- JDBC 2.x ドライバーをダウンロードするには、「Athena JDBC 2.x ドライバー」を参照してください。

トピック

- Athena JDBC 3.x ドライバー
- Athena JDBC 2.x ドライバー

Athena JDBC 3.x ドライバー

Athena JDBC ドライバーを使用し、多くのサードパーティ SQL クライアントツールおよびカスタムアプリケーションから Amazon Athena に接続できます。

システム要件

- Java 8 (またはそれ以降) のランタイム環境
- 最低 20 MB の空きディスク容量。

考慮事項と制約事項

次の内容では、Athena JDBC 3.x ドライバーの考慮事項と制限事項の一部が示されます。

- ログ — 3.x ドライバーは SLF4J を使用します。これは、ランタイム時にいくつかのログシステムのいずれかを使用できるようにする抽象化レイヤーです。
- 暗号化 — CSE_KMS 暗号化オプションで Amazon S3 フェッチャーを使用するとき、Amazon S3 クライアントは Amazon S3 バケットに保存される結果を復号化できません。CSE_KMS 暗号化が必要な場合、ストリーミングフェッチャーを引き続き使用できます。Amazon S3 フェッチャーによる CSE_KMS 暗号化のサポートが計画されています。

JDBC 3.x ドライバーのダウンロード

このセクションには、JDBC 3.x ドライバーのダウンロードおよびライセンス情報が含まれています。

> **Important**
>
> JDBC 3.x ドライバーを使用するとき、次の要件に注意してください。
>
> - オープン状態のポート 444 – Athena がクエリ結果のストリーミングに使用するポート 444 には、アウトバウンドトラフィックに対して開放されている状態を維持します。PrivateLink エンドポイントを使用して Athena に接続するときは、PrivateLink エンドポイントにアタッチされているセキュリティグループが、ポート 444 上のインバウンドトラフィックに対して開放されていることを確認してください。

- athena:GetQueryResultsStream ポリシー – JDBC ドライバーを使用する IAM プリンシパルに `athena:GetQueryResultsStream` ポリシーアクションを追加します。このポリシーアクションが API で直接公開されることはありません。ストリーミング結果のサポートの一環として、ODBC および JDBC ドライバーでのみ使用されます。ポリシーの例については、「AWS 管理ポリシー: AWSQuicksightAthenaAccess」を参照してください。

Amazon Athena 3.x JDBC ドライバーをダウンロードするには、次のリンクにアクセスしてください。

JDBC ドライバー uber jar

次のダウンロードは、ドライバーおよびそのすべての依存関係を同じ `.jar` ファイルにパッケージ化します。このダウンロードは、サードパーティーの SQL クライアントでよく使用されます。

3.2.0 uber jar

JDBC ドライバーの lean jar

次のダウンロードには、ドライバー用の lean `.jar` およびドライバーの依存関係用の個別 `.jar` ファイルを含む `.zip` ファイルです。このダウンロードは、ドライバーが使用する依存関係と競合する依存関係を持つカスタムアプリケーションによく使用されます。このダウンロードは、ドライバーの依存関係のどっちを lean jar に含め、カスタムアプリケーションに既に 1 つ以上含まれている場合はどの依存関係を除外するかを選択する場合に便利です。

3.2.0 lean jar

ライセンス

次のリンクには、JDBC 3.x ドライバーの使用許諾契約が含まれています。

License

トピック

- JDBC 3.x ドライバーの開始方法
- Amazon Athena JDBC 3.x 接続パラメータ
- その他の JDBC 3.x 設定
- Amazon Athena ODBC 3.x リリースノート

- 以前のバージョンの Athena JDBC 3.x ドライバー

JDBC 3.x ドライバーの開始方法

このセクションの情報を使用して Amazon Athena JDBC 3.x ドライバーを開始します。

トピック

- インストール手順
- ドライバーの実行
- ドライバーの設定
- Athena JDBC v2 ドライバーからのアップグレード

インストール手順

JDBC 3.x ドライバーは、カスタムアプリケーションまたはサードパーティの SQL クライアントで使用できます。

カスタムアプリケーションの場合

ドライバー jar とその依存関係を含む `.zip` ファイルをダウンロードします。依存関係にはそれぞれ独自の `.jar` ファイルがあります。ドライバー jar を依存関係としてカスタムアプリケーションに追加します。別のソースからアプリケーションにその依存関係を既に追加しているかどうかに基づいて、ドライバー jar の依存関係を選択的に追加します。

サードパーティ製 SQL クライアントの場合

ドライバーの uber jar ファイルをダウンロードし、サードパーティ SQL クライアントの指示に従ってそのクライアントに追加します。

ドライバーの実行

ドライバーを実行するには、カスタムアプリケーションまたはサードパーティの SQL クライアントを使用できます。

カスタムアプリケーションの場合

JDBC インターフェイスを使用してプログラムから JDBC ドライバーを操作します。次のコードは、カスタム Java アプリケーションの例を示しています。

```
public static void main(String args[]) throws SQLException {
    Properties connectionParameters = new Properties();
    connectionParameters.setProperty("Workgroup", "primary");
    connectionParameters.setProperty("Region", "us-east-2");
    connectionParameters.setProperty("Catalog", "AwsDataCatalog");
    connectionParameters.setProperty("Database","sampledatabase");
    connectionParameters.setProperty("OutputLocation","s3://DOC-EXAMPLE-BUCKET");
    connectionParameters.setProperty("CredentialsProvider","DefaultChain");
    String url = "jdbc:athena://";
    AthenaDriver driver = new AthenaDriver();
    Connection connection = driver.connect(url, connectionParameters);
    Statement statement = connection.createStatement();
    String query = "SELECT * from sample_table LIMIT 10";
    ResultSet resultSet = statement.executeQuery(query);
    printResults(resultSet); // A custom-defined method for iterating over a
                             // result set and printing its contents
}
```

サードパーティ製 SQL クライアントの場合

使用している SQL クライアントのマニュアルに従ってください。通常は、SQL クライアントのグラフィカルユーザーインターフェイスを使用してクエリを入力および送信します。クエリ結果は同じインターフェイスに表示されます。

ドライバーの設定

接続パラメータを使用して Amazon Athena JDBC ドライバーを設定できます。サポートされている接続パラメータについては、「Amazon Athena JDBC 3.x 接続パラメータ」を参照してください。

カスタムアプリケーションの場合

カスタムアプリケーションの JDBC ドライバーの接続パラメータを設定するには、次のいずれかの操作を行います。

- パラメータ名およびその値を `Properties` オブジェクトに追加します。`Connection#connect` を呼び出すとき、そのオブジェクトを URL と一緒に渡します。例については、ドライバーの実行のサンプルの Java アプリケーションを参照してください。
- 接続文字列 (URL) では、次の形式を使用してパラメータ名およびその値をプロトコルプレフィックスの直後に追加します。

```
<parameterName>=<parameterValue>;
```

次の例のように、各パラメーター名/パラメータ値のペアの末尾にはセミコロンを使用します。セミコロンの後には、空白を入れないでください。

```
String url = "jdbc:athena://WorkGroup=primary;Region=us-east-1;...;";AthenaDriver
 driver = new AthenaDriver();Connection connection = driver.connect(url, null);
```

> **Note**
>
> パラメータを接続文字列および Properties オブジェクトの両方で指定する場合、接続文字列の値が優先されます。両方に同じパラメータを指定することはお勧めしません。

- 次の例のように、パラメータ値を引数として AthenaDataSource のメソッドに追加します。

```
AthenaDataSource dataSource = new AthenaDataSource();
    dataSource.setWorkGroup("primary");
    dataSource.setRegion("us-east-2");
    ...
    Connection connection = dataSource.getConnection();
    ...
```

サードパーティ製 SQL クライアントの場合

使用している SQL クライアントの指示に従ってください。通常、クライアントはパラメータ名とその値を入力するグラフィカルユーザーインターフェイスを用意します。

Athena JDBC v2 ドライバーからのアップグレード

JDBC バージョン 3 の接続パラメータのほとんどは、バージョン 2 (Simba) JDBC ドライバーとの下位互換性があります。つまり、バージョン 2 の接続文字列はバージョン 3 のドライバーで再利用できることを意味します。ただし、一部の接続パラメータは変更されています。これらの変更は本書で説明されています。バージョン 3 の JDBC ドライバーにアップグレードするときは、必要に応じて既存の設定を更新してください。

ドライバークラス

一部の BI ツールでは、JDBC ドライバー .jar ファイルからドライバークラスを指定するように求められます。ほとんどのツールはこのクラスを自動的に見つけます。バージョン 3 ドライバーのクラスの完全修飾名は com.amazon.athena.jdbc.AthenaDriver です。バージョン 2 ドライバーでは、クラスは com.simba.athena.jdbc.Driver でした。

接続文字列

バージョン 3 ドライバーは、JDBC 接続文字列 URL の先頭にあるプロトコルに `jdbc:athena://` を使用します。バージョン 3 ドライバーはバージョン 2 プロトコル `jdbc:awsathena://` もサポートしていますが、バージョン 2 プロトコルの使用は非推奨になっています。定義されていない動作を避けるため、バージョン 3 は、バージョン 2 (または `jdbc:awsathena://` で始まる接続文字列を受け入れるその他のドライバー) が DriverManager クラスで登録されている場合、`jdbc:awsathena://` で始まる接続文字列を受け入れません。

認証情報プロバイダー

バージョン 2 ドライバーは、完全修飾名を使用してさまざまな認証情報プロバイダー (例: `com.simba.athena.amazonaws.auth.DefaultAWSCredentialsProviderChain`) を特定します。バージョン 3 ドライバーは短い名前 (例: `DefaultChain`) を使用します。新しい名前は、各認証情報プロバイダーの対応するセクションで説明されています。

以前の AWS SDK for Java から AWSCredentialsProvider インターフェイスを実装するのではなく、新しい AWS SDK for Java から AwsCredentialsProvider インターフェイスを実装するには、バージョン 2 ドライバー用に作成されたカスタム認証情報プロバイダーをバージョン 3 ドライバー用に変更する必要があります。

`PropertiesFileCredentialsProvider` は、JDBC 3.x ドライバーではサポートされていません。このプロバイダーは JDBC 2.x ドライバーで使用されていましたが、サポート終了間近の AWS SDK for Java の以前のバージョンに属しています。JDBC 3.x ドライバーで同じ機能を実現するには、代わりに AWS 設定プロファイルの認証情報 プロバイダーを使用してください。

ログレベル

次のテーブルには、JDBC バージョン 2 とバージョン 3 のドライバーにある `LogLevel` パラメータの違いが示されています。

JDBC ドライバーバージョン	パラメータ名	パラメータタイプ	デフォルト値	使用できる値	接続文字列の例
v2	`LogLevel`	オプションです。	0	0~6	`LogLevel=6;`
v3	`LogLevel`	オプションです。	TRACE	OFF、ERROR、WARN、INF	`LogLevel=INFO;`

JDBC ドライバーバージョン	パラメータ名	パラメータタイプ	デフォルト値	使用できる値	接続文字列の例
				O、DEBUG、TRACE	

クエリ ID の取得

バージョン 2 ドライバーでは、Statement インスタンスを com.interfaces.core.IStatementQueryInfoProvider にアンラップします。これは #getPReparedQueryId および #getQueryId の 2 つのメソッドを持つインターフェイスです。これらのメソッドを使用し、実行されたクエリのクエリ実行 ID を取得できます。

バージョン 3 ドライバーでは、Statement、PreparedStatement、ResultSet インスタンスを com.amazon.athena.jdbc.AthenaResultSet インターフェイスにアンラップします。このインターフェイスには #getQueryExecutionId という 1 つのメソッドがあります。

Amazon Athena JDBC 3.x 接続パラメータ

サポートされている接続パラメータはここで 基本的な接続パラメータ、詳細接続パラメータ、認証接続パラメータ の 3 つのセクションに分かれています。[詳細接続パラメータ] および [認証接続パラメータ] セクションには、関連するパラメータをまとめたサブセクションがあります。

トピック

- 基本的な接続パラメータ
- 詳細接続パラメータ
- 認証接続パラメータ

基本的な接続パラメータ

次のセクションでは、JDBC 3.x ドライバーの基本的な接続パラメータについて説明します。

リージョン

クエリが実行される AWS リージョン。リージョンのリストについては、「Amazon Athena エンドポイントとクォータ」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
リージョン	AwsRegion (廃止)	必須 (ただし、指定しない場合は DefaultAwsRegionProviderChain を使用して検索されます)	なし

カタログ

ドライバーでアクセスされるデータベースおよびテーブルを含むカタログ。データカタログの詳細については、「DataCatalog」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
カタログ	なし	オプションです。	AwsDataCatalog

データベース

クエリが実行されるデータベース。データベース名で明示的に修飾されていないテーブルは、このデータベースに解決されます。データベースの詳細については、「Database」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
データベース	スキーマ	オプションです。	デフォルト

Workgroup

クエリが実行されるワークグループ。ワークグループの詳細については、「WorkGroup」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
WorkGroup	なし	オプションです。	プライマリー

出力場所

クエリ結果が保存される Amazon S3 内の場所。出力場所の詳細については、「ResultConfiguration」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
OutputLocation	S3OutputLocation (廃止)	必須 (ワークグループが出力場所を指定している場合を除く)	なし

詳細接続パラメータ

次のセクションでは、JDBC 3.x ドライバーの詳細接続パラメータについて説明します。

トピック

- 結果の暗号化パラメータ
- 結果フェッチパラメータ
- クエリ結果の再利用パラメータ
- クエリ実行ポーリングパラメータ
- エンドポイントオーバーライドパラメータ
- プロキシ設定パラメータ
- ログパラメータ
- アプリケーション名
- 接続テスト
- 再試行回数

結果の暗号化パラメータ

以下の点に注意してください。

- AWS KMS キーは、`EncryptionOption` が `SSE_KMS` または `CSE_KMS` の場合に指定する必要があります。
- AWS KMS キーは、`EncryptionOption` が指定されていないか、`EncryptionOption` が `SSE_S3` の場合に指定できません。

暗号化オプション

Amazon S3 に保存されている前提でクエリ結果に使用される暗号化のタイプ。クエリ結果の暗号化の詳細については、「Amazon Athena API リファレンス」の「EncryptionConfiguration」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用できる値
EncryptionOption	S3OutputEncOption (廃止)	オプションです。	なし	SSE_S3、SSE_KMS、CSE_KMS

KMS キー

KMS キーの ARN または ID (SSE_KMS または CSE_KMS が暗号化オプションとして選択されている場合)。暗号化オプションの詳細については、「Amazon Athena API リファレンス」の「EncryptionConfiguration」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
KmsKey	S3OutputEncKMSKey (廃止)	オプションです。	なし

結果フェッチパラメータ

結果フェッチャー

クエリ結果のダウンロードに使用されるフェッチャー。

デフォルトの結果フェッチャーの S3 は、Athena API を使用せずに Amazon S3 から直接クエリ結果をダウンロードします。ほとんどの場合、これは最速のオプションです。クエリ結果が CSE_KMS で暗号化されているか、クエリ結果へのユーザーアクセスを許可するポリシーが s3:CalledVia を使用して Athena からの呼び出しのみを許可している場合、このオプションは利用できません。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用できる値
ResultFetcher	なし	オプションです。	S3	S3、GetQueryResults、GetQueryResultsStream

> **Note**
>
> JDBC 2.x ドライバーでは、`UseResultsetStreaming = 1` 設定が、result set streaming API を使用するようにドライバーを設定します。JDBC 3.x ドライバーの同等の設定は `ResultFetcher=GetQueryResultsStream` になります。

フェッチサイズ

このパラメータの値は、内部バッファの最小値として使用され、結果をフェッチする際のターゲットページサイズとして使用されます。0 (ゼロ) の値は、ドライバーは以下で説明するデフォルト値を使用する必要があることを意味します。最大値は 1,000,000 です。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
FetchSize	RowsToFetchPerBlock (廃止)	オプションです。	0

- `GetQueryResults` フェッチャーは API 呼び出しでサポートされる最大値である 1,000 のページサイズを常に使用します。フェッチサイズが 1,000 を超えると、最小サイズを超えるバッファを埋めるために連続して複数の API 呼び出しが行われます。
- `GetQueryResultsStream` フェッチャーは、設定されたフェッチサイズをページサイズとして使用するか、デフォルトの 10,000 を使用します。
- `S3` フェッチャーは、設定されたフェッチサイズをページサイズとして使用するか、デフォルトの 10,000 を使用します。

クエリ結果の再利用パラメータ

結果の再利用を有効にする

クエリ実行時に同じクエリによる以前の結果を再利用できるかどうかを指定します。クエリ結果の再利用の詳細については、「ResultReuseByAgeConfiguration」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
EnableResultReuseByAge	なし	オプションです。	FALSE

結果再利用の最大有効期間

Athena が再利用を考慮するべき以前のクエリ結果の最大有効期間 (分単位)。結果再利用の最大有効期間の詳細については、「ResultReuseByAgeConfiguration」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
MaxResultReuseAgeInMinutes	なし	オプションです。	60

クエリ実行ポーリングパラメータ

クエリ実行の最小ポーリング間隔

Athena にクエリ実行ステータスをポーリングする前に待機する最小時間 (ミリ秒単位)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
MinQueryExecutionPollingIntervalMillis	MinQueryExecutionPollingInterval (廃止)	オプションです。	100

クエリ実行の最大ポーリング間隔

Athena にクエリ実行ステータスをポーリングする前に待機する最大時間 (ミリ秒単位)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
MaxQueryExecutionPollingIntervalMillis	MaxQueryExecutionPollingInterval (廃止)	オプションです。	5000

クエリ実行ポーリング間隔の乗数

ポーリング期間を延長する要素。デフォルトでは、ポーリングは `MinQueryExecutionPollingIntervalMillis` の値から始まり、`MaxQueryExecutionPollingIntervalMillis` の値に達するまでポーリングごとに倍増します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
QueryExecutionPollingIntervalMultiplier	なし	オプションです。	2

エンドポイントオーバーライドパラメータ

Athena エンドポイントオーバーライド

ドライバーが Athena に API 呼び出しを行うために使用するエンドポイント。

以下の点に注意してください。

- 提示された URL に `https://` または `http://` プロトコルが指定されていない場合、ドライバーは `https://` プレフィックスを挿入します。
- このパラメータが指定されていない場合、ドライバーはデフォルトエンドポイントを使用します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AthenaEndpoint	EndpointOverride (廃止)	オプションです。	なし

Athena ストリーミングサービスエンドポイントのオーバーライド

ドライバーが Athena ストリーミングサービスを使用するとき、クエリ結果をダウンロードするために使用するエンドポイント。Athena ストリーミングサービスはポート 444 で利用できます。

以下の点に注意してください。

- 提示された URL に `https://` または `http://` プロトコルが指定されていない場合、ドライバーは `https://` プレフィックスを挿入します。
- 提示された URL にポートが指定されていない場合、ドライバーはストリーミングサービスのポート 444 を挿入します。
- `AthenaStreamingEndpoint` パラメータが指定されていない場合、ドライバーは `AthenaEndpoint` オーバーライドを使用します。`AthenaStreamingEndpoint` オーバーライドも、`AthenaEndpoint` オーバーライドも指定されていない場合、ドライバーはデフォルトのストリーミングエンドポイントを使用します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AthenaStreamingEndpoint	StreamingEndpointOverride (廃止)	オプションです。	なし

LakeFormation エンドポイントオーバーライド

AWS Lake Formation AssumeDecoratedRoleWithSAML API を使用して一時的な認証情報を取得するとき、ドライバーが Lake Formation サービスに使用するエンドポイント。このパラメータが指定されていない場合、ドライバーはデフォルトの Lake Formation エンドポイントを使用します。

以下の点に注意してください。

- 提示された URL に `https://` または `http://` プロトコルが指定されていない場合、ドライバーは `https://` プレフィックスを挿入します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LakeFormationEndpoint	LfEndpointOverride (廃止)	オプションです。	なし

S3 エンドポイントオーバーライド

ドライバーが Amazon S3 フェッチャーを使用するとき、クエリ結果をダウンロードするために使用するエンドポイント。このパラメータが指定されていない場合、ドライバーはデフォルトの Amazon S3 エンドポイントを使用します。

以下の点に注意してください。

- 提示された URL に `https://` または `http://` プロトコルが指定されていない場合、ドライバーは `https://` プレフィックスを挿入します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
S3Endpoint	なし	オプションです。	なし

STS エンドポイントオーバーライド

AWS STSAssumeRoleWithSAML API を使用して一時的な認証情報を取得するとき、ドライバーが AWS STS サービスに使用するエンドポイント。このパラメータが指定されていない場合、ドライバーはデフォルトの AWS STS エンドポイントを使用します。

以下の点に注意してください。

- 提示された URL に `https://` または `http://` プロトコルが指定されていない場合、ドライバーは `https://` プレフィックスを挿入します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
StsEndpoint	StsEndpointOverride (廃止)	オプションです。	なし

プロキシ設定パラメータ

プロキシのホスト

プロキシホストの URL。Athena がプロキシ経由でリクエストする必要がある場合、このパラメータを使用してください。

> **Note**
>
> ProxyHost の URL の先頭には必ずプロトコル https:// または http:// を含めてください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ProxyHost	なし	オプションです。	なし

プロキシのポート

プロキシホストで使用するポート。Athena がプロキシ経由でリクエストする必要がある場合、このパラメータを使用してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ProxyPort	なし	オプションです。	なし

プロキシユーザー名

プロキシサーバーで認証するユーザー名。Athena がプロキシ経由でリクエストする必要がある場合、このパラメータを使用してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ProxyUsername	ProxyUID (廃止)	オプションです。	なし

プロキシのパスワード

プロキシサーバーで認証するパスワード。Athena がプロキシ経由でリクエストする必要がある場合、このパラメータを使用してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ProxyPassword	ProxyPWD (廃止)	オプションです。	なし

プロキシ免除ホスト

プロキシが有効になっているとき (すなわち、ProxyHost および ProxyPort 接続パラメータが設定されているとき)、ドライバがプロキシを使わずに接続する一連のホスト名。ホストはパイプ (|) 文字で区切る必要があります (例えば、host1.com|host2.com)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ProxyExemptHosts	NonProxyHosts	オプションです。	なし

ID プロバイダー用に有効になっているプロキシ

ドライバーが ID プロバイダーに接続するとき、プロキシを使用するかどうかを指定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ProxyEnabledForIdP	UseProxyForIdP	オプションです。	FALSE

ログパラメータ

このセクションでは、ログに関連するパラメータについて説明します。

ログレベル

ドライバーログのレベルを指定します。LogPath パラメータも設定しない限り、何もログされません。

> **Note**
>
> 特別な要件がない限り、LogPath パラメータのみを設定することをお勧めします。LogPath パラメータのみを設定すると、ロギングが有効になってデフォルトの TRACE ログレベルが使用されます。TRACE ログレベルは最も詳細なログを提供します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用できる値
LogLevel	なし	オプションです。	TRACE	OFF、ERROR、WARN、INFO、DEBUG、TRACE

ログパス

ドライバーログが保存されるドライバーを実行するコンピューター上のディレクトリへのパス。一意の名前のログファイルが指定したディレクトリ内に作成されます。設定すると、ドライバーログが有効になります。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LogPath	なし	オプションです。	なし

アプリケーション名

ドライバーを使用するアプリケーションの名前。このパラメータの値を指定すると、その値はドライバーが Athena に対して行う API 呼び出しのユーザーエージェント文字列に含まれます。

> **Note**
>
> `DataSource` オブジェクトの `setApplicationName` を呼び出すことにより、アプリケーション名を設定することもできます。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ApplicationName	なし	オプションです。	なし

接続テスト

TRUE に設定すると、JDBC 接続でクエリが実行されなくても、ドライバは JDBC 接続が作成されるたびに接続テストを実行します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ConnectionTest	なし	オプションです。	TRUE

> **Note**
>
> 接続テストでは SELECT 1 クエリを Athena に送信し、接続が正しく設定されていることを確認します。すなわち、2 つのファイルが Amazon S3 (結果セットおよびメタデータ) に保存され、「Amazon Athena 料金表」ポリシーに従って追加料金が適用される場合があります。

再試行回数

ドライバーが再実行可能なリクエストを Athena に再送信するべき最大回数。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
NumRetries	MaxErrorRetry (廃止)	オプションです。	なし

認証接続パラメータ

Athena JDBC 3.x ドライバーは、いくつかの認証方法をサポートしています。必要な接続パラメータは、使用する認証方法によって異なります。

トピック

- IAM 認証情報
- デフォルト認証情報
- AWS 設定プロファイルの認証情報
- インスタンスプロファイルの認証情報
- カスタム認証情報

- JWT 認証情報
- Azure AD 認証情報
- Okta 認証情報
- Ping 認証情報
- AD FS 認証情報
- Browser Azure AD 認証情報
- Browser SAML 認証情報

IAM 認証情報

次の接続パラメータを設定することにより、IAM 認証情報を JDBC ドライバーと使用して Amazon Athena に接続できます。

ユーザー

AWS アクセスキー ID。アクセスキーに関する詳細については、「IAM ユーザーガイド」の「AWS セキュリティ認証情報」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ユーザー	AccessKeyId	必須	なし

パスワード

AWS シークレットキーの ID。アクセスキーに関する詳細については、「IAM ユーザーガイド」の「AWS セキュリティ認証情報」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
パスワード	SecretAccessKey	オプションです。	なし

セッショントークン

一時的な AWS 認証情報を使用している場合は、セッショントークンを指定する必要があります。一時的なセキュリティ認証情報の詳細については、「IAM ユーザーガイド」の「IAM の一時的なセキュリティ認証情報」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
[SessionToken]	なし	オプションです。	なし

デフォルト認証情報

次の接続パラメータを設定することにより、クライアントシステムで設定するデフォルトの認証情報を使用して Amazon Athena に接続できます。デフォルト認証情報の使用に関する詳細については、「AWS SDK for Java デベロッパーガイド」の「デフォルト認証情報プロバイダチェーンの使用」を参照してください。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `DefaultChain` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`DefaultChain`

AWS 設定プロファイルの認証情報

次の接続パラメータを設定することで、AWS 設定プロファイルに保存されている認証情報を使用できます。AWS 設定プロファイルは通常、「`~/.aws`」ディレクトリのファイルに保存されます)。AWS 設定プロファイルの詳細については、「AWS SDK for Java 開発者ガイド」の「プロファイルの使用」を参照してください。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `ProfileCredentials` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`ProfileCredentials`

プロファイル名

Athena へのリクエストを認証するために認証情報を使用する必要がある AWS 設定プロファイルの名前。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ProfileName	なし	必須	なし

> **Note**
>
> プロファイル名は、`CredentialsProviderArguments` パラメータの値として指定することもできますが、この使用は廃止されています。

インスタンスプロファイルの認証情報

この認証タイプは Amazon EC2 インスタンスで使用されます。「インスタンスプロファイル」Amazon EC2 インスタンスにアタッチされたプロファイルです。インスタンスプロファイル認証情報プロバイダーを使用すると、AWS 認証情報の管理を Amazon EC2 インスタンスメタデータサービスに委任されます。これにより、デベロッパーは認証情報を Amazon EC2 インスタンスに永続的に保存する必要がなくなり、一時的な認証情報のローテーションや管理について心配する必要がなくなります。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `InstanceProfile` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`InstanceProfile`

カスタム認証情報

この認証タイプを使用して、「AwsCredentialsProvider」インターフェイスを実装する Java クラスを使用して独自の認証情報を入力できます。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値は、「AwsCredentialsProvider」インターフェイスを実装するカスタムクラスの完全修飾クラス名に設定します。ランタイム時には、そのクラスは JDBC ドライバーを使用するアプリケーションの Java クラスパス上にある必要があります。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`AwsCredentialsProvider` のカスタム実装の完全修飾クラス名

認証情報プロバイダーの引数

カスタム認証情報プロバイダーコンストラクタの文字列引数のカンマ区切りリスト。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
CredentialsProviderArguments	AwsCredentialsProviderArguments (廃止)	オプションです。	なし

JWT 認証情報

この認証タイプでは、外部 ID プロバイダーから取得した JSON Web トークン (JWT) を接続パラメータとして使用して Athena で認証できます。外部認証情報プロバイダーは、AWS と既にフェデレート設定されている必要があります。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `JWT` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`JWT`

JWT ウェブ ID トークン

外部のフェデレーション ID プロバイダーから取得した JWT トークン。このトークンは Athena での認証に使用されます。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
JwtWebIdentityToken	web_identity_token (廃止)	必須	なし

JWT ロール ARN

引き受けるロールの Amazon リソースネーム (ARN)。引き受けるロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
JwtRoleArn	role_arn (廃止)	必須	なし

JWT ロールセッション名

JWT 認証情報を認証に使用するときのセッションの名前。名前は、任意の名前でかまいません。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
JwtRoleSessionName	role_session_name (廃止)	必須	なし

Azure AD 認証情報

Azure AD ID プロバイダーを使用して Athena への認証を可能にする SAML ベースの認証メカニズム。この方法では、Athena と Azure AD の間にフェデレーションが既に設定されていることを前提としています。

> **Note**
>
> このセクションの一部のパラメーター名にはエイリアスがあります。エイリアスはパラメーター名と機能的に同等であり、JDBC 2.x ドライバーとの下位互換性を保つために提供されています。パラメーター名は、より明確で一貫性のある命名規則に従うように改良されたため、非推奨になったエイリアスの代わりにパラメーター名を使用することをお勧めします。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `AzureAD` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`AzureAD`

ユーザー

Azure AD での認証に使用する Azure AD ユーザーのメールアドレス。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ユーザー	UID (廃止)	必須	なし

パスワード

Azure AD ユーザーのパスワード。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
パスワード	PWD (廃止)	必須	なし

Azure AD テナント ID

Azure AD アプリケーションのテナント ID。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AzureAdTenantId	tenant_id (廃止)	必須	なし

Azure AD クライアント ID

Azure AD アプリケーションのクライアント ID。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AzureAdClientId	client_id (廃止)	必須	なし

Azure AD クライアントシークレット

Azure AD アプリケーションのクライアントシークレット。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AzureAdClientSecret	client_secret (廃止)	必須	なし

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PreferredRole	preferred_role (廃止)	オプションです。	なし

ロールセッションの期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
RoleSessionDuration	時間 (廃止)	オプションです。	3600

Lake Formation 有効

AssumeRoleWithSAML AWS STS API アクションの代わりに、一時的な IAM 認証情報を取得するために AssumeDecoratedRoleWithSAML Lake Formation API アクションを使用するかどうかを指定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LakeFormationEnabled	なし	オプションです。	FALSE

Okta 認証情報

Okta ID プロバイダーを使用して Athena への認証を可能にする SAML ベースの認証メカニズム。この方法では、Athena と Okta の間にフェデレーションが既に設定されていることを前提としています。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `Okta` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`Okta`

ユーザー

Okta での認証に使用する Okta ユーザーのメールアドレス。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ユーザー	UID (廃止)	必須	なし

パスワード

Okta ユーザーのパスワード。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
パスワード	PWD (廃止)	必須	なし

Okta ホスト名

Okta 組織の URL。Okta アプリケーションの [埋め込みリンク] URL から、`idp_host` パラメータを抽出できます。この手順については、「Okta から ODBC の設定情報を取得する」を参照してください。`https://` の後に続く最初のセグメントから `okta.com` までは、IdP ホストです (例えば、`https://trial-1234567.okta.com` で始まる URL の `trial-1234567.okta.com`)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
OktaHostName	IdP_Host (廃止)	必須	なし

Okta アプリケーション ID

アプリケーション用の 2 つの部分で構成される識別子です。Okta アプリケーションの [リンク埋め込み] URL からアプリケーション ID を抽出できます。この手順については、「Okta から ODBC の設定情報を取得する」を参照してください。アプリケーション ID は、URL の最後の 2 つのセグメントで、中央のスラッシュも含まれます。これら 2 つのセグメントは、数字と大文字と小文字が混在する 20 文字の文字列 2 組です (例: `Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4`)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
OktaAppId	App_ID (廃止)	必須	なし

Okta アプリケーション名

Okta アプリケーションの名前。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
OktaAppName	App_Name (廃止)	必須	なし

Okta MFA タイプ

多要素認証 (MFA) を必要とするように Okta を設定した場合、使用する 2 番目の要素に応じて Okta MFA タイプと追加のパラメータを指定する必要があります。

Okta MFA タイプは、Okta での認証に使用する 2 番目の認証要素タイプです (パスワードに次ぐもの)。サポートされている 2 番目の要素には、Okta Verify アプリを介して配信されるプッシュ通知、ならびに Okta Verify や Google Authenticator によって生成または SMS を介して送信される一時的なワンタイムパスワード (TOTP) が含まれています。個々の組織のセキュリティポリシーによって、ユーザーのログインに MFA が必要かどうかが決まります。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用できる値
OktaMfaType	okta_mfa_type (廃止)	Okta が MFA を必要とするように設定されている場合は必須です	なし	`oktaverifywithpush` , `oktaverifywithtotp` , `googleauthenticator` , `smsauthentication`

Okta 電話番号

`smsauthentication` MFA タイプが選択されたとき、Okta が SMS を使用して一時的なワンタイムパスワードを送信する電話番号。電話番号は、米国またはカナダの電話番号である必要があります。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
OktaPhoneNumber	okta_phone_number (廃止)	OktaMfaType が `smsauthentication` の場合は必須	なし

Okta MFA の待機時間

ドライバーがタイムアウト例外を発生させる前、ユーザーが Okta からのプッシュ通知を確認するために待機する時間 (秒単位)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
OktaMfaWaitTime	okta_mfa_wait_time (廃止)	オプションです。	60

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PreferredRole	preferred_role (廃止)	オプションです。	なし

ロールセッションの期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
RoleSessionDuration	時間 (廃止)	オプションです。	3600

Lake Formation 有効

AssumeRoleWithSAML AWS STS API アクションの代わりに、一時的な IAM 認証情報を取得するために AssumeDecoratedRoleWithSAML Lake Formation API アクションを使用するかどうかを指定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LakeFormationEnabled	なし	オプションです。	FALSE

Ping 認証情報

Ping フェデレーション ID プロバイダーを使用して Athena への認証を可能にする SAML ベースの認証メカニズム。この方法では、Athena と Ping フェデレーション の間にフェデレーションが既に設定されていることを前提としています。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `Ping` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`Ping`

ユーザー

Ping フェデレーションで認証に使用する Ping フェデレーションユーザーのメールアドレス。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ユーザー	UID (廃止)	必須	なし

パスワード

Ping フェデレーションユーザーのパスワード。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
パスワード	PWD (廃止)	必須	なし

PingHostName

Ping サーバーのアドレス。アドレスを確認するには、次の URL にアクセスして [SSO アプリケーションエンドポイント] フィールドを確認してください。

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PingHostName	IdP_Host (廃止)	必須	なし

PingPortNumber

IdP ホストへの接続に使用するポート番号。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PingPortNumber	IdP_Port (廃止)	必須	なし

PingPartnerSpId

サービスプロバイダのアドレス。サービスプロバイダのアドレスを確認するには、次の URL にアクセスして [SSO アプリケーションエンドポイント] フィールドを確認してください。

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PingPartnerSpId	Partner_SPID (廃止)	必須	なし

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PreferredRole	preferred_role (廃止)	オプションです。	なし

ロールセッションの期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
RoleSessionDuration	時間 (廃止)	オプションです。	3600

Lake Formation 有効

AssumeRoleWithSAML AWS STS API アクションの代わりに、一時的な IAM 認証情報を取得するために AssumeDecoratedRoleWithSAML Lake Formation API アクションを使用するかどうかを指定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LakeFormationEnabled	なし	オプションです。	FALSE

AD FS 認証情報

Microsoft Active Directory Federation Services (AD FS) を使用した Athena への認証を可能にする SAML ベースの認証メカニズムです。この手法は、ユーザーが Athena と AD FS 間のフェデレーションを既に設定していることを前提としています。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を ADFS に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	ADFS

ユーザー

AD FS での認証に使用する AD FS ユーザーの E メールアドレスです。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ユーザー	UID (廃止)	フォームベースの認証に必要です。Windows 統合認証の場合はオプションです。	なし

パスワード

AD FS ユーザーのパスワードです。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
パスワード	PWD (廃止)	フォームベースの認証に必要です。Windows 統合認証の場合はオプションです。	なし

ADFS ホスト名

AD FS サーバーのアドレスです。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AdfsHostName	IdP_Host (廃止)	必須	なし

AD FS ポート番号

AD FS サーバーへの接続に使用するポート番号です。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AdfsPortNumber	IdP_Port (廃止)	必須	なし

ADFS 依存パーティ

信頼できる依存パーティ。このパラメータを使用して、AD FS 依存パーティエンドポイント URL を上書きします。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AdfsRelyingParty	LoginToRP (廃止)	オプションです。	`urn:amazon:webservices`

ADFS WIA 有効

Boolean。このパラメータを使用して、AD FS で Windows 統合認証 (WIA) を有効にします。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AdfsWiaEnabled	none	オプションです。	FALSE

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールについては、「AWS Security Token Service API リファレンス」の「`AssumeRole`」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PreferredRole	preferred_role (廃止)	オプションです。	なし

ロールセッションの期間

ロールセッションの期間 (秒)。詳細については、AWS Security Token Service API リファレンスの「`AssumeRole`」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
RoleSessionDuration	時間 (廃止)	オプションです。	`3600`

Lake Formation 有効

一時的な IAM 認証情報を取得するために、AssumeRoleWithSAML AWS STS API アクションではなく AssumeDecoratedRoleWithSAML Lake Formation API アクションを使用するかどうかを指定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LakeFormationEnabled	none	オプションです。	FALSE

Browser Azure AD 認証情報

Browser Azure AD は、Azure AD ID プロバイダーと連携する SAML ベースの認証メカニズムであり、多要素認証をサポートします。標準の Azure AD 認証メカニズムとは異なり、このメカニズムには接続パラメータにユーザー名、パスワード、クライアントシークレットは不要です。標準の Azure AD 認証メカニズムと同様に、ブラウザ Azure AD もユーザーが Athena と Azure AD の間にフェデレーションを既に設定していることを前提としています。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `BrowserAzureAD` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`BrowserAzureAD`

Azure AD テナント ID

Azure AD アプリケーションのテナント ID。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AzureAdTenantId	tenant_id (廃止)	必須	なし

Azure AD クライアント ID

Azure AD アプリケーションのクライアント ID。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
AzureAdClientId	client_id (廃止)	必須	なし

ID プロバイダーの応答タイムアウト

ドライバーが Azure AD からの SAML レスポンスの待機を停止するまでの時間 (秒単位)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
IdpResponseTimeout	idp_response_timeout (廃止)	オプションです。	120

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PreferredRole	preferred_role (廃止)	オプションです。	なし

ロールセッションの期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
RoleSessionDuration	時間 (廃止)	オプションです。	3600

Lake Formation 有効

AssumeRoleWithSAML AWS STS API アクションの代わりに、一時的な IAM 認証情報を取得するために AssumeDecoratedRoleWithSAML Lake Formation API アクションを使用するかどうかを指定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LakeFormationEnabled	なし	オプションです。	FALSE

Browser SAML 認証情報

Browser SAML は、SAML ベースの ID プロバイダーと連携する汎用認証プラグインであり、多要素認証をサポートします。

認証情報プロバイダー

AWS へのリクエストの認証に使用される認証情報プロバイダー。このパラメータの値を `BrowserSaml` に設定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値	使用する値
CredentialsProvider	AWSCredentialsProviderClass (廃止)	必須	なし	`BrowserSaml`

シングルサインオンのログイン URL

SAML ベースの ID プロバイダー上のアプリケーションにおける Single sign-on URL。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
SsoLoginUrl	login_url (廃止)	必須	なし

リッスンポート

SAML レスポンスをリッスンするために使用されるポート番号。この値は、SAML ベースの ID プロバイダー (例えば、`http://localhost:7890/athena`) を設定した URL と一致する必要があります。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
ListenPort	listen_port (廃止)	オプションです。	7890

ID プロバイダーの応答タイムアウト

ドライバーが Azure AD からの SAML レスポンスの待機を停止するまでの時間 (秒単位)。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
IdpResponseTimeout	idp_response_timeout (廃止)	オプションです。	120

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
PreferredRole	preferred_role (廃止)	オプションです。	なし

ロールセッションの期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
RoleSessionDuration	時間 (廃止)	オプションです。	3600

Lake Formation 有効

AssumeRoleWithSAML AWS STS API アクションの代わりに、一時的な IAM 認証情報を取得するために AssumeDecoratedRoleWithSAML Lake Formation API アクションを使用するかどうかを指定します。

パラメータ名	エイリアス	パラメータタイプ	デフォルト値
LakeFormationEnabled	なし	オプションです。	FALSE

その他の JDBC 3.x 設定

次のセクションでは、JDBC 3.x ドライバーの一部の追加設定について説明します。

ネットワークタイムアウト

ドライバーが Athena に API 呼び出しを行うときに応答するまで待機する時間 (ミリ秒単位) この時間が経過すると、ドライバーはタイムアウト例外を発生させます。

ネットワークタイムアウトは接続パラメータとして設定できません。設定するには、JDBC `setNetworkTimeout` オブジェクトで `Connection` メソッドを呼び出します。この値は JDBC 接続のライフサイクル中に変更できます。このパラメータのデフォルト値は `infinity` です。

次の例では、ネットワークタイムアウトを 5,000 ミリ秒に設定しています。

```
...
AthenaDriver driver = new AthenaDriver();
Connection connection = driver.connect(url, connectionParameters);
connection.setNetworkTimeout(null, 5000);
...
```

クエリタイムアウト

クエリが送信された後、トライバーが Athena 上でクエリが完了するまで待機する時間 (秒単位) この時間が経過すると、ドライバーは送信されたクエリをキャンセルしようとし、タイムアウト例外を発生させます。

クエリタイムアウトは接続パラメータとして設定できません。設定するには、JDBC `setQueryTimeout` オブジェクトで `Statement` メソッドを呼び出します。この値は JDBC ステー

トメントのライフサイクル中に変更できます。このパラメータのデフォルト値は 0 (ゼロ) です。値が 0 の場合、クエリは完了するまで実行できることを意味します (Service Quotas が適用されます)。

次の例では、クエリタイムアウトを 5 秒に設定しています。

```
...
AthenaDriver driver = new AthenaDriver();
Connection connection = driver.connect(url, connectionParameters);
Statement statement = connection.createStatement();
statement.setQueryTimeout(5);
...
```

Amazon Athena ODBC 3.x リリースノート

これらのリリースノートには、Amazon Athena JDBC 3.x ドライバーの改良と修正の詳細が記載されています。

3.2.0

リリース日: 2024 年 4 月 26 日

改良点

- カタログ操作のパフォーマンス – ワイルドカード文字を使用しないカタログ操作のパフォーマンスが改善されました。
- 最小ポーリング間隔の変更 – 最小ポーリング間隔のデフォルトを変更し、ドライバーが Athena に対して行う API 呼び出し回数を減らしました。クエリ候補がすばやく検出されます。
- BI ツールの検出可能性 – ビジネスインテリジェンスツールのドライバーをより簡単に検出できるようになりました。
- データ型マッピング – Athena `binary`、`array`、および `struct` DDL データ型へのデータ型マッピングが改善されました。
- AWS SDK バージョン – ドライバーで使用されている AWS SDK バージョンが 2.25.34 に更新されました。

修正内容

- フェデレーションカタログのテーブルリスト – フェデレーションカタログが空のテーブルリストを返す問題を修正しました。

- getSchemas – JDBC DatabaseMetadata#getSchemas メソッドが、すべてのカタログからではなくデフォルトのカタログからのみデータベースを取得する問題を修正しました。
- getColumns – JDBC DatabaseMetadata#getColumns メソッドが null カタログ名で呼び出されたときに、null カタログが返される問題を修正しました。

3.1.0

リリース日: 2024 年 2 月 15 日

改良点

- Microsoft Active Directory Federation Services (AD FS) Windows 統合認証とフォームベース認証のサポートが追加されました。
- バージョン 2.x との下位互換性を保つため、awsathena JDBC サブプロトコルが受け入れられるようになりましたが、非推奨の警告が生成されます。代わりに athena JDBC サブプロトコルを使用してください。
- AwsDataCatalog がカタログパラメータのデフォルトとなり、default がデータベースパラメータのデフォルトとなりました。これらの変更により、現在のカタログとデータベースの正しい値 (NULL ではない) が返されるようになりました。
- JDBC 仕様に準拠し、IS_AUTOINCREMENT と IS_GENERATEDCOLUMN は NO ではなく空の文字列を返すようになりました。
- Athena int データ型は、other ではなく Athena integer と同じ JDBC 型にマップされるようになりました。
- Athena の列メタデータにオプションの precision および scale フィールドが含まれていない場合、ドライバーは ResultSet 列内の対応する値に 0 を返すようになりました。
- AWS SDK バージョンは、2.21.39 に更新されました。

修正内容

- Athena からのプレーンテキスト結果の列数が Athena 結果メタデータの列数と一致しない場合に例外が発生する GetQueryResultsStream の問題を修正しました。

3.0.0

リリース日: 2023 年 11 月 16 日

Athena JDBC 3.x ドライバーは、より優れたパフォーマンスと互換性を提供する新世代ドライバーです。JDBC 3.x ドライバーは、Amazon S3 からの直接的なクエリ結果の読み取りをサポートするので、大規模なクエリ結果を使用するアプリケーションのパフォーマンスが向上します。新しいドライバーではサードパーティーとの依存関係も少なくなっているため、BI ツールやカスタムアプリケーションとの統合が容易になります。

以前のバージョンの Athena JDBC 3.x ドライバー

最新バージョンの JDBC 3.x ドライバーを使用することを強くお勧めします。ドライバーの最新バージョンには、最新の改良と修正が含まれています。ご使用のアプリケーションが最新バージョンと互換性がない場合にのみ、古いバージョンを使用してください。

JDBC ドライバー uber jar

次のダウンロードは、ドライバーおよびそのすべての依存関係を同じ `.jar` ファイルにパッケージ化します。このダウンロードは、サードパーティーの SQL クライアントでよく使用されます。

- 3.1.0 uber jar
- 3.0.0 uber jar

JDBC ドライバーの lean jar

次のダウンロードには、ドライバー用の lean `.jar` およびドライバーの依存関係用の個別 `.jar` ファイルを含む `.zip` ファイルです。このダウンロードは、ドライバーが使用する依存関係と競合する依存関係を持つカスタムアプリケーションによく使用されます。このダウンロードは、ドライバーの依存関係のどっちを lean jar に含め、カスタムアプリケーションに既に 1 つ以上含まれている場合はどの依存関係を除外するかを選択する場合に便利です。

- 3.1.0 lean jar
- 3.0.0 lean jar

Athena JDBC 2.x ドライバー

JDBC 接続を使用して、SQL Workbench などのビジネスインテリジェンスツールとその他アプリケーションに Athena を接続することができます。これを実施するには、このページにある Amazon S3 のリンクを使用し、Athena JDBC 2.x ドライバーをダウンロード、インストール、設定を行います。JDBC 接続 URL の構築については、ダウンロード版JDBC ドライバーインストールおよび設定ガイドを参照してください。アクセス許可については、「JDBC および ODBC 接続を

介したアクセス」を参照してください。JDBC ドライバーに関するフィードバックは、athena-feedback@amazon.com に電子メールでお問い合わせください。バージョン 2.0.24 以降では、2 つ (AWS SDK を含むものと含まないもの) のバージョンのドライバを使用できます。

> **Important**
>
> JDBC ドライバーを使用するときは、次の要件に注意してください。
>
> - オープン状態のポート 444 – Athena がクエリ結果のストリーミングに使用するポート 444 には、アウトバウンドトラフィックに対して開放されている状態を維持します。PrivateLink エンドポイントを使用して Athena に接続するときは、PrivateLink エンドポイントにアタッチされているセキュリティグループが、ポート 444 上のインバウンドトラフィックに対して開放されていることを確認してください。ポート 444 がブロックされている場合は、「[Simba][AthenaJDBC](100123) An error has occurred.」というエラーメッセージが表示されます。列の初期化中の例外。
> - athena:GetQueryResultsStream ポリシー – JDBC ドライバーを使用する IAM プリンシパルに `athena:GetQueryResultsStream` ポリシーアクションを追加します。このポリシーアクションが API で直接公開されることはありません。ストリーミング結果のサポートの一環として、ODBC および JDBC ドライバーでのみ使用されます。ポリシーの例については、「AWS 管理ポリシー: AWSQuicksightAthenaAccess」を参照してください。
> - 複数のデータカタログに対応するJDBCドライバを使用する場合 - 複数のデータカタログのために Athena で JDBC ドライバーを使用するには (外部の Hive メタストアまたはフェデレーティッドクエリを使用する場合など)、JDBC 接続文字列に `MetadataRetrievalMethod=ProxyAPI` を含めてください。
> - 4.1 ドライバー - 2023 年以降、JDBC バージョン 4.1 のドライバーサポートを終了します。今後、アップデートの予定はありません。JDBC 4.1 ドライバーを使用している場合は、4.2 ドライバーへの移行を強くお勧めします。

AWS SDK を使用する JDBC 2.x ドライバー

JDBC ドライバーバージョン 2.1.5 は、JDBC API 4.2 データ規格に準拠しており、JDK 8.0 またはそれ以降が必要です。ご使用の JRE (Java Runtime Environment) のバージョンを確認する方法については、Java の ドキュメントを参照してください。

JDBC 4.2 ドライバーの `.jar` ファイルをダウンロードするには、次のリンクを使用します。

- AthenaJDBC42-2.1.5.1000.jar

次の .zip ファイルのダウンロードには、JDBC 4.2 用の .jar ファイル、および AWS SDK とそれに付随するドキュメント、リリースノート、ライセンス、契約書が含まれています。

- SimbaAthenaJDBC-2.1.5.1000.zip

AWS SDK を使用しない JDBC 2.x ドライバー

JDBC ドライバーバージョン 2.1.5 は、JDBC API 4.2 データ規格に準拠しており、JDK 8.0 またはそれ以降が必要です。ご使用の JRE (Java Runtime Environment) のバージョンを確認する方法については、Java の ドキュメントを参照してください。

AWS SDK なしで JDBC 4.2 ドライバー .jar ファイルをダウンロードするには、次のリンクを使用してください。

- AthenaJDBC42-2.1.5.1001.jar

次の .zip ファイルのダウンロードには、JDBC 4.2 用の .jar ファイル、および付属ドキュメント、リリースノート、ライセンス、契約書が含まれています。これには、AWS SDK は含まれません。

- SimbaAthenaJDBC-2.1.5.1001.zip

JDBC 2.x ドライバーのリリースノート、ライセンス契約、通知

必要なバージョンをダウンロードしたら、リリースノートをお読みになり、ライセンス契約および表示を確認してください。

- リリースノート
- ライセンス契約
- 通知
- サードパーティーライセンス

JDBC 2.x ドライバーのドキュメント

ドライバー用の次のドキュメントをダウンロードします。

- JDBC ドライバーのインストールおよび設定ガイド。このガイドを使用してドライバーをインストールし、設定します。

- JDBC ドライバー移行ガイド。このガイドを使用して以前のバージョンから現在のバージョンに移行します。

ODBC を使用した Amazon Athena への接続

Amazon Athena には、バージョン 1.x と 2.x の 2 つの ODBC ドライバーが用意されています。Athena ODBC 2.x ドライバーは、Linux、macOS ARM、macOS Intel、および Windows 64 ビットシステムをサポートする新しいオプションです。Athena 2.x ドライバーは 1.x ODBC ドライバーがサポートするすべての認証プラグインをサポートしており、ほとんどすべての接続パラメータには下位互換性があります。

- ODBC 2.x ドライバーをダウンロードするには、「Amazon Athena ODBC 2.x」を参照してください。
- ODBC 1.x ドライバーをダウンロードするには、「Athena ODBC 1.x ドライバー」を参照してください。

トピック

- Amazon Athena ODBC 2.x
- Athena ODBC 1.x ドライバー
- Amazon Athena Power BI コネクタの使用

Amazon Athena ODBC 2.x

ODBC 接続を使用すると、多くのサードパーティ SQL クライアントツールおよびアプリケーションから Amazon Athena に接続できます。ODBC 接続はクライアントコンピュータで設定します。

考慮事項と制約事項

- Athena ODBC 1.x ドライバーから Athena 2.x ODBC ドライバーへと移行するための情報については、「ODBC 2.x ドライバーへの移行」を参照してください。
- CSE_KMS 暗号化オプションで S3 フェッチャーを使用する場合、Amazon S3 クライアントは Amazon S3 バケットに保存される結果を復号化できません。回避策として、Athena ストリーミング API オプションを使用して結果セットを取得してください。

ODBC 2.x ドライバーのダウンロード

Amazon Athena 2.x ODBC ドライバーをダウンロードするには、このページのリンクにアクセスしてください。

> **Important**
>
> ODBC 2.x ドライバーを使用するときは、次の要件に注意してください。
>
> - オープン状態のポート 444 – Athena がクエリ結果のストリーミングに使用するポート 444 には、アウトバウンドトラフィックに対して開放されている状態を維持します。PrivateLink エンドポイントを使用して Athena に接続するときは、PrivateLink エンドポイントにアタッチされているセキュリティグループが、ポート 444 上のインバウンドトラフィックに対して開放されていることを確認してください。
> - athena:GetQueryResultsStream ポリシー – ODBC ドライバーを使用する IAM プリンシパルに `athena:GetQueryResultsStream` ポリシーアクションを追加します。このポリシーアクションが API で直接公開されることはありません。ストリーミング結果のサポートの一環として、ODBC および JDBC ドライバーでのみ使用されます。ポリシーの例については、「AWS 管理ポリシー: AWSQuicksightAthenaAccess」を参照してください。

Linux

ドライバーのバージョン	ダウンロードリンク
Linux 64 ビット版対応 ODBC 2.0.3.0	Linux 64 ビット ODBC ドライバー 2.0.3.0Linux 64 ビット ODBC ドライバー 2.0.3.0

MacOS (ARM)

ドライバーのバージョン	ダウンロードリンク
MacOS 64 ビット版 (ARM) 対応 ODBC 2.0.3.0	macOS 64 ビット ODBC ドライバー 2.0.3.0 (ARM)

macOS (Intel)

ドライバーのバージョン	ダウンロードリンク
MacOS 64 ビット (Intel) 対応 ODBC 2.0.3.0	macOS 64 ビット ODBC ドライバー 2.0.3.0 (Intel)

Windows

ドライバーのバージョン	ダウンロードリンク
Windows 64 ビット用 ODBC 2.0.3.0	Windows 64 ビット ODBC ドライバー 2.0.3.0Windows 64 ビット ODBC ドライバー 2.0.3.0

トピック

- ODBC 2.x ドライバーの開始方法
- Athena ODBC 2.x 接続パラメータ
- ODBC 2.x ドライバーへの移行
- ODBC 2.x ドライバーのトラブルシューティング
- Amazon Athena ODBC 2.x リリースノート

ODBC 2.x ドライバーの開始方法

Amazon Athena ODBC 2.x ドライバーを開始するには、このセクションの情報を使用してください。ドライバーは、Windows、Linux、および macOS オペレーティングシステムでサポートされています。

トピック

- Windows
- Linux
- macOS

Windows

Amazon Athena へのアクセスに Windows クライアントコンピューターを使用する場合は、Amazon Athena ODBC ドライバーが必要です。

Windows システム要件

ウェブブラウザを使用せずに Amazon Athena データベースに直接アクセスするクライアントコンピュータに、Amazon Athena ODBC ドライバーをインストールします。

使用する Windows システムは、以下の要件を満たしている必要があります。

- 管理者権限がある
- 次のいずれかの OS。
  - Windows 11、10 または 8.1。
  - Windows Server 2019、2016、または 2012。
- 最低でも 100 MB の空きディスク容量。
- 64 ビット Windows 用の Visual Studio の Microsoft Visual C++ 再頒布可能パッケージがインストールされている。

Amazon Athena ODBC ドライバーをインストールする

Windows 用 Amazon Athena ODBC ドライバーをダウンロードしてインストールする方法

1. `AmazonAthenaODBC-2.x.x.x.msi` インストールファイルをダウンロードします。
2. インストールファイルを起動し、[次へ] を選択します。
3. 使用許諾契約書の条項に同意する場合は、チェックボックス > [次へ] の順に選択します。
4. インストール場所を変更するには、[参照] を選択して目的のフォルダを参照し、[OK] を選択します。
5. インストール場所を確定するには、[次へ] を選択します。
6. [Install] (インストール) を選択します。
7. インストールが完了したら、[完了] を選択します。

ドライバー設定オプションの設定方法

Windows の Amazon Athena ODBC ドライバーの動作を制御するには、次の方法でドライバー設定オプションを指定できます。

- データソース名 (DSN) を設定する場合は [ODBC データソース管理者] プログラム内で設定。
- 次の場所に Windows レジストリキーを追加または変更する。

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\YOUR_DSN_NAME
```

- プログラムで接続するときに、接続文字列にドライバーオプションを設定する。

Windows 上でのデータソース名を設定する

ODBC ドライバーをダウンロードしてインストールした後、クライアントコンピュータか Amazon EC2 インスタンスにデータソース名 (DSN) エントリを追加する必要があります。SQL クライアントツールは、このデータソースを使用して Amazon Athena データベースに接続し、クエリします。

システム DSN エントリを作成するには

1. Windows の [スタート] メニューから [ODBC データソース (64 ビット)] を右クリックし、[その他] > [管理者として実行] の順に選択します。
2. [ODBC データソース管理者] で、[ドライバー] タブを選択します。
3. [名前] 列に [Amazon Athena ODBC (x64)] が表示されていることを確認します。
4. 次のいずれかを行います。
   - コンピュータ上のすべてのユーザーに対してドライバーを設定する場合は、[システム DSN] タブを選択します。別のアカウントを使用してデータをロードするアプリケーションでは、別のアカウントのユーザー DSN を検出できない場合があるため、システム DSN 構成オプションを使用することをお勧めします。

     > **Note**
     > [システム DSN] オプションを使用するには、管理者権限が必要です。

   - ユーザーアカウントのみに対してドライバーを設定するには、[ユーザー DSN] タブを選択します。
5. 追加] を選択します。[新しいデータソースを作成] ダイアログボックスが開きます。
6. [Amazon Athena ODBC (x64)] > [完了] の順に選択します。
7. [Amazon Athena ODBC 設定] ダイアログボックスに、次の情報を入力します。これらのオプションの詳細な情報については、主な ODBC 2.x 接続パラメータを参照してください。

- [データソース名] に、データソースを識別するために使用する名前を入力します。
- [説明] には、データソースをすばやく識別できるような説明を入力します。
- [リージョン] には、Athena を使用する AWS リージョン の名前を入力します (例: **us-west-1**)。
- [カタログ] には、Amazon Athena カタログの名前を入力します。デフォルトは [AWSDataCatalog] で、AWS Glue によって使用されます。
- [データベース] には、Amazon Athena データベースの名前を入力します。デフォルトは [デフォルト] です。
- [ワークグループ] には、Amazon Athena ワークグループの名前を入力します。デフォルトは [プライマリ] です。
- [S3 出力場所] に、クエリ結果が保存されることになる Amazon S3 内の口ケーション (例: **s3://DOC-EXAMPLE-BUCKET/**) を入力します。
- (オプション) [暗号化オプション] で、暗号化オプションを選択します。デフォルト: NOT_SET。
- (オプション) [KMS キー] では、必要に応じて暗号 KMS キーを選択します。

8. IAM 認証の設定オプションを指定するには、[認証オプション] を選択します。
9. 次の情報を入力します。

   - [認証タイプ] で [IAM 認証情報] を選択します。これがデフォルトです。使用できる認証タイプの詳細については、「認証オプション」を参照してください。
   - [ユーザー名] には、ユーザー名を入力します。
   - [パスワード] には、パスワードを入力します。
   - [セッショントークン] には、一時的な AWS 認証情報を使用する場合はセッショントークンを入力します。一時的な認証情報の詳細については、「IAM ユーザーガイド」の「AWS リソースを使用した一時的な認証情報の使用」を参照してください。

10. [OK] を選択します。
11. [Amazon Athena ODBC 設定] ダイアログボックスの下部で、[テスト] を選択します。クライアントコンピュータが Amazon Athena に正常に接続すると、[接続テスト] ボックスに [接続に成功しました] と表示されます。成功しなかった場合は、ダイアログボックスに関連するエラー情報と共に [接続に失敗しました] と表示されます。
12. [OK] を選択して、接続テストを閉じます。作成したデータソースが、[データソース名] リストに表示されます。

Windows で DSN なし接続を使用する

DSN を使用しない接続を使用すれば、データソース名 (DSN) なしでデータベースに接続できます。次の例は、Amazon Athena に接続する Amazon Athena ODBC (x64) ODBC ドライバーの接続文字列を示しています。

```
DRIVER={Amazon Athena ODBC (x64)};Catalog=AwsDataCatalog;AwsRegion=us-
west-1;Schema=test_schema;S3OutputLocation=
s3://DOC-EXAMPLE-BUCKET/;AuthenticationType=IAM Credentials;UID=YOUR_UID;PWD=YOUR_PWD;
```

Linux

Amazon Athena へのアクセスに Linux クライアントコンピューターを使用する場合は、Amazon Athena ODBC ドライバーが必要です。

Linux システム要件

ドライバーをインストールする Linux コンピューターは、それぞれ以下の最小要件を満たしている必要があります。

- ルートアクセス権がある。
- 以下の Linux ディストリビューションのいずれかを使用している。
  - Red Hat Enterprise Linux (RHEL) 7 または 8
  - CentOS 7 または 8。
- 100 MB のディスク空き容量がある。
- unixODBC のバージョン 2.3.1 以降を使用している。
- GNU C ライブラリ (glibc) のバージョン 2.26 以降を使用している。

Linux への ODBC データコネクタのインストール

Linux オペレーティングシステムに Amazon Athena ODBC ドライバーをインストールするには、以下の手順を実行します。

Amazon Athena ODBC ドライバーを Linux にインストールする

1. 次のいずれかのコマンドを入力します。

   ```
   sudo rpm -Uvh AmazonAthenaODBC-2.X.Y.Z.rpm
   ```

または

```
sudo yum --nogpgcheck localinstall AmazonAthenaODBC-2.X.Y.Z.rpm
```

2. インストールが完了したら、以下のコマンドのいずれかを入力して、ドライバーがインストールされていることを確認します。

   - ```
     yum list | grep amazon-athena-odbc-driver
     ```

     出力:

     ```
     amazon-athena-odbc-driver.x86_64 2.0.2.1-1.amzn2int installed
     ```

   - ```
     rpm -qa | grep amazon
     ```

     出力:

     ```
     amazon-athena-odbc-driver-2.0.2.1-1.amzn2int.x86_64
     ```

Linux でのデータソース名の設定

ドライバーがインストールされると、以下の場所に .odbc.ini および .odbcinst.ini のサンプルファイルを見つけることができます。

- /opt/athena/odbc/ini/.

この場所にある .ini ファイルを、Amazon Athena ODBC ドライバーとデータソース名 (DSN) の設定例として使用してください。

> **Note**
>
> デフォルトで、ODBC ドライバーマネージャーはホームディレクトリにある隠し設定ファイル .odbc.ini と .odbcinst.ini を使用します。

unixODBC を使用して .odbc.ini と .odbcinst.ini ファイルへのパスを指定するには、次の手順を実行します。

unixODBC を使用して ODBC **.ini** ファイルの場所を指定する

1. 以下の例にあるように、odbc.ini ファイルのフルパスとファイル名に ODBCINI 設定します。

```
export ODBCINI=/opt/athena/odbc/ini/odbc.ini
```

2. 以下の例にあるように、odbcinst.ini のファイルが含まれるディレクトリのフルパスに ODBCSYSINI を設定します。

```
export ODBCSYSINI=/opt/athena/odbc/ini
```

3. 以下のコマンドを入力して、unixODBC ドライバーマネージャーと正しい odbc*.ini ファイルを使用していることを確認します。

```
username % odbcinst -j
```

サンプル出力

```
unixODBC 2.3.1
DRIVERS............: /opt/athena/odbc/ini/odbcinst.ini
SYSTEM DATA SOURCES: /opt/athena/odbc/ini/odbc.ini
FILE DATA SOURCES..: /opt/athena/odbc/ini/ODBCDataSources
USER DATA SOURCES..: /opt/athena/odbc/ini/odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

4. データソース名 (DSN) を使用してデータストアに接続する場合は、odbc.ini ファイルを設定してデータソース名 (DSN) を定義します。以下の例にあるように、odbc.ini ファイルのプロパティを設定して、データストアの接続情報を指定する DSN を作成します。

```
[ODBC Data Sources]
athena_odbc_test=Amazon Athena ODBC (x64)

[ATHENA_WIDE_SETTINGS]  # Special DSN-name to signal driver about logging
 configuration.
LogLevel=0              # To enable ODBC driver logs, set this to 1.
UseAwsLogger=0          # To enable AWS-SDK logs, set this to 1.
LogPath=/opt/athena/odbc/logs/ # Path to store the log files. Permissions to the
 location are required.
```

```
[athena_odbc_test]
Driver=/opt/athena/odbc/lib/libathena-odbc.so
AwsRegion=us-west-1
Workgroup=primary
Catalog=AwsDataCatalog
Schema=default
AuthenticationType=IAM Credentials
UID=
PWD=
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/
```

5. 以下の例にあるように、odbcinst.ini ファイルを設定します。

```
[ODBC Drivers]
Amazon Athena ODBC (x64)=Installed

[Amazon Athena ODBC (x64)]
Driver=/opt/athena/odbc/lib/libathena-odbc.so
Setup=/opt/athena/odbc/lib/libathena-odbc.so
```

6. Amazon Athena ODBC ドライバーをインストールして設定したら、以下の例にあるように、unixODBC isql コマンドラインツールを使用して接続を確認します。

```
username % isql -v "athena_odbc_test"
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

macOS

Amazon Athena へのアクセスに macOS クライアントコンピューターを使用する場合は、Amazon Athena ODBC ドライバーが必要です。

macOS システム要件

ドライバーをインストールする macOS コンピューターは、それぞれ以下の最小要件を満たしている必要があります。

- macOS バージョン 14 以降を使用している。
- 100 MB のディスク空き容量がある。
- iODBC のバージョン 3.52.16 以降を使用している。

macOS への ODBC データコネクタのインストール

macOS オペレーティングシステム用の Amazon Athena ODBC ドライバーをダウンロードしてインストールするには、以下の手順を実行します。

macOS 用の Amazon Athena ODBC ドライバーをダウンロードしてインストールする

1. .pkg パッケージファイルをダウンロードします。
2. .pkg ファイルをダブルクリックします。
3. ウィザードの手順に従ってドライバーをインストールします。
4. [ライセンス契約] ページで [続行] を押してから、[同意する] を選択します。
5. [Install] (インストール) を選択します。
6. インストールが完了したら、[完了] を選択します。
7. 以下のコマンドを入力して、ドライバーがインストールされていることを確認します。

```
> pkgutil --pkgs | grep athenaodbc
```

出力は、システムに応じて以下のいずれかのようになります。

```
com.amazon.athenaodbc-x86_64.Config
com.amazon.athenaodbc-x86_64.Driver
```

または

```
com.amazon.athenaodbc-arm64.Config
com.amazon.athenaodbc-arm64.Driver
```

macOS でのデータソース名の設定

ドライバーがインストールされると、以下の場所に .odbc.ini および .odbcinst.ini のサンプルファイルを見つけることができます。

- Intel プロセッサコンピューター: /opt/athena/odbc/x86_64/ini/
- ARM プロセッサコンピューター: /opt/athena/odbc/arm64/ini/

この場所にある .ini ファイルを、Amazon Athena ODBC ドライバーとデータソース名 (DSN) の設定例として使用してください。

> **Note**
>
> デフォルトで、ODBC ドライバーマネージャーはホームディレクトリにある隠し設定ファイル .odbc.ini と .odbcinst.ini を使用します。

iODBC ドライバーマネージャーを使用して .odbc.ini と .odbcinst.ini ファイルへのパスを指定するには、次の手順を実行します。

iODBC ドライバーマネージャーを使用して ODBC .ini ファイルの場所を指定する

1. odbc.ini のファイルのフルパスとファイル名に ODBCINI を設定します。
   - Intel プロセッサ搭載の macOS コンピューターでは、以下の構文を使用します。

     ```
     export ODBCINI=/opt/athena/odbc/x86_64/ini/odbc.ini
     ```

   - ARM プロセッサ搭載の macOS コンピューターでは、以下の構文を使用します。

     ```
     export ODBCINI=/opt/athena/odbc/arm64/ini/odbc.ini
     ```

2. odbcinst.ini のファイルのフルパスとファイル名に ODBCSYSINI を設定します。
   - Intel プロセッサ搭載の macOS コンピューターでは、以下の構文を使用します。

     ```
     export ODBCSYSINI=/opt/athena/odbc/x86_64/ini/odbcinst.ini
     ```

   - ARM プロセッサ搭載の macOS コンピューターでは、以下の構文を使用します。

```
export ODBCSYSINI=/opt/athena/odbc/arm64/ini/odbcinst.ini
```

3. データソース名 (DSN) を使用してデータストアに接続する場合は、odbc.ini ファイルを設定してデータソース名 (DSN) を定義します。以下の例にあるように、odbc.ini ファイルのプロパティを設定して、データストアの接続情報を指定する DSN を作成します。

```
[ODBC Data Sources]
athena_odbc_test=Amazon Athena ODBC (x64)

[ATHENA_WIDE_SETTINGS] # Special DSN-name to signal driver about logging
 configuration.
LogLevel=0             # set to 1 to enable ODBC driver logs
UseAwsLogger=0         # set to 1 to enable AWS-SDK logs
LogPath=/opt/athena/odbc/logs/ # Path to store the log files. Permissions to the
 location are required.

[athena_odbc_test]
Description=Amazon Athena ODBC (x64)
# For ARM:
Driver=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
# For Intel:
# Driver=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
AwsRegion=us-west-1
Workgroup=primary
Catalog=AwsDataCatalog
Schema=default
AuthenticationType=IAM Credentials
UID=
PWD=
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/
```

4. 以下の例にあるように、odbcinst.ini ファイルを設定します。

```
[ODBC Drivers]
Amazon Athena ODBC (x64)=Installed

[Amazon Athena ODBC (x64)]
# For ARM:
Driver=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
Setup=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
# For Intel:
# Driver=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
```

```
# Setup=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
```

5. Amazon Athena ODBC ドライバーをインストールして設定したら、以下の例にあるように、iodbctest コマンドラインツールを使用して接続を確認します。

```
username@ % iodbctest
iODBC Demonstration program
This program shows an interactive SQL processor
Driver Manager: 03.52.1623.0502

Enter ODBC connect string (? shows list): ?

DSN                             | Driver
------------------------------------------------------------------------------
athena_odbc_test                | Amazon Athena ODBC (x64)

Enter ODBC connect string (? shows list): DSN=athena_odbc_test;
Driver: 2.0.2.1 (Amazon Athena ODBC Driver)

SQL>
```

Athena ODBC 2.x 接続パラメータ

[Amazon Athena ODBC 設定] ダイアログボックスのオプションには、[認証オプション]、[詳細オプション]、[ロギングオプション]、[エンドポイントオーバーライド]、[プロキシオプション] が含まれます。各項目の詳細については、対応するリンク先をご覧ください。

- 主な ODBC 2.x 接続パラメータ
- 認証オプション
- 詳細オプション
- ログ記録オプション
- エンドポイントオーバーライド
- プロキシオプション

主な ODBC 2.x 接続パラメータ

以下のセクションでは、主な各接続パラメータについて説明します。

データソース名

データソースの名前を指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
DSN	DSN を使用しない接続タイプの場合のオプション	none	DSN=AmazonAthenaOdbcUsWest1;

説明

データソースの説明が含まれます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
説明	オプションです。	none	Description=Connection to Amazon Athena us-west-1;

カタログ

データカタログ名を指定します。詳細については、「Amazon Athena API リファレンス」の「DataCatalog」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
カタログ	オプションです。	AwsDataCatalog	Catalog=AwsDataCatalog;

リージョン

AWS リージョン を指定します。AWS リージョン の詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AwsRegion	必須	none	AwsRegion =us-west-1;

データベース

データベース名を指定します。詳細については、「Amazon Athena API リファレンス」の「Database」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
Schema	オプションです。	default	Schema=default;

Workgroup

ワークグループ名を指定します。詳細については、「Amazon Athena API リファレンス」の「ワークグループ」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
Workgroup	オプションです。	primary	Workgroup =primary;

出力場所

クエリ結果が保存される Amazon S3 内の場所を指定します。出力場所の詳細については、「Amazon Athena API リファレンス」の「ResultConfiguration」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
S3OutputLocation	必須	none	S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/;

暗号化オプション

[ダイアログパラメータ名]: 暗号化オプション

暗号化オプションを指定します。暗号化オプションの詳細については、「Amazon Athena API リファレンス」の「EncryptionConfiguration」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	使用できる値	接続文字列の例
S3OutputEncOption	オプションです。	none	NOT_SET, SSE_S3, SSE_KMS, CSE_KMS	S3OutputEncOption=SSE_S3;

KMS キー

暗号化用の KMS キーを指定します。KMS キー用の暗号化設定に関する詳細については、「Amazon Athena API リファレンス」の「EncryptionConfiguration」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
S3OutputEncKMSKey	オプションです。	none	S3OutputEncKMSKey=your_key;

接続テスト

ODBC データソース管理者には、Amazon Athena への ODBC 2.x 接続をテストするために使用できる [テスト] オプションが用意されています。この手順については、「Windows 上でのデータソース名を設定する」を参照してください。接続をテストすると、ODBC ドライバーは GetWorkGroup Athena API アクションをコールします。コールでは、指定した認証タイプと対応する認証情報プロバイダを使用して、認証情報を取得します。ODBC 2.x ドライバーを使用する場合、接続テストは無料です。このテストでは、Amazon S3 バケットにクエリ結果は生成されません。

認証オプション

以下の認証タイプを使用して Amazon Athena に接続できます。どのタイプでも、接続文字列名は AuthenticationType、パラメータタイプは Required、デフォルト値は IAM Credentials となります。各認証タイプのパラメータについては、それぞれのリンクを参照してください。一般的な認証パラメータについては、「一般的な認証パラメータ」を参照してください。

[Authentication type] (認証タイプ)	接続文字列の例
IAM 認証情報	`AuthenticationType=IAM Credentials;`
IAM プロフィール	`AuthenticationType=IAM Profile;`
AD FS	`AuthenticationType=ADFS;`
Azure AD	`AuthenticationType=AzureAD;`
Browser Azure AD	`AuthenticationType=BrowserAzureAD;`
Browser SAML	`AuthenticationType=BrowserSAML;`
Browser SSO OIDC	`AuthenticationType=BrowserSSOOIDC;`
デフォルト認証情報	`AuthenticationType=Default Credentials;`
外部認証情報	`AuthenticationType=External Credentials;`
インスタンスプロファイル	`AuthenticationType=Instance Profile;`
JWT	`AuthenticationType=JWT;`
Okta	`AuthenticationType=Okta;`
Ping	`AuthenticationType=Ping;`

IAM 認証情報

IAM 認証情報により、このセクションで説明する接続文字列パラメータを使用して ODBC ドライバーで Amazon Athena に接続できます。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=IAM Credentials;`

ユーザー ID

AWS アクセスキー ID。アクセスキーの取得に関する詳細については、「IAM ユーザーガイド」の「AWS セキュリティ認証情報」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UID	必須	none	`UID=AKIAIOSFODNN7EXAMPLE;`

パスワード

AWS シークレットキーの ID。アクセスキーの取得に関する詳細については、「IAM ユーザーガイド」の「AWS セキュリティ認証情報」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
PWD	必須	none	`PWD=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKE;`

セッショントークン

一時的な AWS 認証情報を使用している場合は、セッショントークンを指定する必要があります。一時的なセキュリティ認証情報の詳細については、「IAM ユーザーガイド」の「IAM の一時的なセキュリティ認証情報」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
[SessionToken]	オプションです。	none	`SessionToken=AQoDYXdzEJr...<remainder of session token>;`

IAM プロフィール

ODBC ドライバーを使用して、Amazon Athena に接続するように名前付きプロファイルを設定できます。ホストする Amazon EC2 インスタンスプロファイルで利用可能な認証情報を使用するには、`credential_source` パラメータを `Ec2InstanceMetadata` に設定します。名前付きプロファイルでカスタム認証情報プロバイダを使用する場合は、プロファイル設定で `plugin_name` パラメータの値を指定します。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=IAM Profile;`

AWS プロファイル

ODBC 接続に使用するプロファイル名。プロファイルの詳細については、「AWS Command Line Interface ユーザーガイド」の「名前付きプロファイルを使用する」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AWS プロファイル	必須	none	`AWSProfile=default;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。優先ロールパラメータは、カスタム認証情報プロバイダがプロファイル設定の `plugin_name` パラメータで指定されている場合に使用されます。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。セッション期間の詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。セッション期間パラメータは、カスタム認証情報プロバイダがプロファイル設定の `plugin_name` パラメータで指定されている場合に使用されます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	`900`	`duration=900;`

プラグイン名

名前付きプロファイルで使用される、カスタム認証情報プロバイダの名前を指定します。このパラメータには、ODBC データソース管理者の [認証タイプ] フィールドの値と同じ値を指定できますが、`AWSProfile` 設定でのみ使用されます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
plugin_name	オプションです。	none	`plugin_name=AzureAD;`

AD FS

AD FS は、Active Directory フェデレーションサービス (AD FS) の ID プロバイダと連携する SAML ベースの認証プラグインです。このプラグインは、統合 Windows 認証とフォームベース認証をサポートしています。統合 Windows 認証を使用する場合は、ユーザー名とパスワードを省略できます。AD FS と Athena の設定に関する詳細については、「ODBC クライアントを使用して Microsoft AD FS ユーザーに Amazon Athena へのフェデレーテッドアクセスを設定する」を参照してください。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=ADFS;`

ユーザー ID

AD FS サーバーに接続するためのユーザー名。統合 Windows 認証を使用する場合は、ユーザー名を省略できます。AD FS の設定でユーザー名が必要な場合は、接続パラメータでユーザー名を指定する必要があります。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UID	Windows 統合認証の場合はオプション	none	`UID=domain\username;`

パスワード

AD FS サーバーに接続するためのパスワード。[ユーザー名] フィールドと同様に、統合 Windows 認証を使用する場合はユーザー名を省略できます。AD FS の設定でパスワードが必要な場合は、接続パラメータでパスワードを指定する必要があります。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
PWD	Windows 統合認証の場合はオプション	none	`PWD=password_3EXAMPLE;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。SAML アサーションに複数の役割がある場合は、このパラメータを指定して引き受ける役割を選択できます。このロールは SAML アサーションに含まれている必要があります。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。セッション期間の詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	`900`	`duration=900;`

IdP Host

AD FS サービスホストの名前。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_host	必要	none	`idp_host=<server-name>.<company.com>;`

IdP ポート

AD FS ホストへの接続に使用するポート。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_port	必須	none	`idp_port=443;`

LoginToRP

信頼できる依存パーティ。このパラメータを使用して、AD FS 依存パーティエンドポイント URL を上書きします。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
LoginToRP	オプションです。	`urn:amazon:webservices`	`LoginToRP=trustedparty;`

Azure AD

Azure AD は Azure AD ID プロバイダと連携する SAML ベースの認証プラグインです。このプラグインは、多要素認証 (MFA) をサポートしません。MFA サポートが必要な場合は、代わりに `BrowserAzureAD` プラグインを使用することを検討してください。

認証タイプ

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=AzureAD;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	`none`	`preferred_role=arn:aws:iam::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	`900`	`duration=900;`

テナント ID

アプリケーションのテナント ID を指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_tenant	必須	none	`idp_tenant=123zz112z-z12d-1z1f-11zz-f111aa111234;`

クライアント ID

アプリケーションのクライアント ID を指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
client_id	必須	none	`client_id=9178ac27-a1bc-1a2b-1a2b-a123abcd1234;`

クライアントシークレット

クライアントのシークレットを指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
client_secret	必須	none	`client_secret=zG12q~.xzG1xxxZ1wX1.~ZzXXX1XxkHZizeT1zzZ;`

Browser Azure AD

Browser Azure AD は Azure AD ID プロバイダと連携する SAML ベースの認証プラグインで、多要素認証をサポートします。標準の Azure AD プラグインとは異なり、このプラグインでは接続パラメータにユーザー名、パスワード、クライアントシークレットは不要です。

認証タイプ

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=BrowserAzureAD;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。SAML アサーションに複数の役割がある場合は、このパラメータを指定して引き受ける役割を選択できます。指定されたロールが SAML アサーションに含まれている必要があります。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。セッション期間の詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	`900`	`duration=900;`

テナント ID

アプリケーションのテナント ID を指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_tenant	必須	none	`idp_tenant=123zz112z-z12d-1z1f-11zz-f111aa111234;`

クライアント ID

アプリケーションのクライアント ID を指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
client_id	必須	none	`client_id=9178ac27-a1bc-1a2b-1a2b-a123abcd1234;`

タイムアウト

プラグインが Azure AD からの SAML レスポンスが得られるまで待機するのを停止するまでの時間 (秒単位)。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
timeout	オプションです。	`120`	`timeout=90;`

Azure ファイルキャッシュを有効にする

一時的な認証情報キャッシュを有効にします。この接続パラメータにより、一時的な認証情報を複数のプロセス間でキャッシュおよび再利用できます。Microsoft Power BI などの BI ツールを使用するとき、このオプションを使用して開かれるブラウザーのウィンドウ数を減らすことができます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
browser_azure_cache	オプションです。	1	`browser_azure_cache=0;`

Browser SAML

Browser SAML は、SAML ベースの ID プロバイダと連携する汎用認証プラグインであり、多要素認証をサポートします。詳細な設定情報については、「ODBC、SAML 2.0、および Okta ID プロバイダを使用したシングルサインオンの設定」を参照してください。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=BrowserSAML;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。SAML アサーションに複数の役割がある場合は、このパラメータを指定して引き受ける役割を選択できます。このロールは SAML アサーションに含まれている必要があります。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	`900`	`duration=900;`

ログイン URL

アプリケーションに表示されるシングルサインオン URL。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
login_url	必須	none	`login_url =https:// trial-123 4567.okta.com/ app/trial-123 4567_okta browsersaml_1/ zzz4izzzAzDFB zZz1234/sso/ saml;`

リッスンポート

SAML レスポンスをリッスンするために使用されるポート番号。この値は、IdP を設定した IAM Identity Center の URL (例: `http://localhost:7890/athena`) と一致する必要があります。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
listen_port	オプションです。	`7890`	`listen_po rt=7890;`

タイムアウト

プラグインが ID プロバイダからの SAML レスポンスが得られるまで待機するのを停止するまでの時間 (秒単位)。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
timeout	オプションです。	`120`	`timeout=90;`

Browser SSO OIDC

Browser SSO OIDC は AWS IAM Identity Center で動作する認証プラグインです。IAM Identity Center を有効にして使用するための情報については、「AWS IAM Identity Center ユーザーガイド」の「ステップ 1: IAM Identity Center を有効にする」を参照してください。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=BrowserSSOOIDC;`

IAM Identity Center の開始 URL

AWS アクセスポータルの URL。IAM Identity Center の StartDeviceAuthorization API アクションは、この値を `startUrl` パラメータに使用します。

AWS アクセスポータル URL をコピーする方法

1. AWS Management Console にサインインし、AWS IAM Identity Center コンソールを https://console.aws.amazon.com/singlesignon/ で開きます。
2. ナビゲーションペインで [設定] を選択します。
3. [設定] ページにある [ID ソース] で、AWS アクセスポータル URL のクリップボードアイコンを選択します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
sso_oidc_start_url	必須	none	`sso_oidc_start_url=https://app_id.awsapps.com/start;`

IAM Identity Center リージョン

SSO が設定されている AWS リージョン。`SSOOIDCClient` および `SSOClient` AWS SDK クライアントは、この値を `region` パラメータに使用します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
sso_oidc_region	必須	none	`sso_oidc_region=us-east-1;`

スコープ

クライアントによって定義されるスコープのリスト。承認されると、このリストはアクセストークンが付与されたときの権限を制限します。IAM Identity Center の RegisterClient API アクションは、この値を `scopes` パラメータに使用します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
sso_oidc_scopes	オプションです。	none	`sso_oidc_scopes=scope1,scope2,scope3;`

アカウント ID

ユーザーに割り当てられる AWS アカウント の識別子。IAM Identity Center の GetRoleCredentials API は、この値を `accountId` パラメータに使用します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
sso_oidc_account_id	必須	none	`sso_oidc_account_id=123456789123;`

ロール名

ユーザーに割り当てられているロールのわかりやすい名前。このアクセス許可セットに指定した名前は、使用可能なロールとして AWS アクセスポータルに表示されます。IAM Identity Center の GetRoleCredentials API アクションは、この値を `roleName` パラメータに使用します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
sso_oidc_role_name	必須	none	`sso_oidc_role_name=AthenaReadAccess;`

タイムアウト

ポーリング SSO API がアクセストークンをチェックする秒数。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
sso_oidc_timeout	オプションです。	`120`	`sso_oidc_timeout=60;`

ファイルキャッシュを有効にする

一時的な認証情報キャッシュを有効にします。この接続パラメータにより、一時的な認証情報を複数のプロセス間でキャッシュおよび再利用できます。Microsoft Power BI などの BI ツールを使用するとき、このオプションを使用して開かれるブラウザーのウィンドウ数を減らすことができます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
sso_oidc_cache	オプションです。	1	`sso_oidc_cache=0;`

デフォルト認証情報

クライアントシステムで設定したデフォルトの認証情報を使用して、Amazon Athena に接続できます。デフォルト認証情報の使用に関する詳細については、「AWS SDK for Java デベロッパーガイド」の「デフォルト認証情報プロバイダチェーンの使用」を参照してください。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=DefaultCredentials;`

外部認証情報

外部認証情報は、任意の外部 SAML ベースの ID プロバイダに接続する際に使用できる汎用認証プラグインです。プラグインを使用するには、SAML レスポンスを返す実行可能ファイルを渡します。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=External Credentials;`

実行可能ファイルのパス

カスタム SAML ベースの認証情報プロバイダのロジックが含まれる実行可能ファイルへのパス。実行可能ファイルの出力は、ID プロバイダからの解析された SAML レスポンスである必要があります。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ExecutablePath	必須	none	ExecutablePath=C:\Users*user_name* *external_credential.exe*

引数リスト

実行可能ファイルに渡すべき引数のリスト。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ArgumentList	オプションです。	none	`ArgumentList=` *`arg1 arg2 arg3`*

インスタンスプロファイル

この認証タイプは EC2 インスタンスで使用され、Amazon EC2 メタデータサービスを通じて提供されます。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=Instance Profile;`

JWT

JSON ウェブトークン (JWT) プラグインは、JSON ウェブトークンを使用して Amazon IAM ロールを引き受けるインターフェイスを提供します。設定は、ID プロバイダによって異なります。Google

Cloud および AWS でのフェデレーション設定については、Google Cloud ドキュメントの「AWS または Azure との Workload Identity 連携を構成する」を参照してください。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=JWT;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。セッション期間の詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	`900`	`duration=900;`

JSON ウェブトークン

AWS STS の AssumeRoleWithWebIdentity API アクションを使用して、IAM の一時的な認証情報を取得するために使用される JSON ウェブトークン。Google Cloud プラットフォーム (GCP) ユーザー

向けに JSON ウェブトークンを生成するための詳細については、「Google Cloud ドキュメント」の「JWT OAuth トークンの使用」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
web_identity_token	必須	none	web_identity_token=eyJhbGc...<remainder of token>;

ロールセッション名

セッションの名前。アプリケーションのユーザーの名前あるいは識別子をロールセッション名として使用するのが一般的なテクニックです。これにより、アプリケーションが使用する一時的なセキュリティ認証情報を、該当するユーザーと簡単に関連付けられます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
role_session_name	必須	none	role_session_name=familiarname;

Okta

Okta は Okta ID プロバイダと連携する SAML ベースの認証プラグインです。Okta と Amazon Athena のフェデレーションを設定するための情報については、「Okta プラグインと Okta ID プロバイダを使用して ODBC 用 の SSO を設定する」を参照してください。

認証タイプ

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	IAM Credentials	AuthenticationType=Okta;

ユーザー ID

Okta のユーザー名。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UID	必須	none	`UID=jane.doe@org.com;`

パスワード

Okta ユーザーのパスワード。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
PWD	必須	none	`PWD=oktauserpasswordexample;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	900	duration=900;

IdP Host

Okta 組織の URL。Okta アプリケーションの [埋め込みリンク] URL から、idp_host パラメータを抽出できます。この手順については、「Okta から ODBC の設定情報を取得する」を参照してください。https:// の後に続く最初のセグメントから okta.com までは、IdP ホストです (例: http://trial-1234567.okta.com)。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_host	必須	None	idp_host=dev-99999999.okta.com;

IdP ポート

IdP ホストへの接続に使用するポート番号。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_port	必須	None	idp_port=443;

Okta アプリ ID

アプリケーション用の 2 つの部分で構成される識別子です。Okta アプリケーションの [埋め込みリンク] URL から、app_id パラメータを抽出できます。この手順については、「Okta から ODBC の設定情報を取得する」を参照してください。アプリケーション ID は、URL の最後の 2 つのセグメントで、中央のスラッシュも含まれます。これら 2 つのセグメントは、数字と大文字と小文字が混在する 20 文字の文字列 2 組です (例: Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4)。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
app_id	必須	None	app_id=0oa25kx8ze9A3example/alnexamplea0piaWa0g7;

Okta アプリ名

Okta アプリケーションの名前。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
app_name	必須	None	app_name=amazon_aws_redshift;

Okta の待機時間

多要素認証 (MFA) コードを取得するまで待機する時間を、秒単位で指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
okta_mfa_wait_time	オプションです。	10	okta_mfa_wait_time=20;

Okta MFA タイプ

MFA ファクターのタイプ。サポートされているタイプは、Google 認証システム、SMS (Okta)、プッシュ通知付き Okta Verify、および TOTP 付き Okta Verify です。個々の組織のセキュリティポリシーによって、ユーザーのログインに MFA が必要かどうかが決まります。

接続文字列名	パラメータタイプ	デフォルト値	使用できる値	接続文字列の例
okta_mfa_type	Optional	None	googleauthenticator, smsauthentication, oktaverifywithpush, oktaverifywithtotp	okta_mfa_type=oktaverifywithpush;

Okta 電話番号

AWS SMS 認証に使用する電話番号。このパラメータは、多要素登録にのみ必要です。携帯電話番号がすでに登録されている、またはセキュリティポリシーで AWS SMS 認証が使用されていない場合は、このフィールドは無視してかまいません。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
okta_mfa_phone_number	MFA 登録には必須、それ以外の場合はオプション	None	okta_mfa_phone_number=19991234567;

Okta ファイルキャッシュを有効にする

一時的な認証情報キャッシュを有効にします。この接続パラメータにより、一時的な認証情報をキャッシュし、BI アプリケーションが開いた複数のプロセス間で再利用できます。このオプションを使用して Okta API のスロットリング制限を回避します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
okta_cache	オプションです。	0	okta_cache=1;

Ping

Ping は PingFederate ID プロバイダと連携する SAML ベースのプラグインです。

[Authentication type] (認証タイプ)

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
AuthenticationType	必須	`IAM Credentials`	`AuthenticationType=Ping;`

ユーザー ID

PingFederate サーバーのユーザー名。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UID	必須	`none`	`UID=pingusername@domain.com;`

パスワード

PingFederate サーバーのパスワード。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
PWD	必須	`none`	`PWD=pingpassword;`

優先ロール

引き受けるロールの Amazon リソースネーム (ARN)。SAML アサーションに複数の役割がある場合は、このパラメータを指定して引き受ける役割を選択できます。このロールは SAML アサーションに含まれている必要があります。ARN ロールの詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
preferred_role	オプションです。	none	`preferred_role=arn:aws:iam::123456789012:id/user1;`

セッション期間

ロールセッションの期間 (秒)。セッション期間の詳細については、「AWS Security Token Service API リファレンス」の「AssumeRole」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
duration	オプションです。	900	`duration=900;`

IdP Host

Ping サーバーのアドレス。アドレスを確認するには、次の URL にアクセスして [SSO アプリケーションエンドポイント] フィールドを確認してください。

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_host	必須	none	`idp_host=ec2-1-83-65-12.compute-1.amazonaws.com;`

IdP ポート

IdP ホストへの接続に使用するポート番号。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
idp_port	必須	None	`idp_port=443;`

パートナー SPID

サービスプロバイダのアドレス。サービスプロバイダのアドレスを確認するには、次の URL にアクセスして [SSO アプリケーションエンドポイント] フィールドを確認してください。

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
partner_spid	必須	None	`partner_spid=https://us-east-1.signin.aws.amazon.com/platform/saml/<...>;`

Ping URI パラメータ

認証リクエストの URI 引数を Ping に渡します。このパラメータを使用して Lake Formation のシングルロール制限を回避します。渡されたパラメータを認識するように Ping を設定し、渡されたロールがユーザーに割り当てられたロールのリストに存在することを確認します。次に SAML アサーションで 1 つのロールを送信します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ping_uri_param	オプションです。	None	`ping_uri_param=role=my_iam_role;`

一般的な認証パラメータ

このセクションのパラメータは、前述のように認証タイプに共通です。

IdP にプロキシを使用する

プロキシを介したドライバーと IdP 間の通信を有効にします。このオプションは、以下の認証プラグインで使用できます。

- AD FS
- Azure AD
- Browser Azure AD
- Browser SSO OIDC
- JWT
- Okta
- Ping

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseProxyForIdP	オプションです。	`0`	`UseProxyForIdP=1;`

Lake Formation の使用

AssumeRoleWithSAML AWS STS API アクションの代わりに AssumeDecoratedRoleWithSAML Lake Formation API アクションを使用して、一時的な IAM 認証情報を取得します。このオプションは、Azure AD、Browser Azure AD、Browser SAML、Okta、Ping、および AD FS 認証プラグインで使用できます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
LakeformationEnabled	オプションです。	`0`	`LakeformationEnabled=1;`

SSL インセキュア (IdP)

IdP と通信するときに SSL を無効にします。このオプションは、Azure AD、Browser Azure AD、Okta、Ping、および AD FS 認証プラグインで使用できます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
SSL_Insecure	オプションです。	`0`	`SSL_Insecure=1;`

エンドポイントオーバーライド

Athena エンドポイントオーバーライド

endpointOverride ClientConfiguration クラスはこの値を使用して Amazon Athena クライアントのデフォルトの HTTP エンドポイントをオーバーライドします。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
EndpointOverride	オプションです。	none	`EndpointOverride=athena.us-west-2.amazonaws.com;`

Athena ストリーミングエンドポイントのオーバーライド

ClientConfiguration.endpointOverride メソッドはこの値を使用して Amazon Athena ストリーミングクライアントのデフォルトの HTTP エンドポイントをオーバーライドします。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。Athena ストリーミングサービスはポート 444 から利用できます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
Streaming EndpointOverride	オプションです。	none	`StreamingEndpointOverride=athena.us-west-1.amazonaws.com:444;`

AWS STS エンドポイントのオーバーライド

ClientConfiguration.endpointOverride メソッドはこの値を使用して AWS STS クライアントのデフォルトの HTTP エンドポイントをオーバーライドします。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
StsEndpointOverride	オプションです。	none	`StsEndpointOverride=sts.us-west-1.amazonaws.com;`

Lake Formation エンドポイントのオーバーライド

`ClientConfiguration.endpointOverride` メソッドはこの値を使用して Lake Formation クライアントのデフォルトの HTTP エンドポイントをオーバーライドします。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
LakeFormationEndpointOverride	オプションです。	none	`LakeFormationEndpointOverride=lakeformation.us-west-1.amazonaws.com;`

SSO エンドポイントのオーバーライド

`ClientConfiguration.endpointOverride` メソッドは、この値を使用して SSO クライアントのデフォルト HTTP エンドポイントをオーバーライドします。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
SSOEndpointOverride	オプションです。	none	`SSOEndpointOverride=portal.sso.us-east-2.amazonaws.com;`

SSO OIDC エンドポイントのオーバーライド

`ClientConfiguration.endpointOverride` メソッドは、この値を使用して SSO OIDC クライアントのデフォルト HTTP エンドポイントをオーバーライドします。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
SSOOIDCEndpointOverride	オプションです。	none	`SSOOIDCEndpointOverride=oidc.us-east-2.amazonaws.com`

詳細オプション

フェッチサイズ

このリクエストで返す結果 (行) の最大数。パラメータ情報については、「GetQuery MaxResults」を参照してください。ストリーミング API の場合、最大値は 10000000 です。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
RowsToFetchPerBlock	オプションです。	非ストリーミング用の `1000` ストリーミング用の `20000`	`RowsToFetchPerBlock=20000;`

結果の再利用を有効にする

クエリの実行時に以前のクエリ結果を再利用できるかどうかを指定します。パラメータ情報については、「ResultReuseByAgeConfiguration」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
EnableResultReuse	オプションです。	`0`	`EnableResultReuse=1;`

結果再利用の上限有効期間

Athena が再利用を検討するべき以前のクエリ結果の最大保存期間を、分単位で指定します。パラメータ情報については、「ResultReuseByAgeConfiguration」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ReusedResultMaxAgeInMinutes	オプションです。	60	ReusedResultMaxAgeInMinutes=90;

ストリーミング API を有効にする

Athena ストリーミング API を使用して結果セットを取得するかどうかを選択します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseResultsetStreaming	オプションです。	0	UseResultsetStreaming=1;

S3 フェッチャーを有効にする

Athena が生成された結果セットを、Amazon S3 と直接やり取りして Amazon S3 バケットから取得します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
EnableS3Fetcher	オプションです。	1	EnableS3Fetcher=1;

複数の S3 スレッドを使用する

複数のスレッドを使用して Amazon S3 からデータを取得します。このオプションを有効にすると、Amazon S3 バケットに保存された結果ファイルが、複数のスレッドを使用して同時に取得されます。

このオプションは、ネットワーク帯域幅が十分である場合にのみ有効にしてください。たとえば、EC2 c5.2xlarge インスタンスでの測定では、シングルスレッドの S3 クライアントは 1 Gbps に達し、マルチスレッドの S3 クライアントではネットワークスループットが 4 Gbps に達しました。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseMultipleS3Threads	オプションです。	0	UseMultipleS3Threads=1;

単一のカタログとスキーマを使用

デフォルトでは、ODBC ドライバーは Athena にクエリを実行して、使用可能なカタログとスキーマのリストを取得します。このオプションでは、[ODBC データソースの管理者] 設定ダイアログボックスまたは接続パラメータで指定されたカタログとスキーマをドライバーに強制的に使用させます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseSingleCatalogAndSchema	オプションです。	0	UseSingleCatalogAndSchema=1;

クエリを使用してテーブルをリストする

LAMBDA カタログ型の場合に、ODBC ドライバーが SHOW TABLES クエリを送信して利用可能なテーブルのリストを取得できるようにします。この設定はデフォルトです。このパラメータが 0 に設定されていると、ODBC ドライバーは Athena ListTableMetadata API を使用して利用可能なテーブルのリストを取得します。LAMBDA カタログ型では、ListTableMetadata の使用がパフォーマンスの低下につながることに注意してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseQueryToListTables	オプションです。	1	UseQueryToListTables=1;

文字列型に WCHAR を使用する

デフォルトで、ODBC ドライバーは Athena 文字列型
char、varchar、string、array、map<>、struct<>、および row に、SQL_CHAR と

SQL_VARCHAR を使用します。このパラメータを 1 に設定すると、文字列型への SQL_WCHAR と SQL_WVARCHAR の使用をドライバーに強制します。ワイド文字型とワイド可変文字型は、異なる言語からの文字が正しく保存および取得されることを確実にするために使用されます。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseWCharForStringTypes	オプションです。	`0`	`UseWCharForStringTypes=1;`

外部カタログをクエリする

ドライバーが Athena から外部カタログをクエリする必要があるかどうかを指定します。詳細については、「ODBC 2.x ドライバーへの移行」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
QueryExternalCatalogs	オプションです。	`0`	`QueryExternalCatalogs=1;`

SSL の検証

AWS SDK を使用するときに、SSL 証明書を検証するかどうかを管理します。この値は `ClientConfiguration.verifySSL` パラメータに渡されます。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
VerifySSL	オプションです。	`1`	`VerifySSL=0;`

S3 結果のブロックサイズ

1 回の Amazon S3 GetObject API リクエストでダウンロードするブロックのサイズをバイト単位で指定します。デフォルト値は 67108864 (64 MB) です。許容される最小値と最大値は 10485760 (10 MB) と 2146435072 (約 2 GB) です。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
S3ResultBlockSize	オプションです。	67108864	S3ResultBlockSize=268435456;

文字列の列長

string データ型の列の列長を指定します。Athena は精度が定義されていない Apache Hive 文字列データ型を使用するため、Athena が報告するデフォルト長は 2147483647 (INT_MAX) になります。通常、BI ツールは列のメモリを事前に割り当てるため、メモリ消費量の増加につながる可能性があります。これを回避するため、Athena ODBC ドライバーは string データ型の列について報告される精度を制限するとともに、StringColumnLength 接続パラメータを公開してデフォルト値を変更できるようにします。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
StringColumnLength	オプションです。	255	StringColumnLength=65535;

複合型の列長

map、struct、および array といった複合データ型の列の列長を指定します。StringColumnLength と同様に、Athena は複合データ型の列にも 0 精度を報告します。Athena ODBC ドライバーは、複合データ型の列のデフォルト精度を設定するとともに、ComplexTypeColumnLength 接続パラメータを公開してデフォルト値を変更できるようにします。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ComplexTypeColumnLength	オプションです。	65535	ComplexTypeColumnLength=123456;

信頼できる CA 証明書

SSL 証明書の信頼ストアの場所を HTTP クライアントに指示します。この値は `ClientConfiguration.caFile` パラメータに渡されます。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
TrustedCerts	オプションです。	`%INSTALL_PATH%/bin`	`TrustedCerts=C:\\Program Files\\Amazon Athena ODBC Driver\\bin\\cacert.pem;`

最小ポーリング時間

Athena にクエリ実行ステータスをポーリングする前に、待機する最小値をミリ秒単位で指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
MinQueryExecutionPollingInterval	オプションです。	`100`	`MinQueryExecutionPollingInterval=200;`

最大ポーリング時間

Athena にクエリ実行ステータスをポーリングする前に、待機する最大値をミリ秒単位で指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
MaxQueryExecutionPollingInterval	オプションです。	`60000`	`MaxQueryExecutionPollingInterval=1000;`

ポーリングの乗数

ポーリング時間を延長する要素を指定します。デフォルトでは、ポーリングは最小ポーリング時間の値から始まり、最大ポーリング時間の値に達するまでポーリングごとに倍増します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
QueryExecutionPollingIntervalMultiplier	オプションです。	2	`QueryExecutionPollingIntervalMultiplier=2;`

最大ポーリング時間

ドライバーは Athena にクエリ実行ステータスをポーリングできる時間の最大値をミリ秒単位で指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
MaxPollDuration	オプションです。	`1800000`	`MaxPollDuration=1800000;`

接続タイムアウト

接続を確立するまでに HTTP 接続が待機する時間 (ミリ秒単位)。この値は `ClientConfiguration.connectTimeoutMs` Athena クライアントで設定されます。指定されていない場合、curl デフォルト値が使用されます。接続パラメータに関する詳細については、「AWS SDK for Java デベロッパーガイド」の「クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
[ConnectionTimeout]	オプションです。	`0`	`ConnectionTimeout=2000;`

リクエストのタイムアウト

HTTP クライアントのソケット読み取りのタイムアウト値を指定します。この値は Athena クライアントの `ClientConfiguration.requestTimeoutMs` パラメータで設定します。詳細については、「AWS SDK for Java デベロッパーガイド」の「クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
RequestTimeout	オプションです。	`10000`	`RequestTimeout=30000;`

プロキシオプション

プロキシのホスト

ユーザーにプロキシ経由のログインを要求する場合は、このパラメータを使用してプロキシホストを設定します。このパラメータは、AWS SDK 内の `ClientConfiguration.proxyHost` パラメータに対応します。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ProxyHost	オプションです。	none	`ProxyHost=127.0.0.1;`

プロキシのポート

このパラメータを使用して、プロキシポートを設定します。このパラメータは、AWS SDK 内の `ClientConfiguration.proxyPort` パラメータに対応します。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ProxyPort	オプションです。	none	`ProxyPort=8888;`

プロキシのユーザー名

このパラメータを使用して、プロキシのユーザー名を設定します。このパラメータは、AWS SDK 内の `ClientConfiguration.proxyUserName` パラメータに対応します。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ProxyUID	オプションです。	none	`ProxyUID=username;`

プロキシのパスワード

このパラメータを使用して、プロキシのパスワードを設定します。このパラメータは、AWS SDK 内の `ClientConfiguration.proxyPassword` パラメータに対応します。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
ProxyPWD	オプションです。	none	`ProxyPWD=password;`

非プロキシホスト

このオプションパラメータを使用して、ドライバーがプロキシを使用せずに接続するホストを指定します。このパラメータは、AWS SDK 内の `ClientConfiguration.nonProxyHosts` パラメータに対応します。詳細については、「AWS SDK for C++ デベロッパーガイド」の「AWS クライアント設定」を参照してください。

NonProxyHost 接続パラメータは、curl の CURLOPT_NOPROXY オプションに渡されます。CURLOPT_NOPROXY 形式の詳細については、curl のドキュメントで「CURLOPT_NOPROXY」を参照してください。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
NonProxyHost	オプションです。	none	NonProxyHost=.amazonaws.com,localhost,.example.net,.example.com;

プロキシの使用

指定されたプロキシを経由するユーザートラフィックを有効にします。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseProxy	オプションです。	none	UseProxy=1;

ログ記録オプション

ここで説明する設定を変更するには、管理者権限が必要です。変更するには、ODBC データソース管理者の [ロギングオプション] ダイアログボックスを使用するか、Windows レジストリを直接変更します。

ログレベル

このオプションは ODBC ドライバーログを有効にします。Windows では、レジストリまたはダイアログボックスを使用することで、ロギングを有効または無効にできます。オプションは次のレジストリパスにあります。

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Amazon Athena\ODBC\Driver
```

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
LogLevel	オプションです。	0	LogLevel=1;

ログパス

ODBC ドライバーログが保存されているファイルへのパスを指定します。レジストリまたはダイアログボックスを使用して、この値を設定できます。オプションは次のレジストリパスにあります。

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Amazon Athena\ODBC\Driver
```

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
LogPath	オプションです。	none	LogPath=C:\Users\ *username*\projects\internal\trunk\;

AWS ロガーを使用する

AWS SDK ロギングを有効にするかどうかを指定します。有効にするには 1 を、無効にするには 0 を指定します。

接続文字列名	パラメータタイプ	デフォルト値	接続文字列の例
UseAwsLogger	オプションです。	0	UseAwsLogger=1;

ODBC 2.x ドライバーへの移行

ほとんどの Athena ODBC 2.x 接続パラメータは ODBC 1.x ドライバーとの下位互換性があるため、既存の接続文字列のほとんどは Athena ODBC 2.x ドライバーでも引き続き使用できます。ただし、次の接続パラメータは変更が必要です。

ログレベル

現行の ODBC ドライバーでは、LOG_OFF（0）から LOG_TRACE（6）までのさまざまなログオプションがありますが、Amazon Athena ODBC ドライバーには 0 (無効) と 1 (有効) の 2 つの値しかありません。

ODBC 2.x ドライバーを口グするための詳細については、「ログ記録オプション」を参照してください。

	ODBC 1.x ドライバー	ODBC 2.x ドライバー
接続文字列名	LogLevel	LogLevel

	ODBC 1.x ドライバー	ODBC 2.x ドライバー
パラメータタイプ	オプションです。	オプションです。
デフォルト値	`0`	`0`
使用できる値	`0-6`	`0,1`
接続文字列の例	`LogLevel=6;`	`LogLevel=1;`

MetadataRetrievalMethod

現行の ODBC ドライバーには、Athena からメタデータを取得するためのオプションがいくつか用意されています。Amazon Athena ODBC ドライバーでは `MetadataRetrievalMethod` が廃止されており、メタデータの抽出には常に Amazon Athena API を使用することになります。

Athena では、外部カタログをクエリするための `QueryExternalCatalogs` フラグが導入されています。現行の ODBC ドライバーを使用して外部カタログをクエリするには、`MetadataRetrievalMethod` を `ProxyAPI` に設定します。Athena ODBC ドライバーを使用して外部カタログをクエリするには、`QueryExternalCatalogs` を `1` に設定します。

	ODBC 1.x ドライバー	ODBC 2.x ドライバー
接続文字列名	`MetadataRetrievalMethod`	`QueryExternalCatalogs`
パラメータタイプ	オプションです。	オプションです。
デフォルト値	`Auto`	`0`
使用できる値	`Auto`, `AWS Glue`, `ProxyAPI`, `Query`	`0`,`1`
接続文字列の例	`MetadataRetrievalMethod=ProxyAPI;`	`QueryExternalCatalogs=1;`

接続テスト

ODBC 1.x ドライバーの接続をテストすると、ドライバーは SELECT 1 クエリを実行して Amazon S3 バケットに 2 つのファイルを生成します。1 つは結果セット用、もう 1 つはメタデータ用です。テスト接続は Amazon Athena の料金ポリシーに従って請求されます。

ODBC 2.x ドライバーの接続をテストすると、ドライバーは GetWorkGroup Athena API アクションをコールします。コールでは、指定した認証タイプと対応する認証情報プロバイダを使用して、認証情報を取得します。ODBC 2.x ドライバーを使用する場合、接続テストは無料です。また、テストでは Amazon S3 バケットにクエリ結果は生成されません。

ODBC 2.x ドライバーのトラブルシューティング

Amazon Athena ODBC ドライバーで問題が発生した場合は、AWS Support (AWS Management Console で [サポート] > [サポートセンター] の順に選択) にお問い合わせください。

以下の情報を必ず含め、サポートチームがユースケースを理解するのに役立つ追加情報があれば提供してください。

- 詳細 — (必須) 使用されているユースケースに関する詳細な情報と、予想される動作と観察された動作の違いも含めてご説明ください。サポートエンジニアが問題を簡単に解決するのに役立つ情報があれば、すべて含めてください。問題が断続的に発生する場合は、問題が発生した日付、タイムスタンプ、または問題が発生する間隔について具体的にお知らせください。
- バージョン情報 — (必須) ドライバーのバージョン、オペレーティングシステム、およびご使用されているアプリケーションに関する情報。たとえば「ODBC ドライバーバージョン 1.2.3、Windows 10 (x64)、Power BI」など。
- ログファイル — (必須) 問題を理解するために必要となる ODBC ドライバーログファイルを最小限の数でご提供ください。ODBC 2.x ドライバーをログするための詳細については、「ログ記録オプション」を参照してください。
- 接続文字列 — (必須) 使用されている ODBC 接続文字列、または使用した接続パラメータを示すダイアログボックスのスクリーンショット。接続パラメータの詳細については、「Athena ODBC 2.x 接続パラメータ」を参照してください。
- 問題が発生する手順 — (オプション) 可能な場合は、問題の再現に役立つ手順あるいはスタンドアロンプログラムを含めてください。
- クエリエラー情報 — (オプション) DML または DDL クエリに関連するエラーがある場合は、次の情報を含めてください。
  - 失敗した DML または DDL クエリの完全版または簡略版。

  - 使用されたアカウント ID、AWS リージョン、およびクエリ実行 ID。
- SAML エラー — (オプション) SAML アサーションを使用した認証に関する問題がある場合は、次の情報を含めてください。
  - 使用された ID プロバイダと認証プラグイン。
  - SAML トークンの例。

Amazon Athena ODBC 2.x リリースノート

このリリースノートには、Amazon Athena ODBC 2.x ドライバーの強化内容、機能、既知の問題、ワークフローの変更点の詳細が記載されています。

2.0.3.0

公開日: 2024 年 4 月 8 日

Amazon Athena ODBC v2.0.3.0 ドライバーには、以下の改善と修正が含まれています。

改良点

- Linux および Mac プラットフォーム上の Okta 認証プラグインに対する MFA サポートを追加しました。
- Windows 用の `athena-odbc.dll` ライブラリと `AmazonAthenaODBC-2.x.x.x.msi` インストーラの両方が署名付きになりました。
- ドライバーと共にインストールされる CA 証明書 `cacert.pem` ファイルを更新しました。
- Lambda カタログのテーブルを一覧表示するために必要な時間が改善されました。LAMBDA カタログ型では、ODBC ドライバーが SHOW TABLES クエリを送信して利用可能なテーブルのリストを取得できるようになりました。詳細については、「クエリを使用してテーブルをリストする」を参照してください。
- `SQL_WCHAR` と `SQL_WVARCHAR` を使用して文字列データ型を報告するための `UseWCharForStringTypes` 接続パラメータを導入しました。詳細については、「文字列型に WCHAR を使用する」を参照してください。

修正内容

- Get-ODBCDSN PowerShell ツールの使用時に発生していたレジストリの破損警告を修正しました。
- クエリ文字列の先頭にあるコメントを処理するように解析ロジックを更新しました。

- 日付とタイムスタンプのデータ型で、年フィールドに 0 を使用できるようになりました。

新しい ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

2.0.2.2

公開日: 2024 年 2 月 13 日

Amazon Athena ODBC v2.0.2.2 ドライバーには、以下の改善と修正が含まれています。

改良点

- `StringColumnLength` と `ComplexTypeColumnLength` の 2 つの接続パラメータを追加しました。これらは、文字列と複合データ型のデフォルト列長を変更するために使用できます。詳細については、文字列の列長および複合型の列長を参照してください。
- Linux および macOS (Intel と ARM) オペレーティングシステムのサポートが追加されました。詳細については、LinuxおよびmacOSを参照してください。
- AWS-SDK-CPP が 1.11.245 タグバージョンに更新されました。
- curl ライブラリが 8.6.0 バージョンに更新されました。

修正内容

- 精度列内にある文字列に似たデータ型の結果セットメタデータで誤った値が報告される原因となっていた問題を解決しました。

ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

2.0.2.1

公開日: 2023 年 12 月 7 日

Amazon Athena ODBC v2.0.2.1 ドライバーには、次の改善と修正が含まれています。

改良点

- すべてのインターフェースの ODBC ドライバースレッドセーフティが改善されました。
- ロギングを有効にすると、日時の値がミリ秒の精度で記録されるようになりました。

- Browser SSO OIDC プラグインによる認証中に、ターミナルが開き、ユーザーにデバイスコードが表示されるようになりました。

修正内容

- ストリーミング API からの結果を解析する際に発生するメモリ解放の問題を解決しました。
- インターフェイス `SQLTablePrivileges()`、`SQLSpecialColumns()`、`SQLProcedureColumns()`、`SQLProcedures(` のリクエストは空の結果セットを返すようになりました。

ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

2.0.2.0

公開日: 2023 年 10 月 17 日

Amazon Athena ODBC v2.0.2.0 ドライバーには、次の改善と修正が含まれています。

改良点

- Browser Azure AD、Browser SSO OIDC、Okta ブラウザベースの認証プラグインにファイルキャッシュ機能が追加されました。

  Power BI やブラウザーベースのプラグインなどの BI ツールは複数のブラウザーウィンドウを使用します。新しいファイルキャッシュ接続パラメータにより、一時的な認証情報をキャッシュし、BI アプリケーションが開いた複数のプロセス間で再利用できます。
- ステートメントが作成された後、アプリケーションが結果セットに関する情報をクエリできるようになりました。
- 遅いクライアントネットワークで使用できるように、デフォルト接続およびリクエストのタイムアウトが延長されました。詳細については、接続タイムアウトおよびリクエストのタイムアウトを参照してください。
- SSO および SSO OIDC にエンドポイントオーバーライドが追加されました。詳細については、「エンドポイントオーバーライド」を参照してください。
- 認証リクエストの URI 引数を Ping に渡す接続パラメータが追加されました。このパラメータを使用して Lake Formation のシングルロール制限を回避できます。詳細については、「Ping URI パラメータ」を参照してください。

修正内容

- 行ベースのバインディングメカニズムを使用する際に発生した整数オーバーフローの問題を修正しました。
- Browser SSO OIDC 認証プラグインの必須接続パラメータのリストからタイムアウトを削除しました。
- `SQLStatistics()`、`SQLPrimaryKeys()`、`SQLForeignKeys()`、`SQLColumnPrivileges()` に欠けていたインターフェイスを追加し、リクエストに応じて空の結果セットを返す機能を追加しました。

新しい ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

2.0.1.1

公開日: 2023 年 8 月 10 日

Amazon Athena ODBC v2.0.1.1 ドライバーには、以下の改善と修正が含まれています。

改良点

- Okta 認証プラグインに URI ロギングを追加しました。
- 外部認証情報プロバイダープラグインに優先ロールパラメータを追加しました。
- AWS 設定ファイルのプロファイル名にプロファイルプレフィックスの処理を追加しました。

修正内容

- Lake Formation および AWS STS クライアントの使用時に発生した AWS リージョン使用の問題を修正しました。
- 欠落しているパーティションキーをテーブル列のリストに復元しました。
- 欠落している `BrowserSSOOIDC` 認証タイプを AWS プロファイルに追加しました。

新しい ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。

2.0.1.0

公開日: 2023 年 6 月 29 日

Amazon Athena は ODBC v2.0.1.0 ドライバーをリリースしました。

Athena は、互換性のある SQL 開発、ビジネスインテリジェンスアプリケーションからのデータへの接続、クエリ、および視覚化のエクスペリエンスを向上させる新しい ODBC ドライバーをリリースしました。Athena ODBC ドライバーの最新バージョンでは、既存のドライバーの機能がサポートされており、簡単にアップグレードできます。新しいバージョンには、AWS IAM Identity Center によるユーザー認証のサポートが含まれています。また、Amazon S3 からクエリ結果を読み取るオプションも用意されているため、クエリ結果もより早く利用できるようになります。

詳細については、「Amazon Athena ODBC 2.x」を参照してください。

Athena ODBC 1.x ドライバー

このページのリンクを使用して、Amazon Athena 1.x ODBC ドライバーライセンス契約、ODBC ドライバー、および ODBC ドキュメントをダウンロードします。ODBC 接続文字列の詳細については、ODBC ドライバーのインストールおよび設定ガイド PDF ファイル (このページからダウンロード可能) を参照してください。アクセス許可については、「JDBC および ODBC 接続を介したアクセス」を参照してください。

> **Important**
>
> ODBC 1.x ドライバーを使用するときは、次の要件に注意してください。
>
> - オープン状態のポート 444 – Athena がクエリ結果のストリーミングに使用するポート 444 には、アウトバウンドトラフィックに対して開放されている状態を維持します。PrivateLink エンドポイントを使用して Athena に接続するときは、PrivateLink エンドポイントにアタッチされているセキュリティグループが、ポート 444 上のインバウンドトラフィックに対して開放されていることを確認してください。
> - athena:GetQueryResultsStream ポリシー – ODBC ドライバーを使用する IAM プリンシパルに `athena:GetQueryResultsStream` ポリシーアクションを追加します。このポリシーアクションが API で直接公開されることはありません。ストリーミング結果のサポートの一環として、ODBC および JDBC ドライバーでのみ使用されます。ポリシーの例については、「AWS 管理ポリシー: AWSQuicksightAthenaAccess」を参照してください。

Windows

ドライバーのバージョン	ダウンロードリンク
Windows 32 ビット版対応 ODBC 1.2.3.1000	Windows 32 ビット ODBC ドライバー 1.2.3.1000
Windows 64 ビット版対応 ODBC 1.2.3.1000	Windows 64 ビット ODBC ドライバー 1.2.3.1000

Linux

ドライバーのバージョン	ダウンロードリンク
Linux 32 ビット版対応 ODBC 1.2.3.1000	Linux 32 ビット ODBC ドライバー 1.2.3.1000
Linux 64 ビット版対応 ODBC 1.2.3.1000	Linux 64 ビット ODBC ドライバー 1.2.3.1000

OSX

ドライバーのバージョン	ダウンロードリンク
OSX 対応 ODBC 1.2.3.1000	OSX ODBC ドライバー 1.2.3.1000

ドキュメント

コンテンツ	ドキュメントのリンク
Amazon Athena ODBC ドライバーライセンス契約	ライセンス契約
ODBC 1.2.3.1000 に関するドキュメント	ODBC ドライバーのインストールおよび設定ガイドバージョン 1.2.3.1000
ODBC 1.2.3.1000 のリリースノート	ODBC ドライバーリリースノートバージョン 1.2.3.1000

ODBC ドライバーノート

プロキシを使用しない接続

プロキシを使用せずにドライバが接続する特定のホストを指定する場合は、ODBC 接続文字列でオプションの NonProxyHost プロパティを使用します。

次の例のように NonProxyHost プロパティでは、プロキシ接続が有効になっている場合に、コネクタがプロキシサーバーを経由せずにアクセスできるホストを、カンマ区切りリストで指定します。

```
.amazonaws.com,localhost,.example.net,.example.com
```

NonProxyHost 接続パラメータは、curl の CURLOPT_NOPROXY オプションに渡されます。CURLOPT_NOPROXY 形式の詳細については、curl のドキュメントで「CURLOPT_NOPROXY」を参照してください。

ODBC クライアントを使用して Microsoft AD FS ユーザーに Amazon Athena へのフェデレーテッドアクセスを設定する

ODBC クライアントを使用して Microsoft Active Directory フェデレーションサービス (AD FS) ユーザーに Amazon Athena へのフェデレーテッドアクセスをセットアップするには、まず AD FS と AWS アカウントとの間に信頼を確立します。この信頼性が確立されると、AD ユーザーは AD 認証情報を使用して AWS へフェデレーションを行い、AWS Identity and Access Management (IAM) ロールのアクセス許可を引き継ぎ、Athena API などの AWS リソースにアクセスできるようになります。

この信頼を作成するには、SAML プロバイダとして AD FS を AWS アカウント に追加し、フェデレーションユーザーが推定できる IAM ロールを作成します。AD FS 側では、証明書利用者として AWS を追加し、認証用 (具体的には Athena と Amazon S3) に適切なユーザー属性を AWS に送信する SAML クレームルールを作成します。

Athena への AD FS アクセスを設定するには、主に次の手順が必要です。

1. IAM SAML プロバイダとロールのセットアップ

2. AD FS の設定

3. Active Directory ユーザーとグループの作成

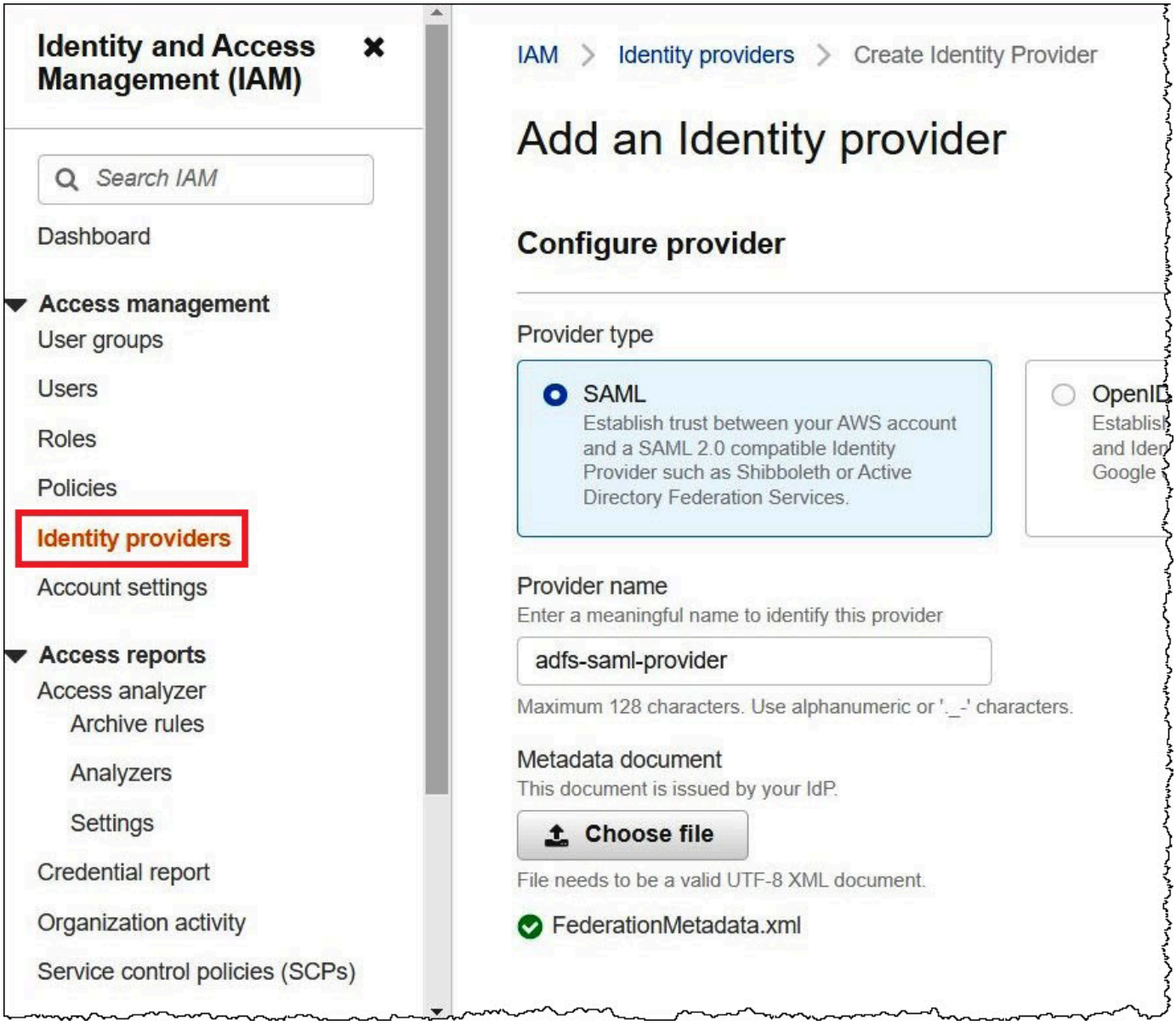
4. Athena への AD FS ODBC 接続を設定する

1. IAM SAML プロバイダとロールのセットアップ

このセクションでは、AD FS を SAML プロバイダとして AWS アカウントに追加し、フェデレーションユーザーが推定できる IAM ロールを作成します。

SAML プロバイダをセットアップするには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[Identity providers (ID プロバイダ)] を選択します。
3. [プロバイダーを追加] をクリックします。
4. [プロバイダのタイプ] では [SAML] を選択します。



5. [プロバイダ名] では **adfs-saml-provider** を入力します。
6. ブラウザで、次のアドレスを入力して、AD FS サーバーのフェデレーション XML ファイルをダウンロードします。この手順を実行するには、ブラウザが AD FS サーバーにアクセスできる必要があります。

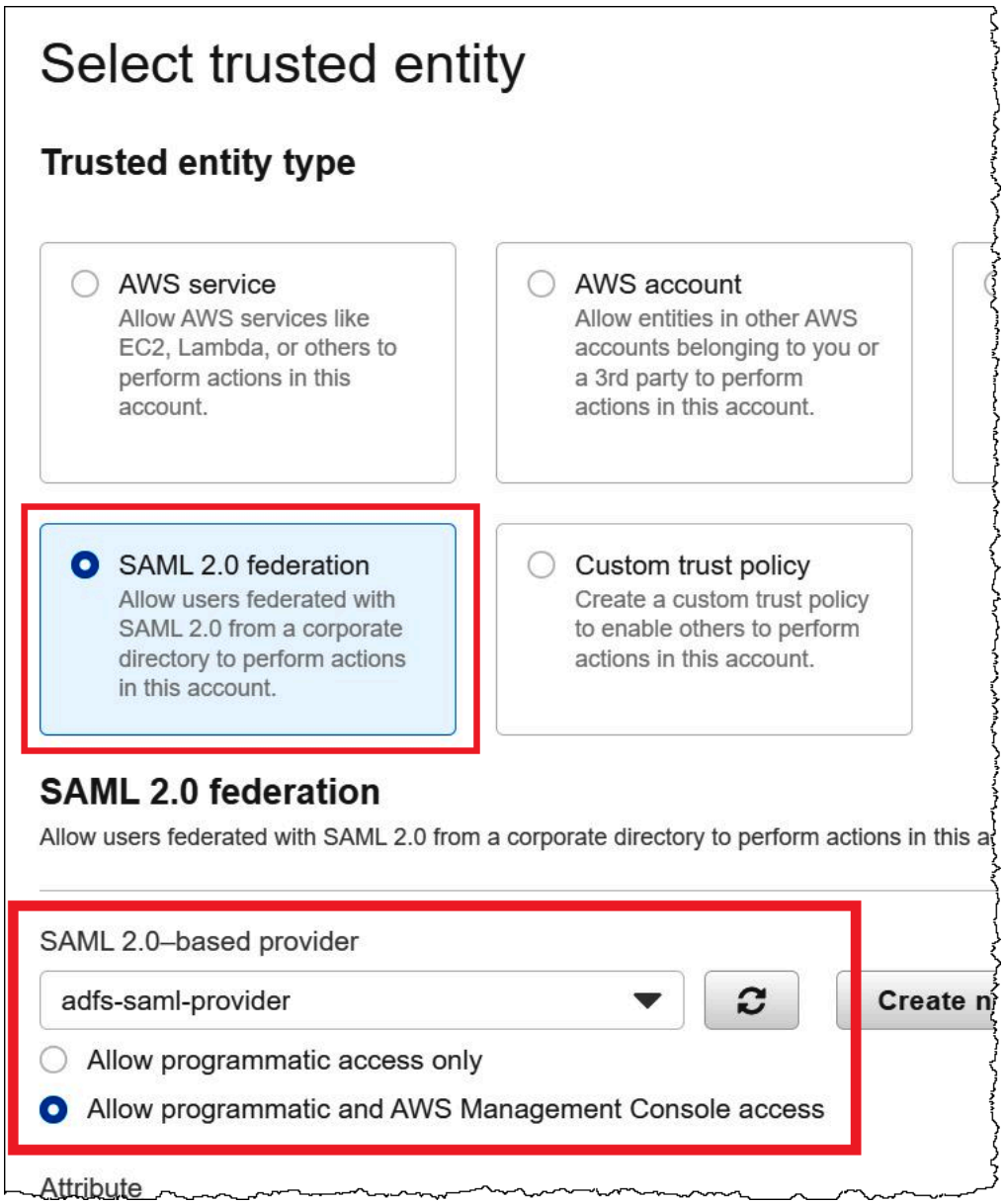
```
https://adfs-server-name/federationmetadata/2007-06/federationmetadata.xml
```

7. IAM コンソールの [Metadata document] (メタデータドキュメント) で、[Choose file] (ファイルを選択) を選択し、フェデレーションメタデータファイルを AWS にアップロードします。
8. 終了するには、[Add provider] (プロバイダを追加) を選択します。

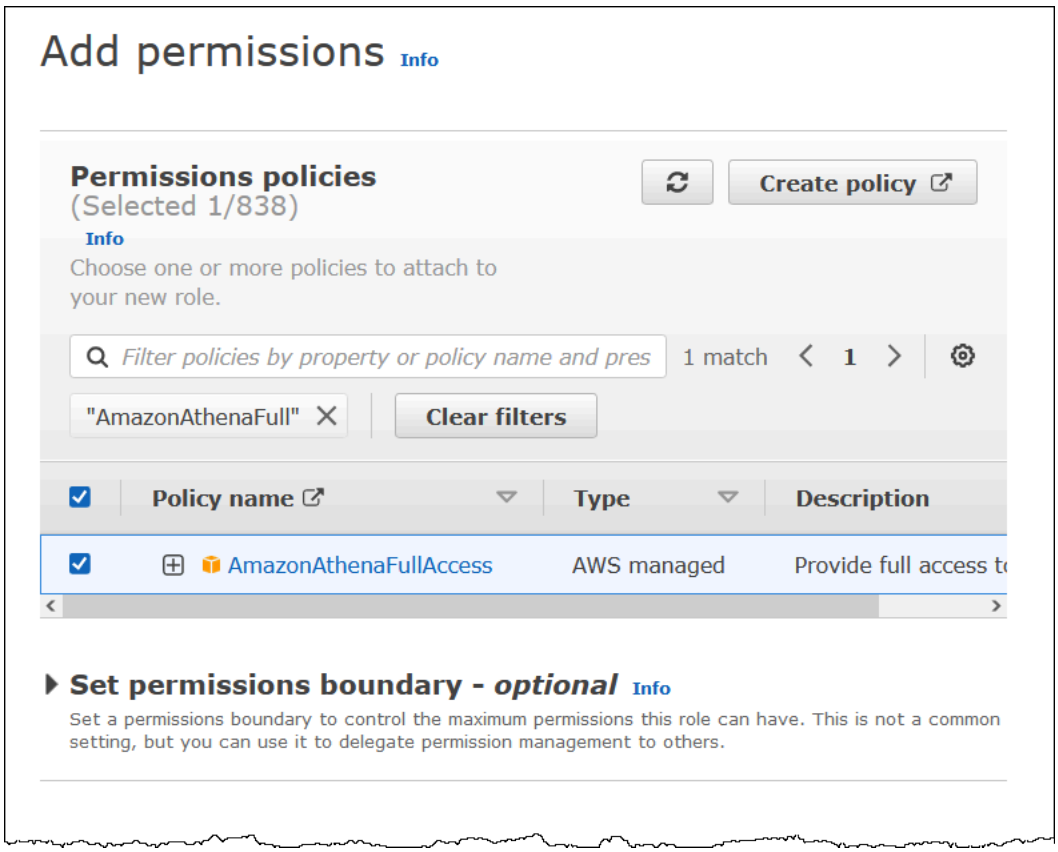
次に、フェデレーションユーザーが設定できる IAM ロールを作成します。

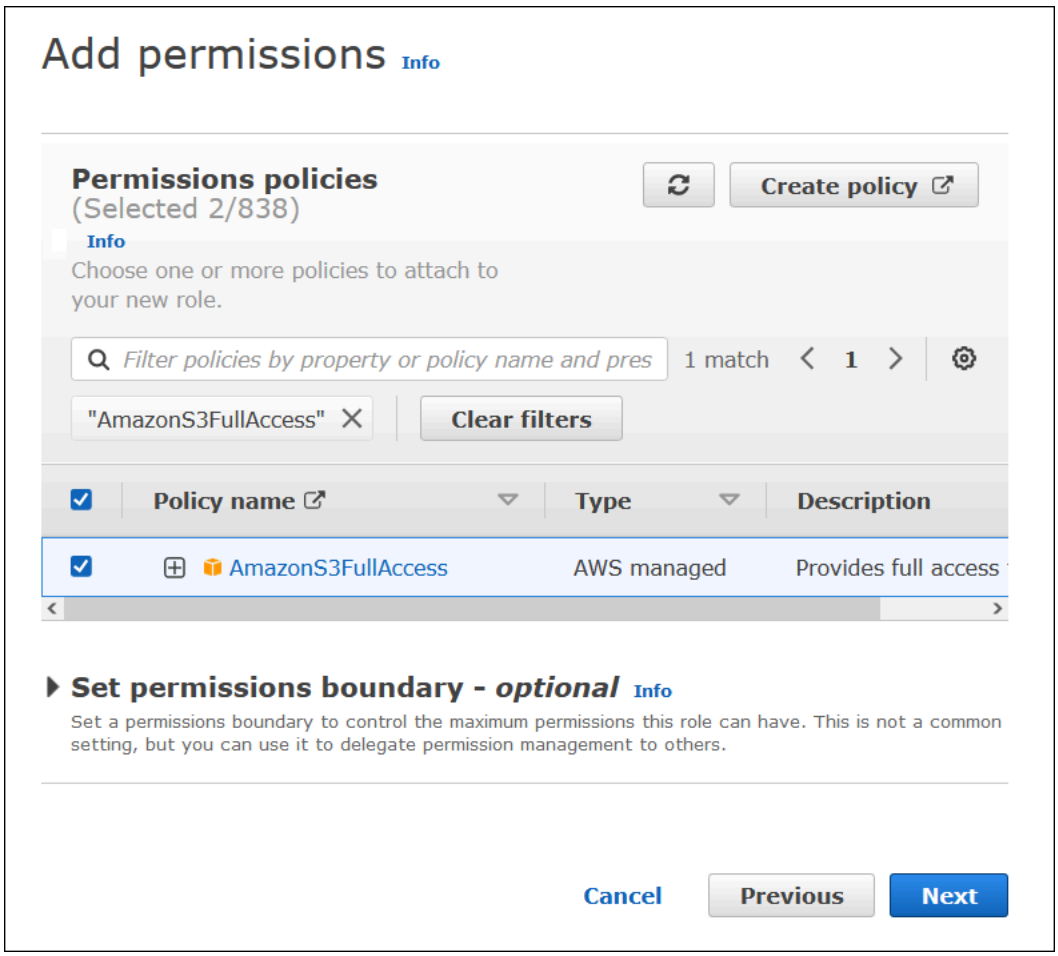
フェデレーションユーザーに IAM ロールを作成するには

1. IAM コンソールのナビゲーションペインで、[Roles] (ロール) を選択します。
2. [ロールの作成] を選択します。
3. [Trusted entity type] (信頼されたエンティティのタイプ) で、[SAML 2.0 Federation] (SAML 2.0 フェデレーション) を選択します。
4. [SAML 2.0-based provider] (SAML 2.0 ベースのプロバイダ) の場合は、作成した [adfs-saml-provider] プロバイダを選択します。
5. [プログラムと AWS 管理コンソールへのアクセスを許可する] 〉 [次へ] の順に選択します。



6. [Add permissions] (アクセス許可の追加) ページで、このロールに必要な IAM のアクセス許可ポリシーをフィルタリングし、対応するチェックボックスを選択します。このチュートリアルでは、`AmazonAthenaFullAccess` および `AmazonS3FullAccess` ポリシーをアタッチします。


Add permissions Info
Permissions policies
(Selected 1/838)
Info
Choose one or more policies to attach to your new role.
Create policy
Filter policies by property or policy name and pres
1 match
1
"AmazonAthenaFull"
Clear filters
Policy name
Type
Description
AmazonAthenaFullAccess
AWS managed
Provide full access t
Set permissions boundary - optional Info
Set a permissions boundary to control the maximum permissions this role can have. This is not a common setting, but you can use it to delegate permission management to others.

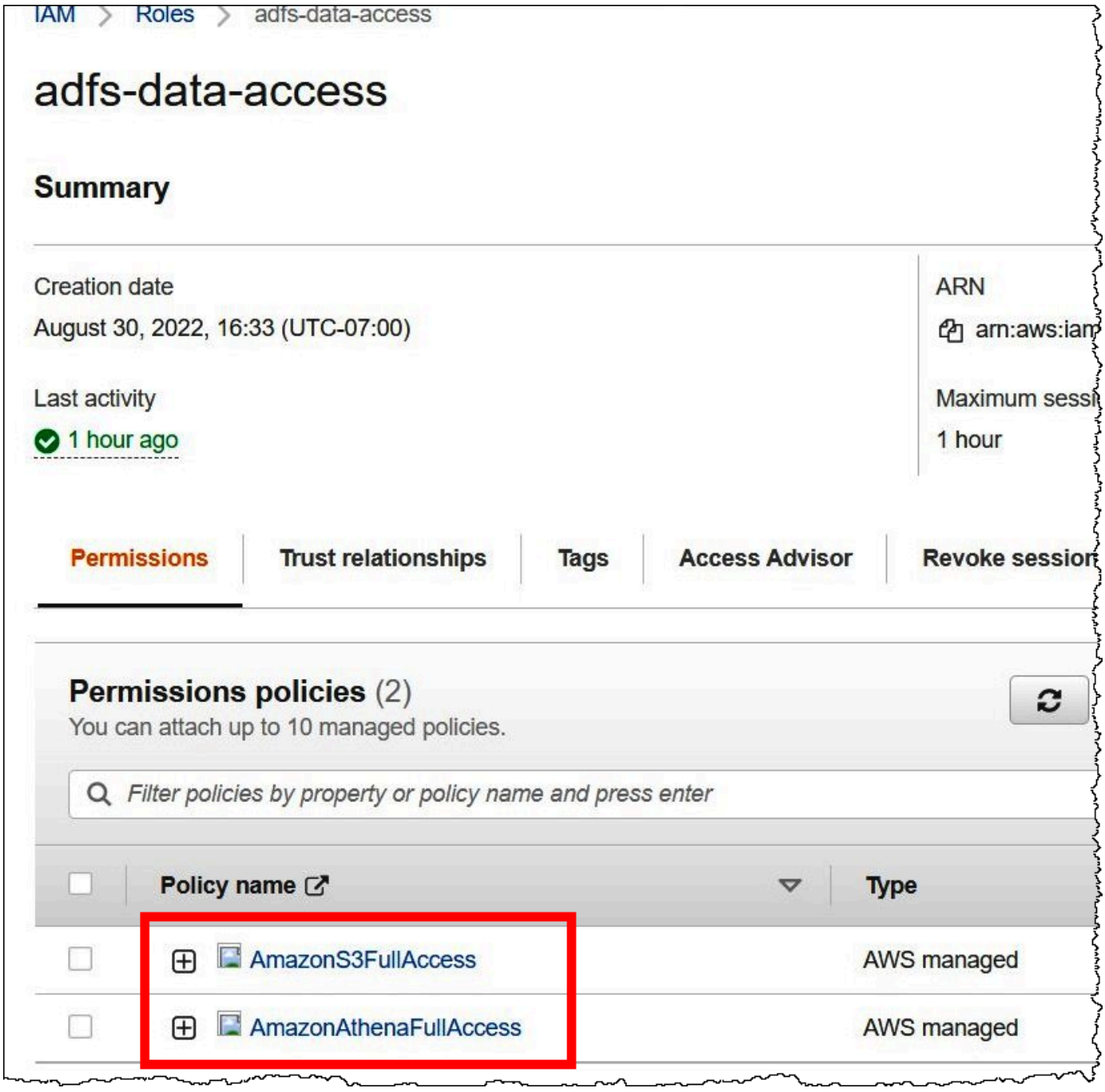


7. [Next] を選択します。
8. [Name, review, and create] (名前、確認、および作成) ページの [Role name] (ロール名) に、ロールの名前を入力します。このチュートリアルでは、[adfs-data-access] という名前を使用します。

   [Step 1: Select trusted entities] (ステップ 1: 信頼できるエンティティを選択する) では、[Principal] (プリンシパル) フィールドは自動的に `"Federated:" "arn:aws:iam::account_id:saml-provider/adfs-saml-provider"` に設定されます。`Condition` フィールドには `"SAML:aud"` と `"https://signin.aws.amazon.com/saml"` を含める必要があります。

[Step 2: Add permissions] (ステップ 2: アクセス許可の追加) には、ロールにアタッチしたポリシーが表示されます。

9. [ロールの作成] を選択します。ロールの作成を確認するバナーメッセージが表示されます。
10. [Roles] (ロール) ページで、作成したロールを選択します。ロールの概要ページには、アタッチされているポリシーが表示されます。



2. AD FS の設定

これで、証明書利用者として AWS を追加して SAML クレームルールを作成する準備が整い、認証用に適切なユーザー属性を AWS に送信できるようになります。

SAML ベースのフェデレーションには、IdP (Active Directory) と、IdP からの認証を使用するサービスまたはアプリケーションである証明書利用者 (AWS) の 2 人の参加者がいます。

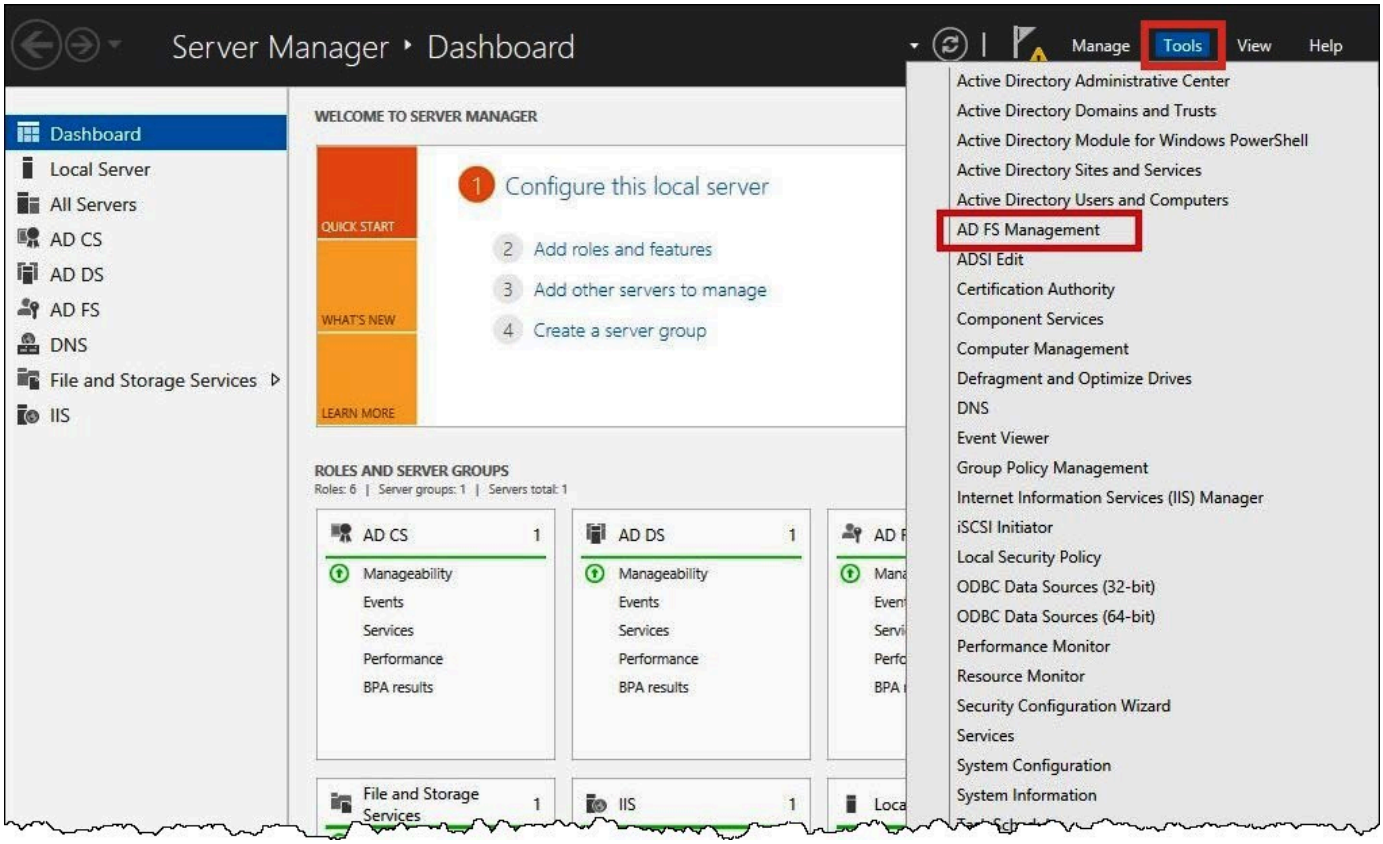
AD FS を設定するには、まず証明書利用者の信頼を追加し、次に証明書利用者の SAML クレームルールを設定します。AD FS はクレームルールを使用して SAML アサーションを作成し、証明書利用者に送信します。SAML アサーションには、AD ユーザーに関する情報が真実であり、ユーザーが認証されていることが記載されています。

証明書利用者の信頼を追加する

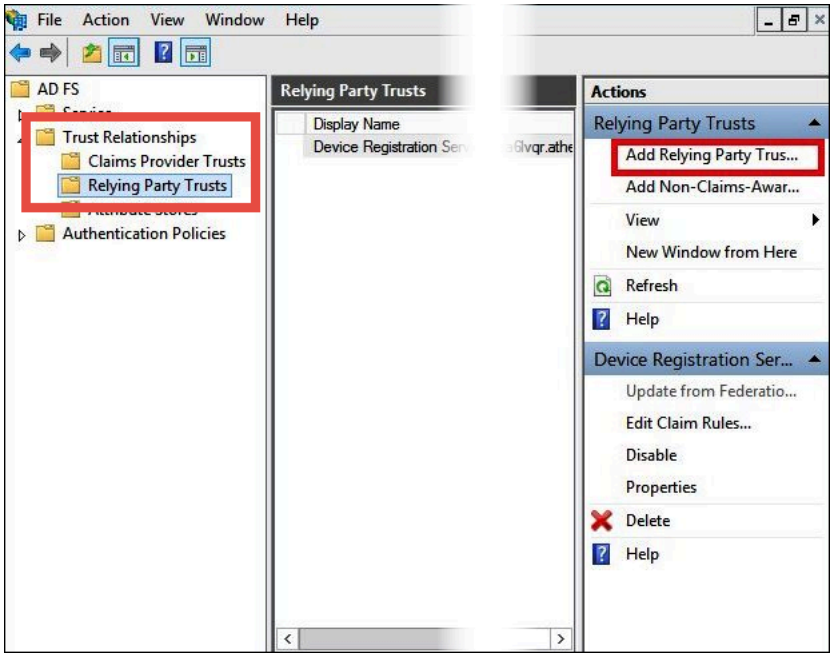
AD FS に証明書利用者の信頼を追加するには、AD FS サーバーマネージャーを使用します。

AD FS で証明書利用者の信頼を追加する

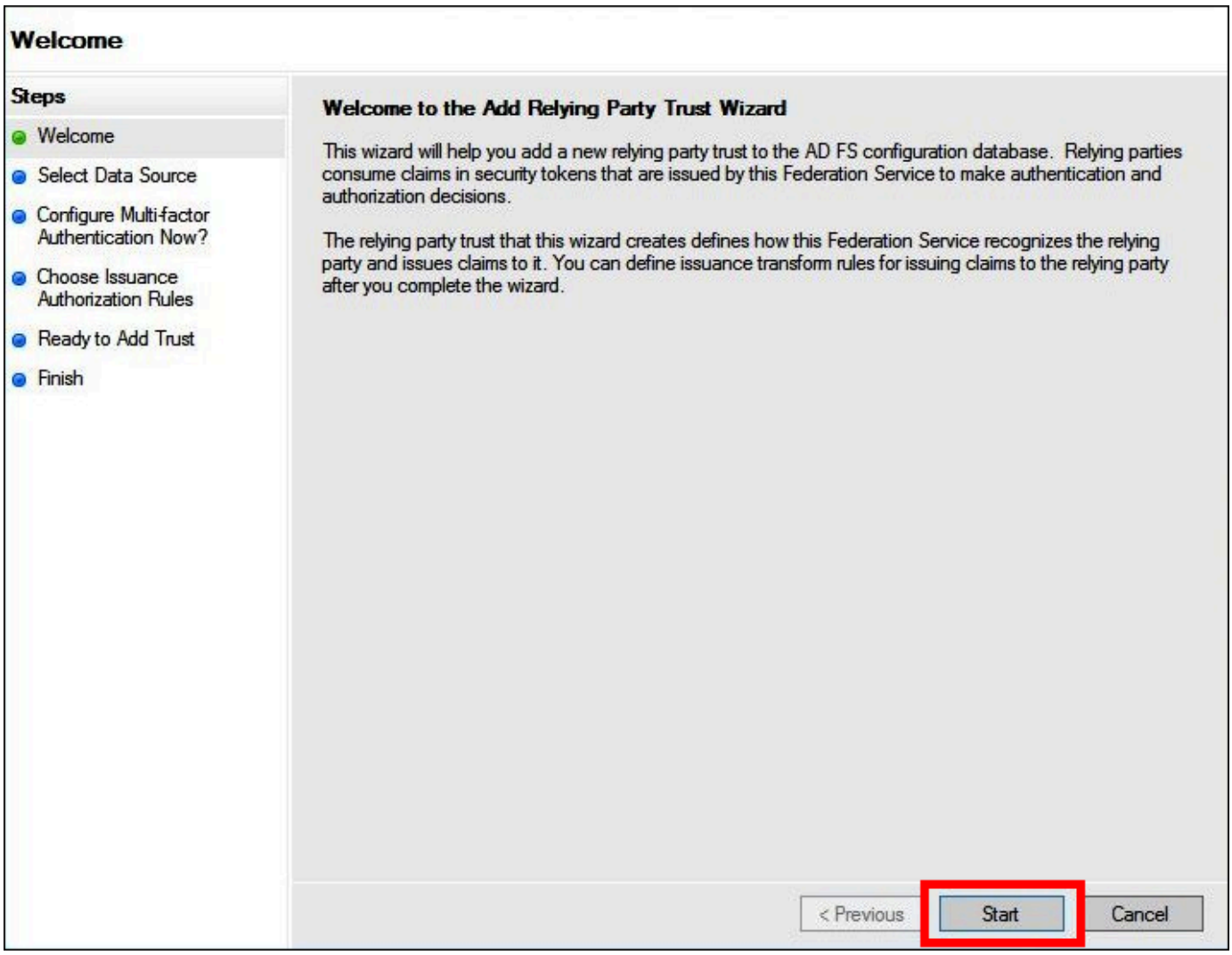
1. AD FS サーバーにサインインします。
2. [Start] (スタート) メニューで、[Server Manager] (サーバーマネージャー) を開きます。
3. [Tools] (ツール) を選択し、[AD FS Management] (AD FS 管理) を選択します。



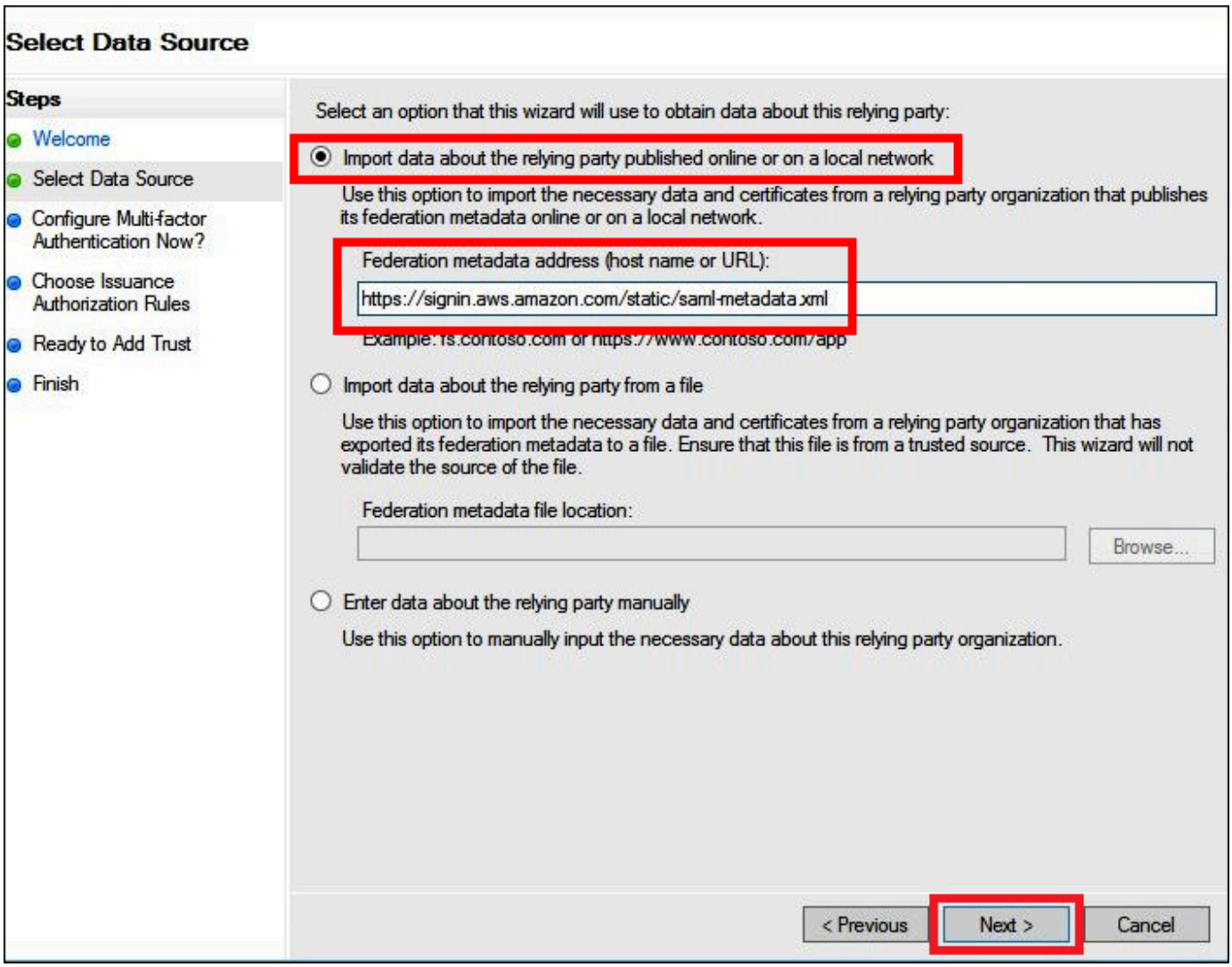
4. ナビゲーションペインの [Trust Relationships] (信頼関係) で、[Relying Party Trusts] (証明書利用者の信頼) を選択します。
5. [Actions] (アクション) で、[Add Relying Party Trust] (証明書利用者の信頼を追加) を選択します。



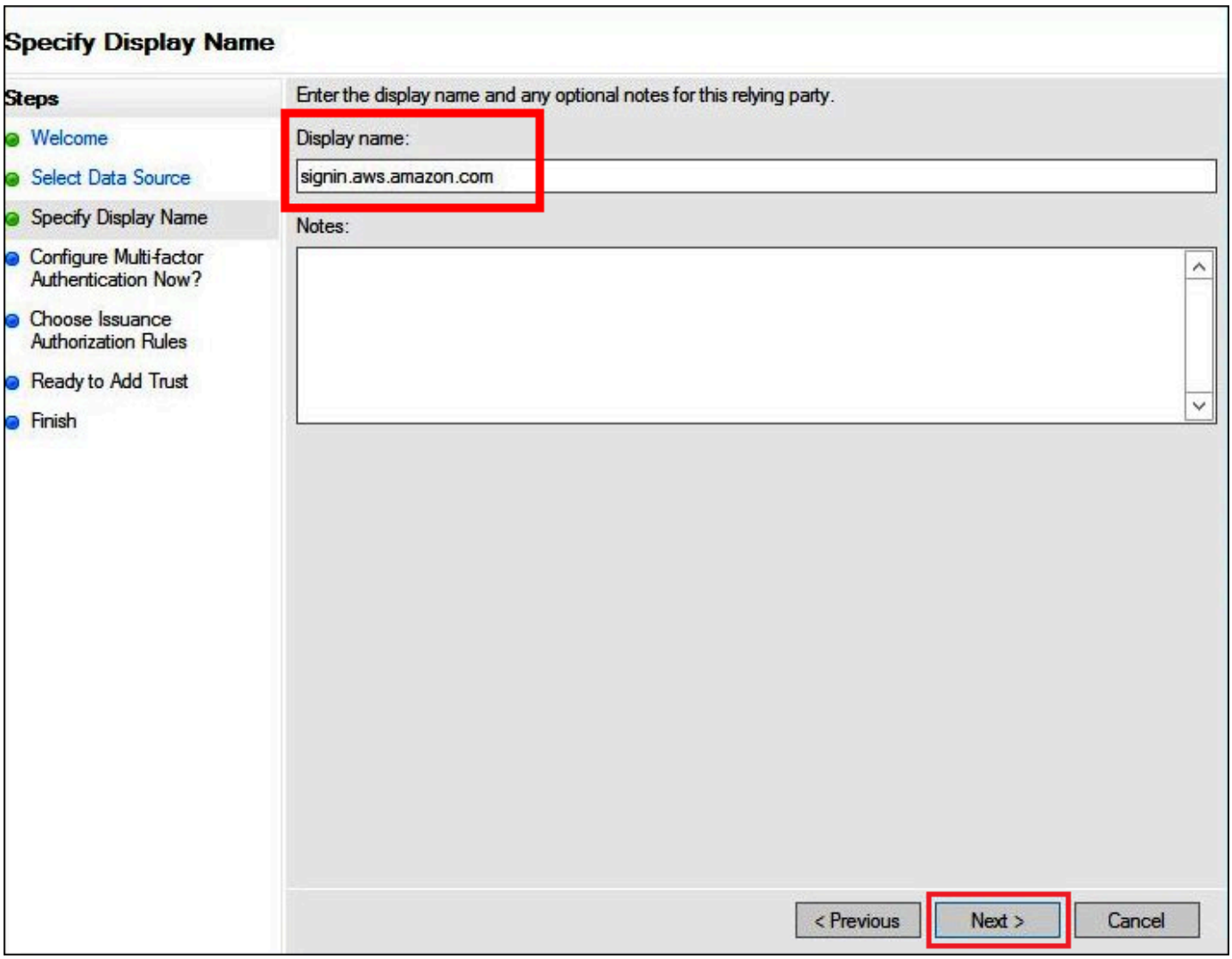
6. 証明書利用者信頼の追加ウィザード ページで、開始を選択します。



7. [Select Data Source] (データソースを選択) ページで、[Import data about the relying party published online or on a local network] (オンラインまたはローカル ネットワークで公開されている証明書利用者に関するデータをインポート) を選択します。
8. [Federation metadata address (host name or URL)] (フェデレーションメタデータアドレス (ホスト名または URL)) に、URL **`https://signin.aws.amazon.com/static/saml-metadata.xml`** を入力します。
9. [Next] を選択します。

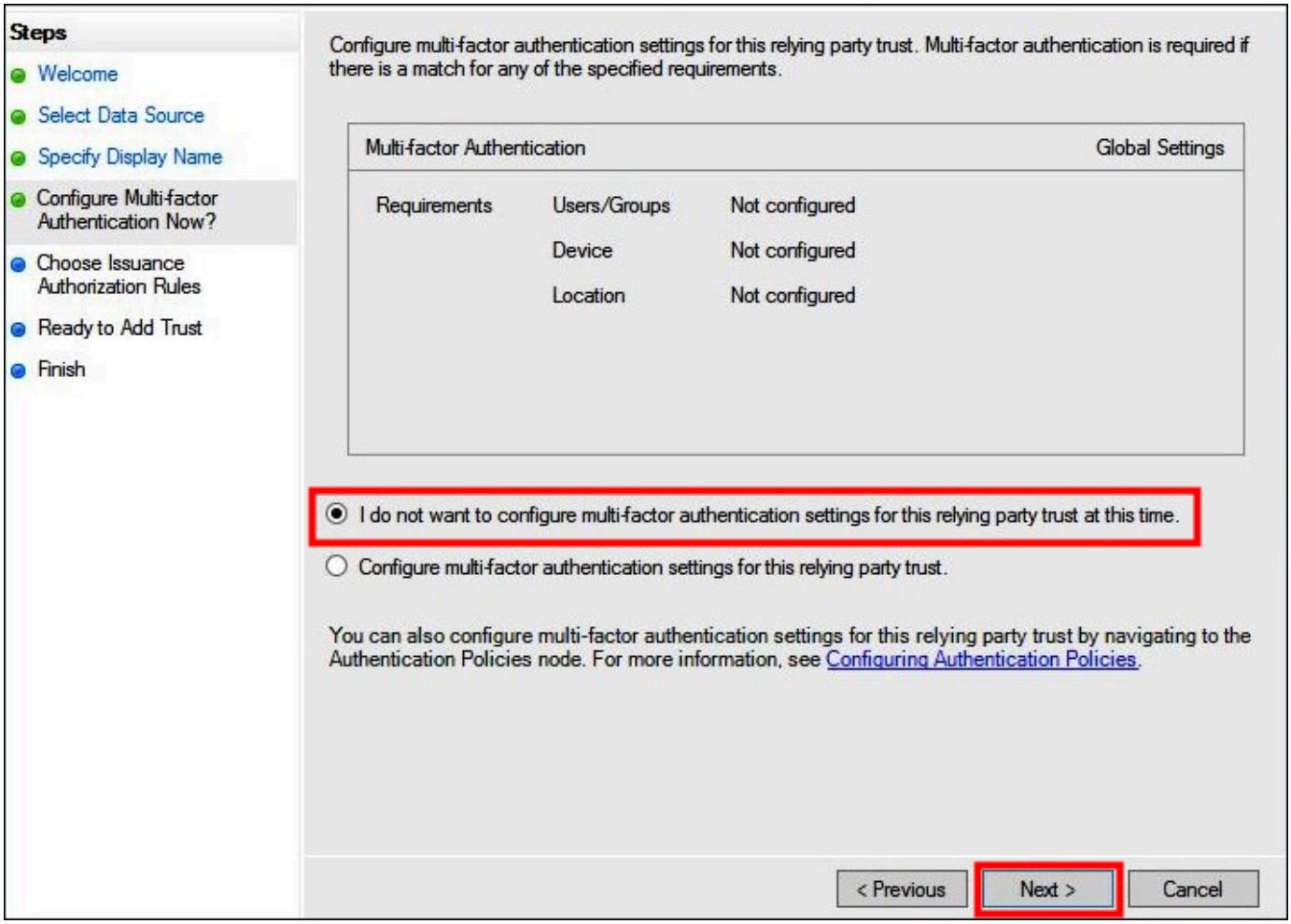


10. [Specify Display Name] (表示名の指定) ページの [Display name] (表示名) に、証明書利用者の表示名を入力し、[Next] (次へ) を選択します。



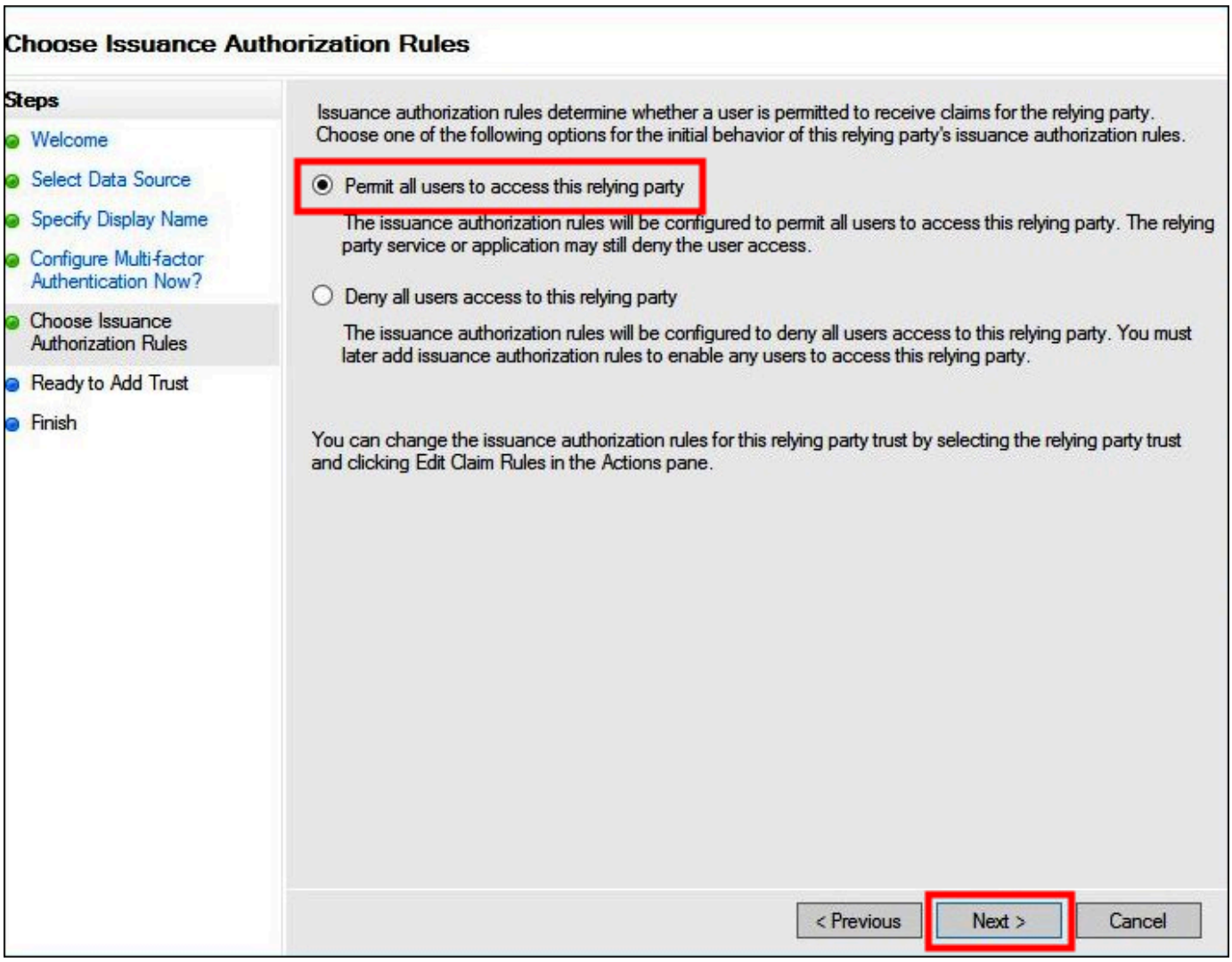
11. このチュートリアルでは、[Configure Multi-factor Authentication Now] (多要素認証を今すぐ設定) ページで、[I do not want to configure multi-factor authentication for this relying party trust at this time] (現時点ではこの依存パーティの信頼に多要素認証を設定しない) を選択します。

    セキュリティを向上させるには、多要素認証を設定して AWS リソースを保護することを推奨します。サンプルデータセットを使用しているため、このチュートリアルでは多要素認証を有効にしていません。

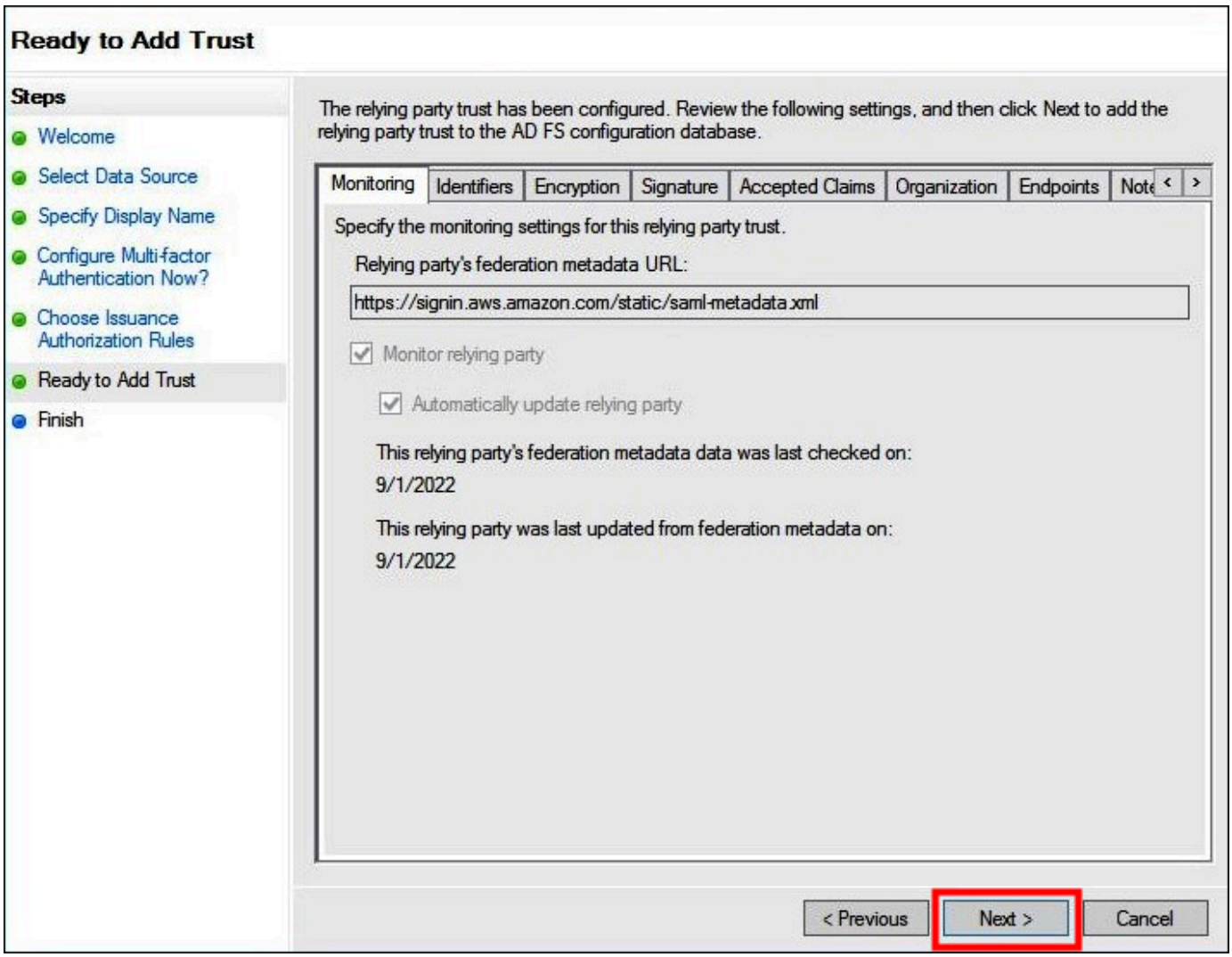


12. [Next] を選択します。
13. [Choose Issuance Authorization Rules] (発行承認ルールの選択) ページで、[Permit all users to access this relying party] (この証明書利用者へのアクセスをすべてのユーザーに許可) を選択します。

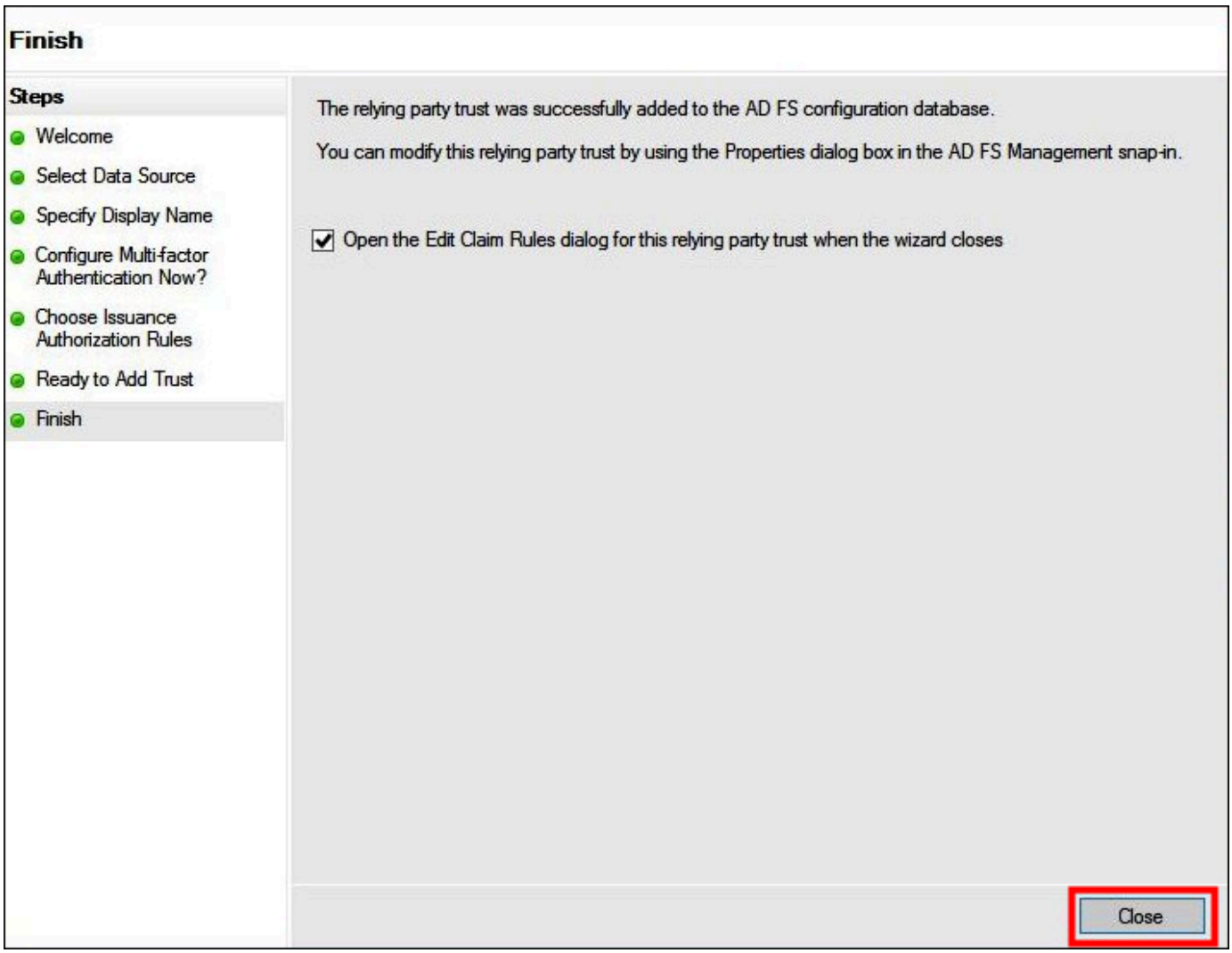
    このオプションを使用すると、AWS で Active Directory 内のすべてのユーザーが AD FS を証明書利用者として使用できます。セキュリティ要件を考慮して、この設定を適宜調整します。



14. [Next] を選択します。
15. [Ready to Add Trust] (信頼を追加する準備完了) ページで、[Next] (次へ) を選択して証明書利用者の信頼を AD FS 設定データベースに追加します。



16. [Finish] (終了) ページで、[Close] (閉じる) を選択します。



証明書利用者向け SAML クレームルールの設定

このタスクでは、クレームルールを 2 セット作成します。

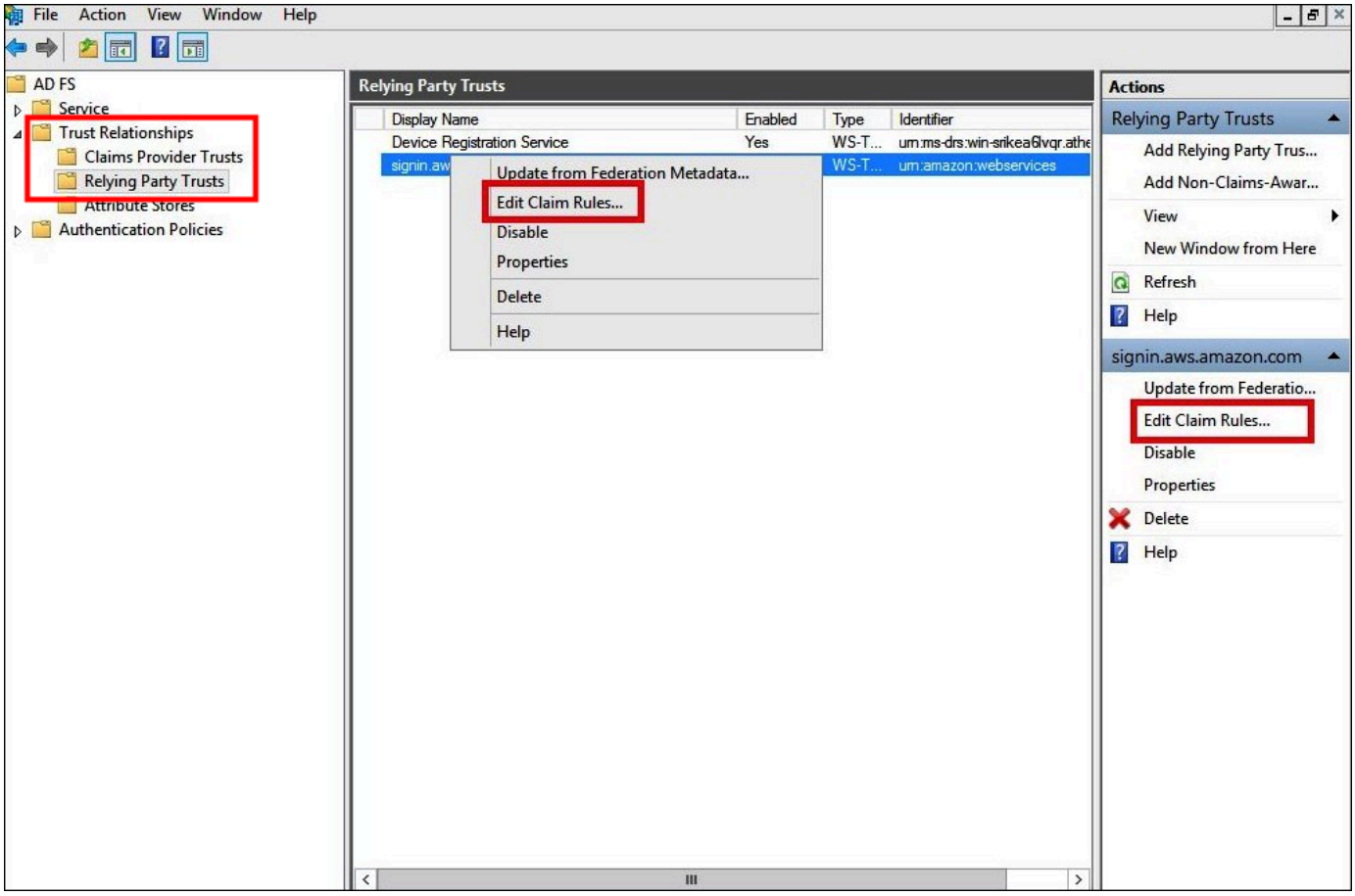
最初のセットであるルール 1 ~ 4 には、AD グループメンバーシップに基づいて IAM ロールを設定するために必要な AD FS クレームルールが含まれています。これらは、AWS Management Console へのフェデレーションアクセスを確立する場合に作成するのと同じルールです。

2 番目のセットであるルール 5 ~ 6 は、Athena のアクセス制御に必要なクレームルールです。

AD FS クレームルールを作成するには

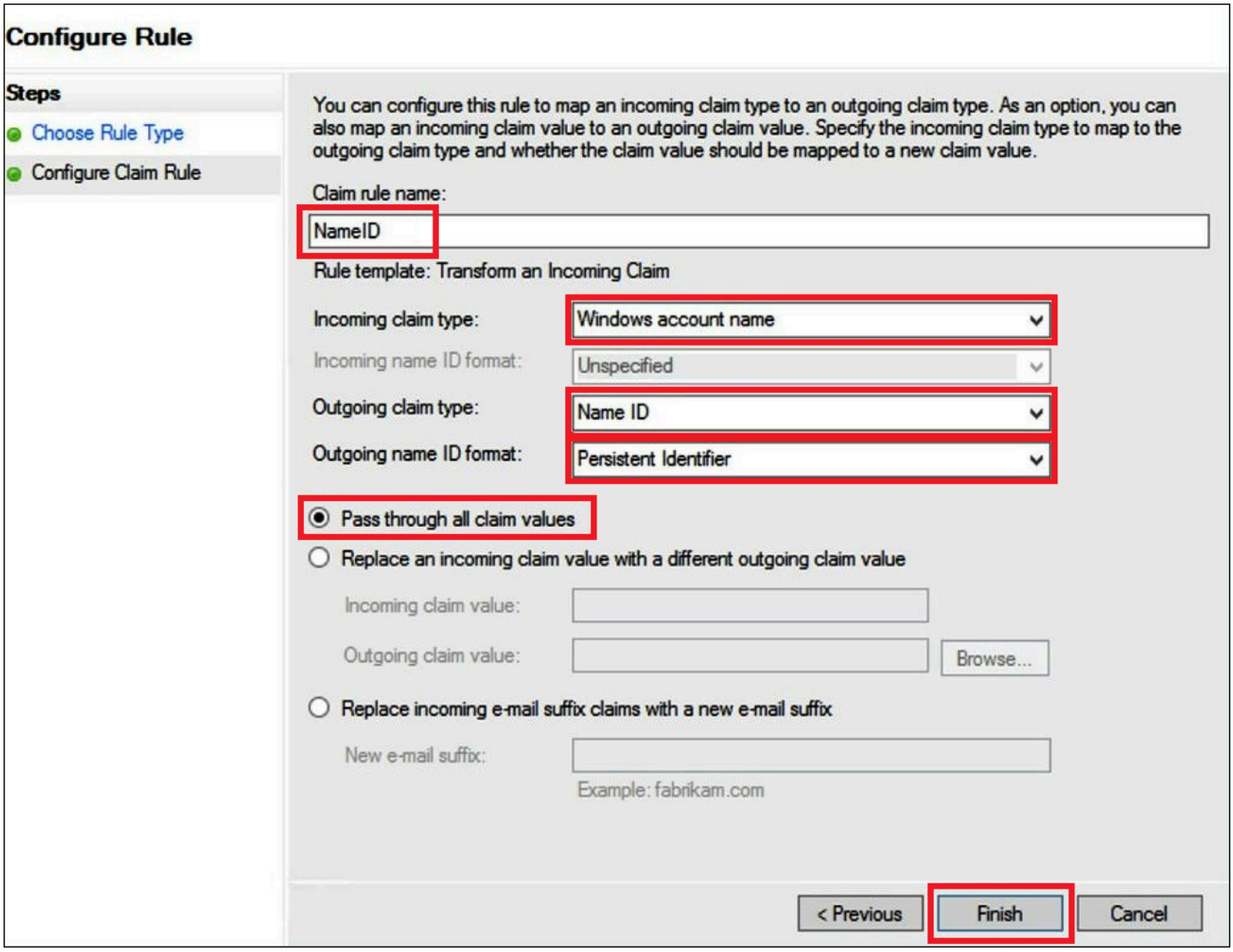
1. AD FS 管理コンソールのナビゲーションペインで、[Trust Relationships] (信頼関係)、[Relying Party Trusts] (証明書利用者の信頼) を選択します。

2. 以前のセクションで作成した証明書利用者を検索します。
3. 証明書利用者を右クリックして [Edit Claim Rules] (クレームルールを編集) を選択するか、[Actions] (アクション) メニューから [Edit Claim Rules] (クレームルールを編集) を選択します。



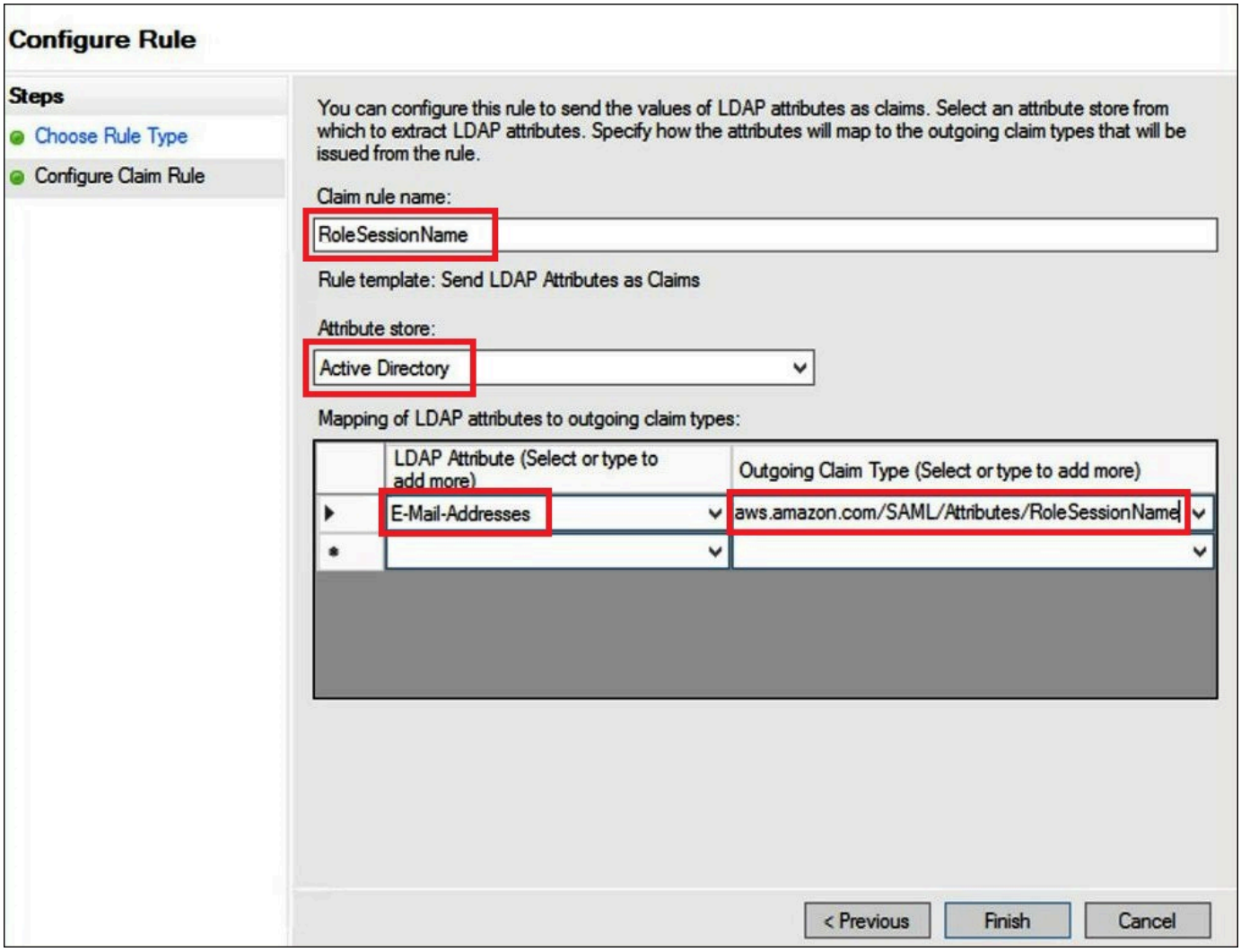
4. ルールの追加] を選択します。
5. 変換要求規則の追加ウィザードの [Configure Rule] (ルールの設定) ページで、次の情報を入力してクレームルール 1 を作成し、[Finish] (完了) を選択します。

   - [Claim Rule name] (クレームルール名) に **`NameID`** を入力します。
   - [Rule template] (ルールテンプレート) には、[Transform an Incoming Claim] (受信クレームの変換) を使用してください。
   - [Incoming claim type] (受信クレーム型) で、[Windows account name] (Windows アカウント名) を選択します。
   - [Outgoing claim type] (発信クレーム型) で、[Name ID] (名前 ID) を選択します。
   - 発信者名 ID フォームで、永続的識別子を選択します。
   - [Pass through all claim values] (すべてのクレーム値を通過) を選択します。



6. [Add Rule] (ルールを追加) を選択し、次の情報を入力してクレームルール 2 を作成し、[Finish] (完了) を選択します。

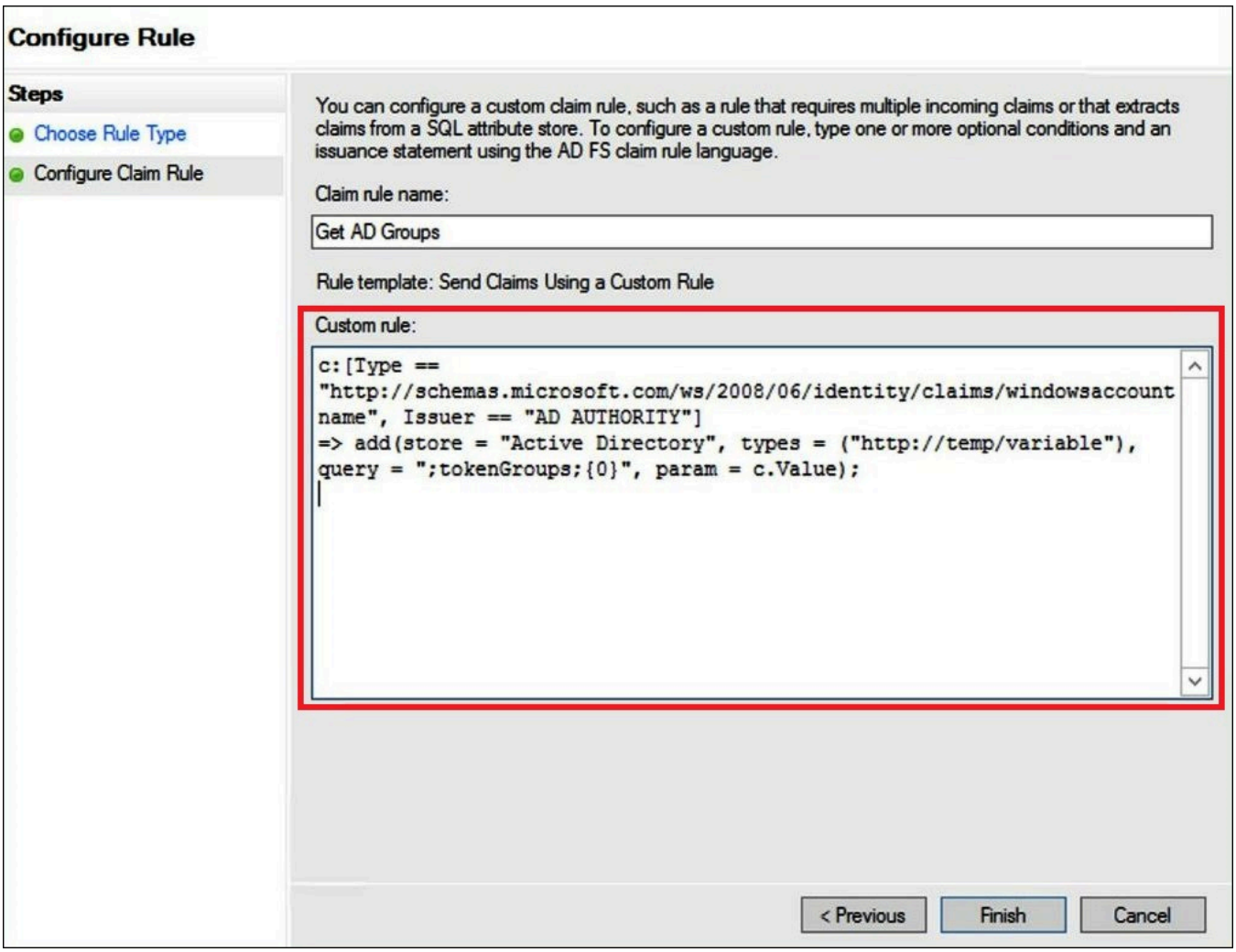
   - [Claim rule name] (クレームルール名) に **`RoleSessionName`** を入力します。
   - [Rule template] (ルールテンプレート) には、[Send LDAP Attribute as Claims] (LDAP 属性をクレームとして送信) を使用してください。
   - 属性を保存するで アクティブディレクトリを選択します。
   - [Mapping of LDAP attributes to outgoing claim types] (LDAP 属性を送信クレームタイプにマッピング) には、属性 **`E-Mail-Addresses`** を追加します。[Outgoing Claim Type] (送信クレーム型) には、 **`https://aws.amazon.com/SAML/Attributes/RoleSessionName`** を入力します。



7. [Add Rule] (ルールを追加) を選択し、次の情報を入力してクレームルール 3 を作成し、[Finish] (完了) を選択します。

   - [Claim rule name] (クレームルール名) に **`Get AD Groups`** を入力します。
   - [Rule template] (ルールテンプレート) には、[Send Claims Using a Custom Rule] (カスタムルールを使用してクレームを送信) を使用してください。
   - [Custom rule] (カスタムルール) には、次のように入力します。

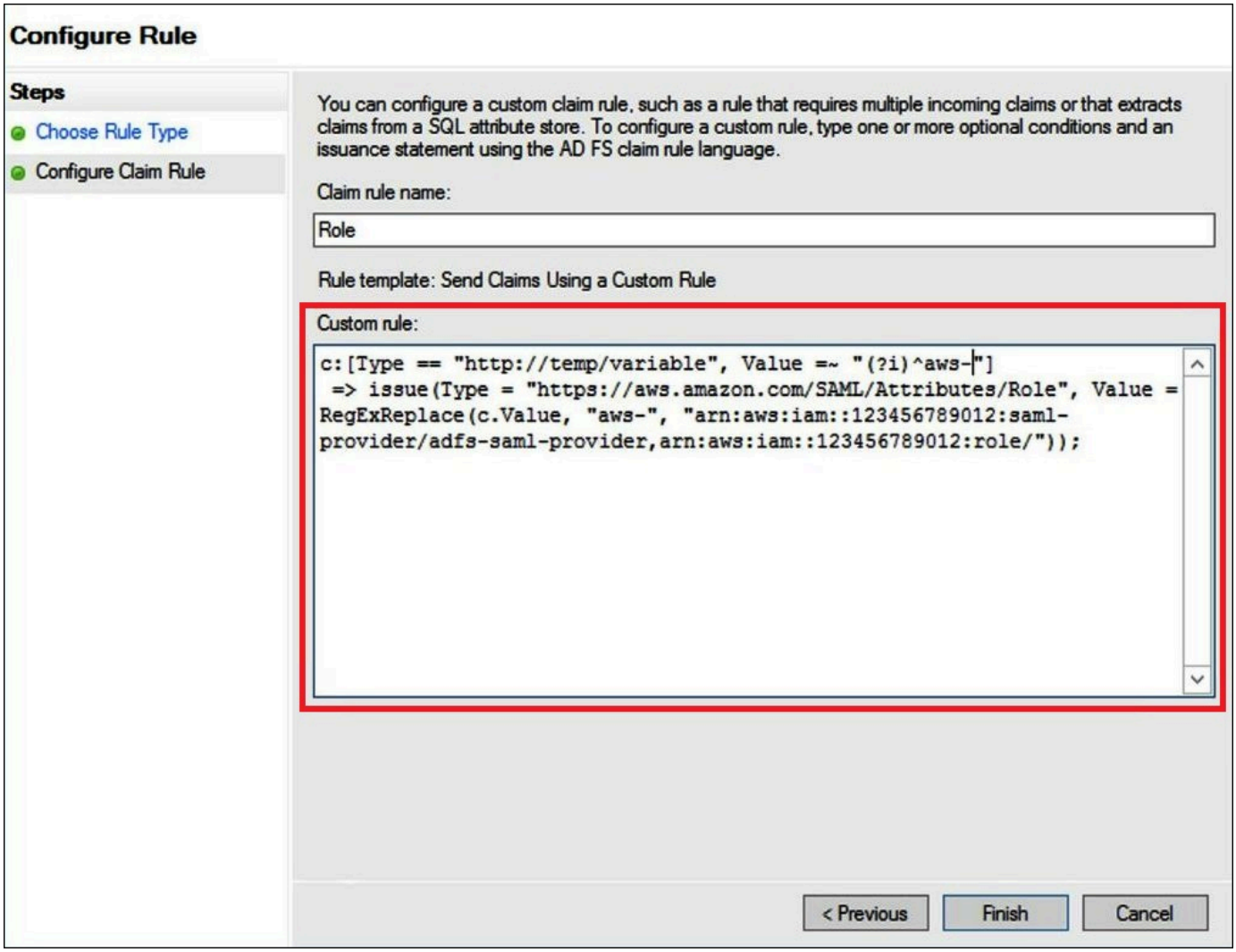
```
c:[Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/
windowsaccountname",
 Issuer == "AD AUTHORITY"]=> add(store = "Active Directory", types = ("http://
temp/variable"),
 query = ";tokenGroups;{0}", param = c.Value);
```



8. ルールの追加] を選択します。次の情報を入力してクレームルール 4 を作成し、[Finish] (完了) を選択します。

   - [Claim rule name] (クレームルール名) に **`Role`** を入力します。
   - [Rule template] (ルールテンプレート) には、[Send Claims Using a Custom Rule] (カスタムルールを使用してクレームを送信) を使用してください。
   - [Custom rule] (カスタムルール) には、アカウント番号と以前に作成した SAML プロバイダの名前を含む次のコードを入力します。

```
c:[Type == "http://temp/variable", Value =~ "(?i)^aws-"]=> issue(Type = "https://
aws.amazon.com/SAML/Attributes/Role",
Value = RegExReplace(c.Value, "aws-", "arn:aws:iam::AWS_ACCOUNT_NUMBER:saml-
provider/adfs-saml-provider,arn:aws:iam:: AWS_ACCOUNT_NUMBER:role/"));
```

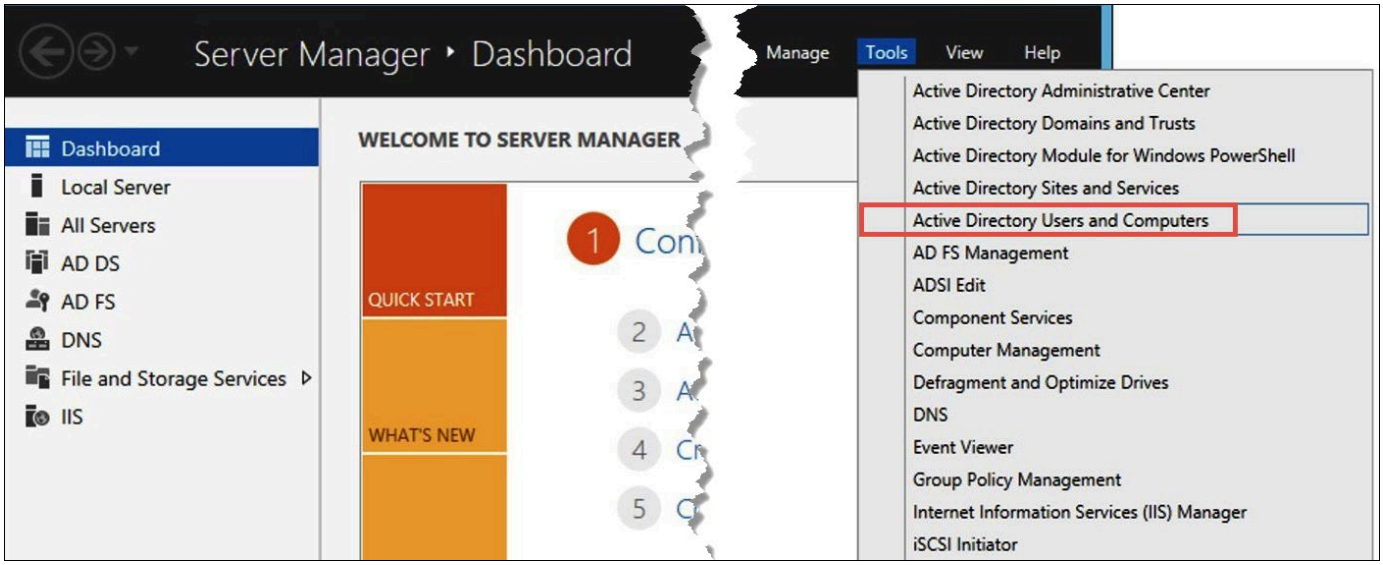


3. Active Directory ユーザーとグループの作成

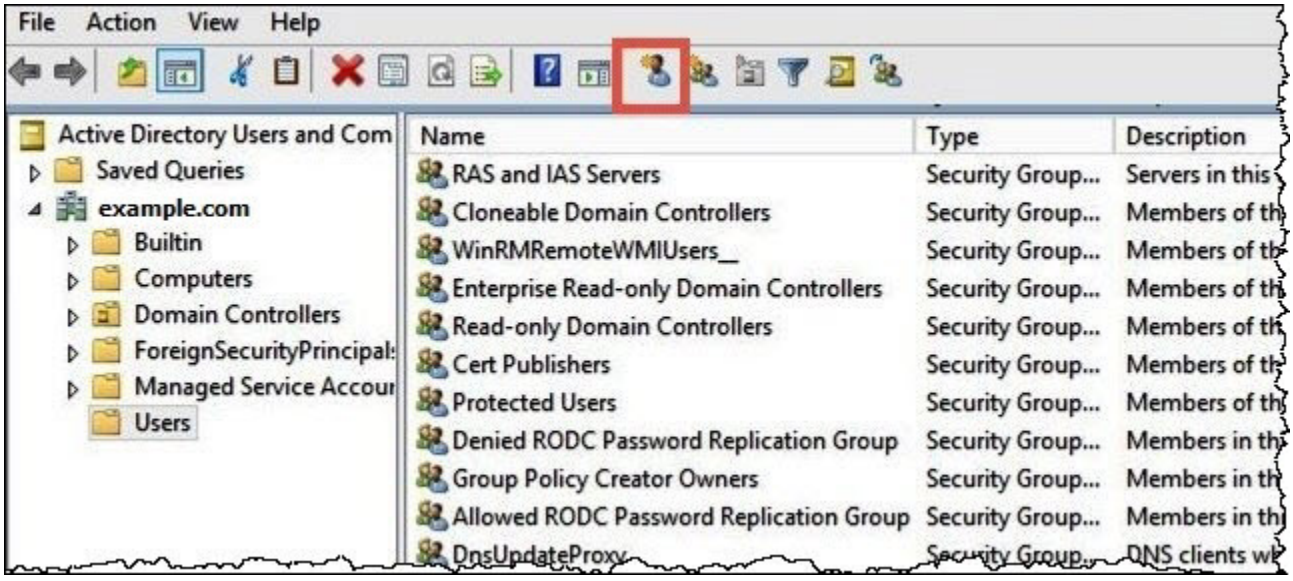
これで、Athena にアクセスする AD ユーザーと、そのユーザーを配置する AD グループを作成して、グループごとにアクセスレベルを制御できる準備ができました。データアクセスのパターンを分類する AD グループを作成後、ユーザーをそれらのグループに追加します。

Athena へアクセスするための AD ユーザーを作成するには

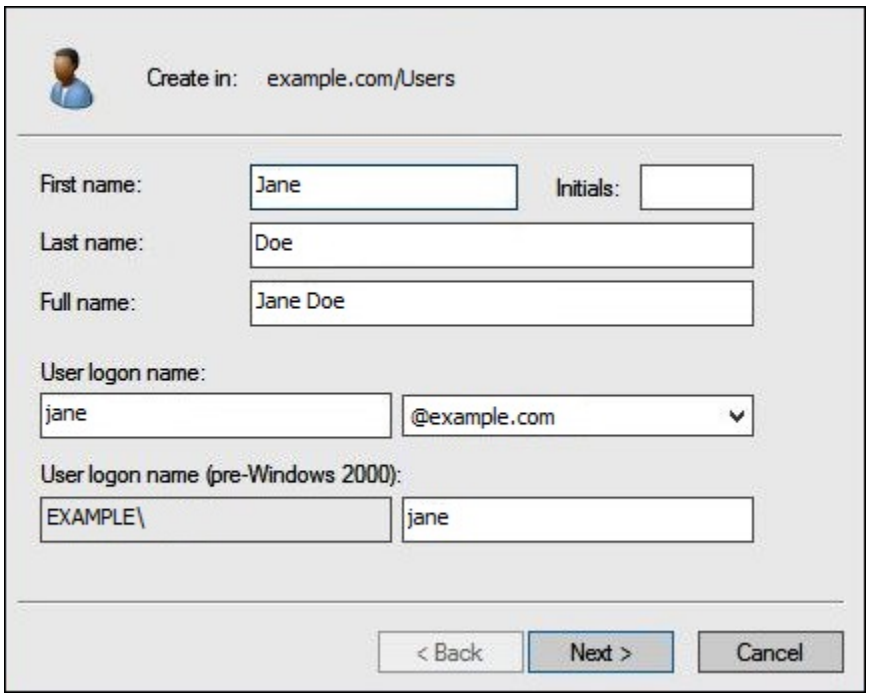
1. サーバーマネージャーダッシュボードで、[Tools] (ツール) を選択し、次に [Active Directory Users and Computers] (Active Directory ユーザーとコンピュータ) を選択します。



2. ナビゲーションペインで [Users (ユーザー)] を選択します。
3. [Active Directory Users and Computers] (Active Directory ユーザーとコンピュータ) のツールバーで、[Create user] (ユーザーの作成) オプションを選択します。

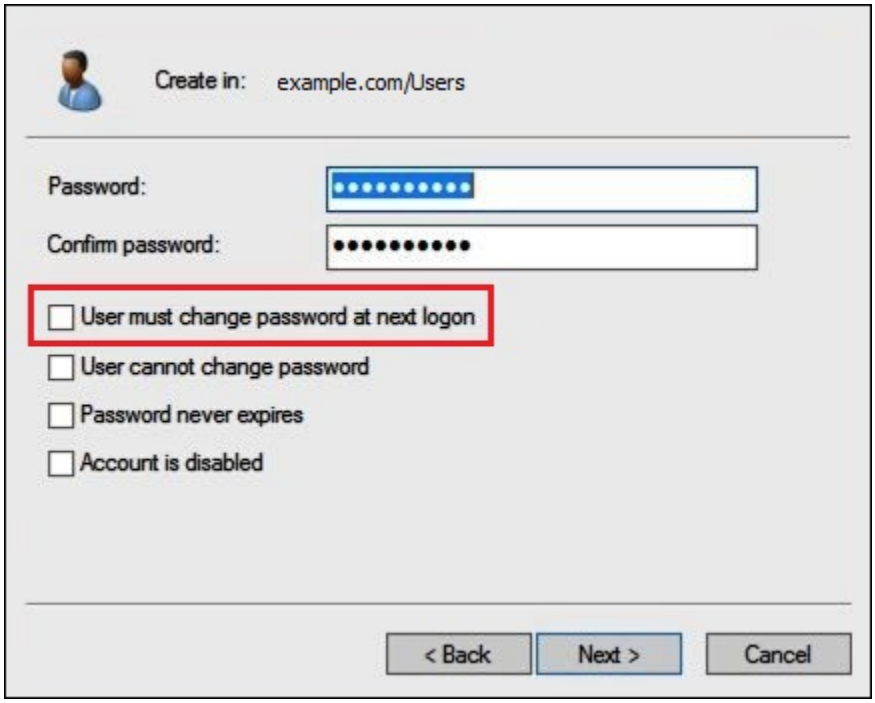


4. [New Object - User] (新規オブジェクト - ユーザー) ダイアログボックスの [First name] (名)、[Last name] (姓)、[Full name] (フルネーム) には名前を入力します。このチュートリアルでは、**`Jane Doe`** を使用します。

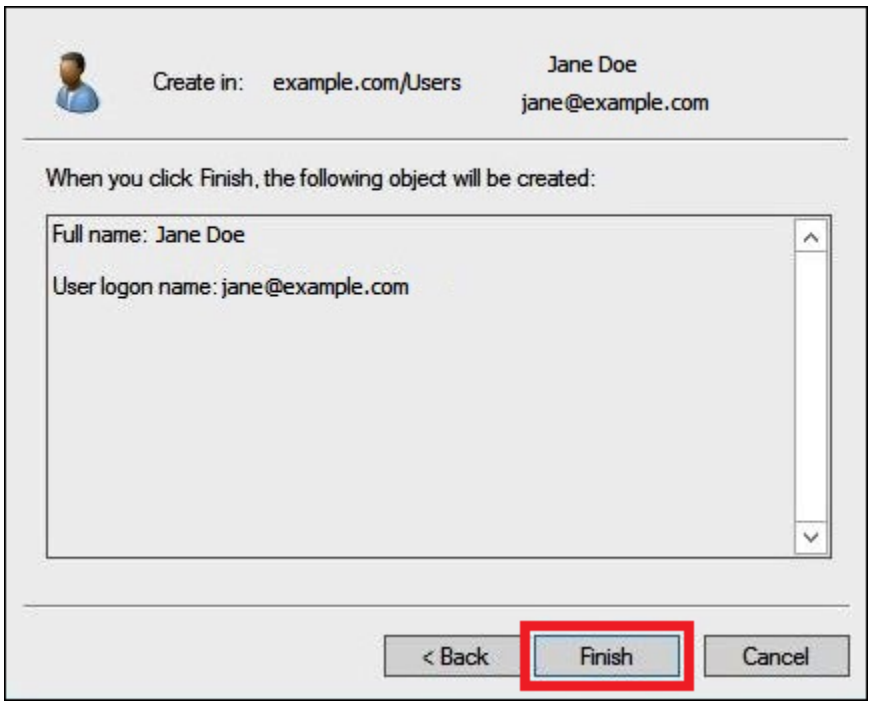


5. [Next] を選択します。
6. [Password] (パスワード) には、パスワードを入力し、確認のため再入力します。

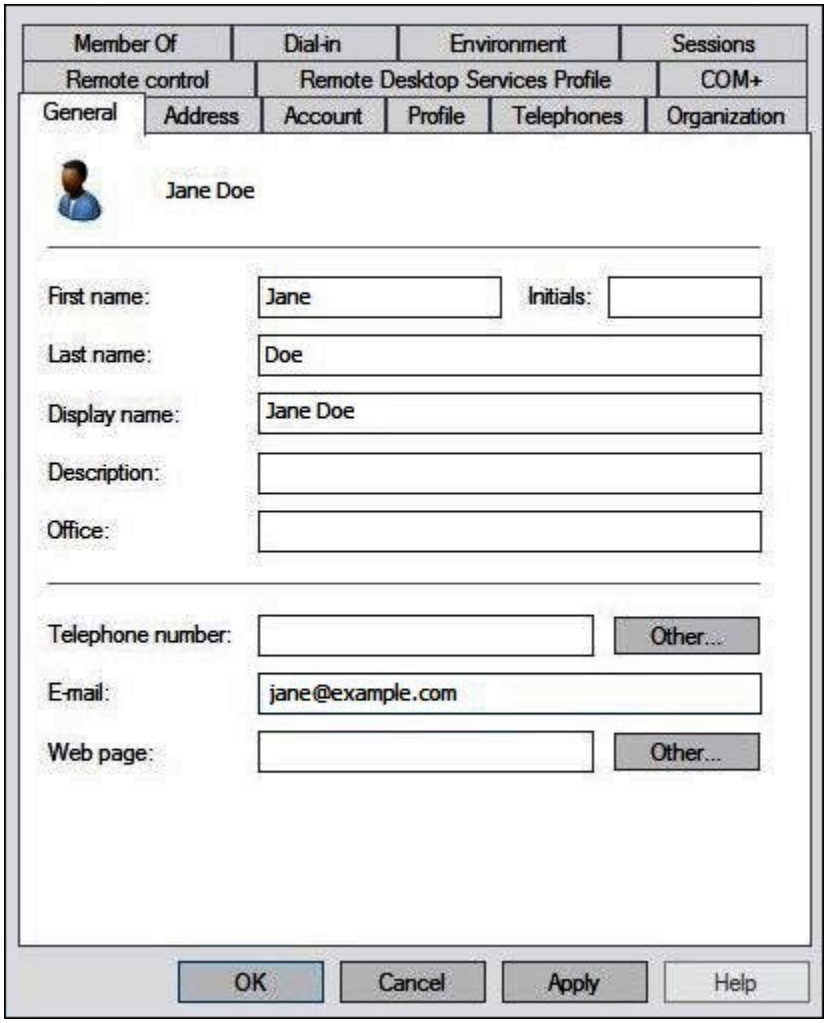
   わかりやすくするために、このチュートリアルでは、[User must change password at next sign on] (ユーザーは次のサインオン時にパスワードを変更する必要がある) を選択解除しています。実際のシナリオでは、新しく作成したユーザーにはパスワードの変更を要求する必要があります。



7. [Next] を選択します。
8. [Finish] を選択します。



9. [Active Directory Users and Computers] (Active Directory ユーザーとコンピュータ) で、[user name] (ユーザー名) を選択します。
10. ユーザーの [Properties] (プロパティ) ダイアログボックスの [E-mail] (メール) に、メールアドレスを入力します。このチュートリアルでは、**`jane@example.com`** を使用します。



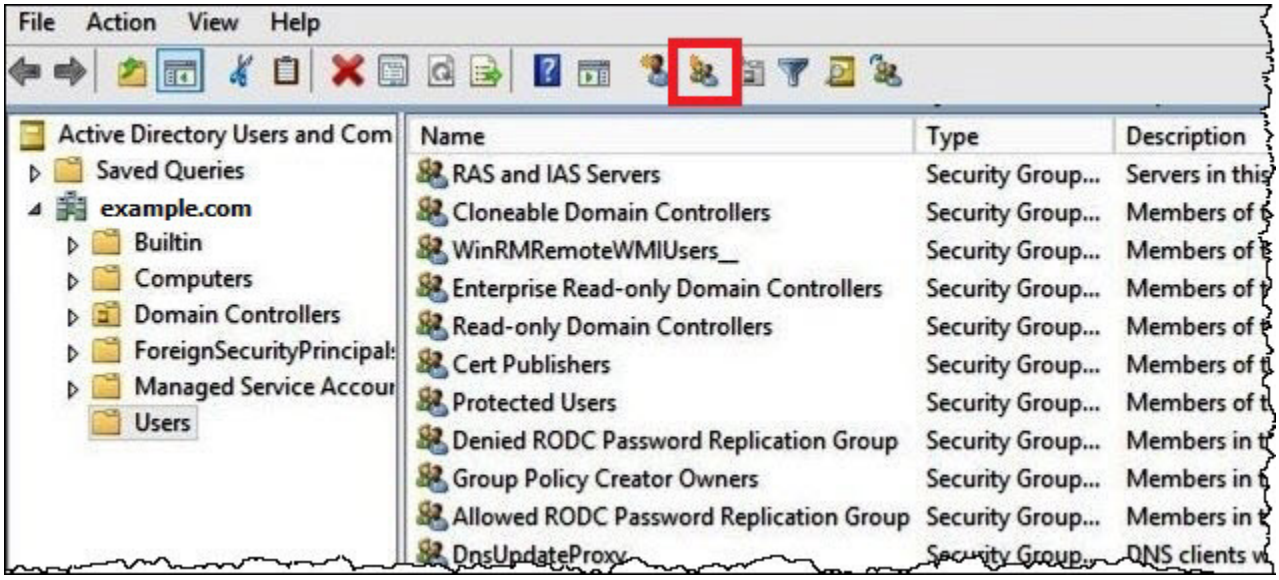
11. [OK] を選択します。

データアクセスパターンを表す AD グループを作成する

メンバーが AWS にログインしたときに `adfs-data-access` IAM ロールを設定する AD グループを作成できます。次の例では、aws-adfs-data-access という名前の AD グループを作成します。

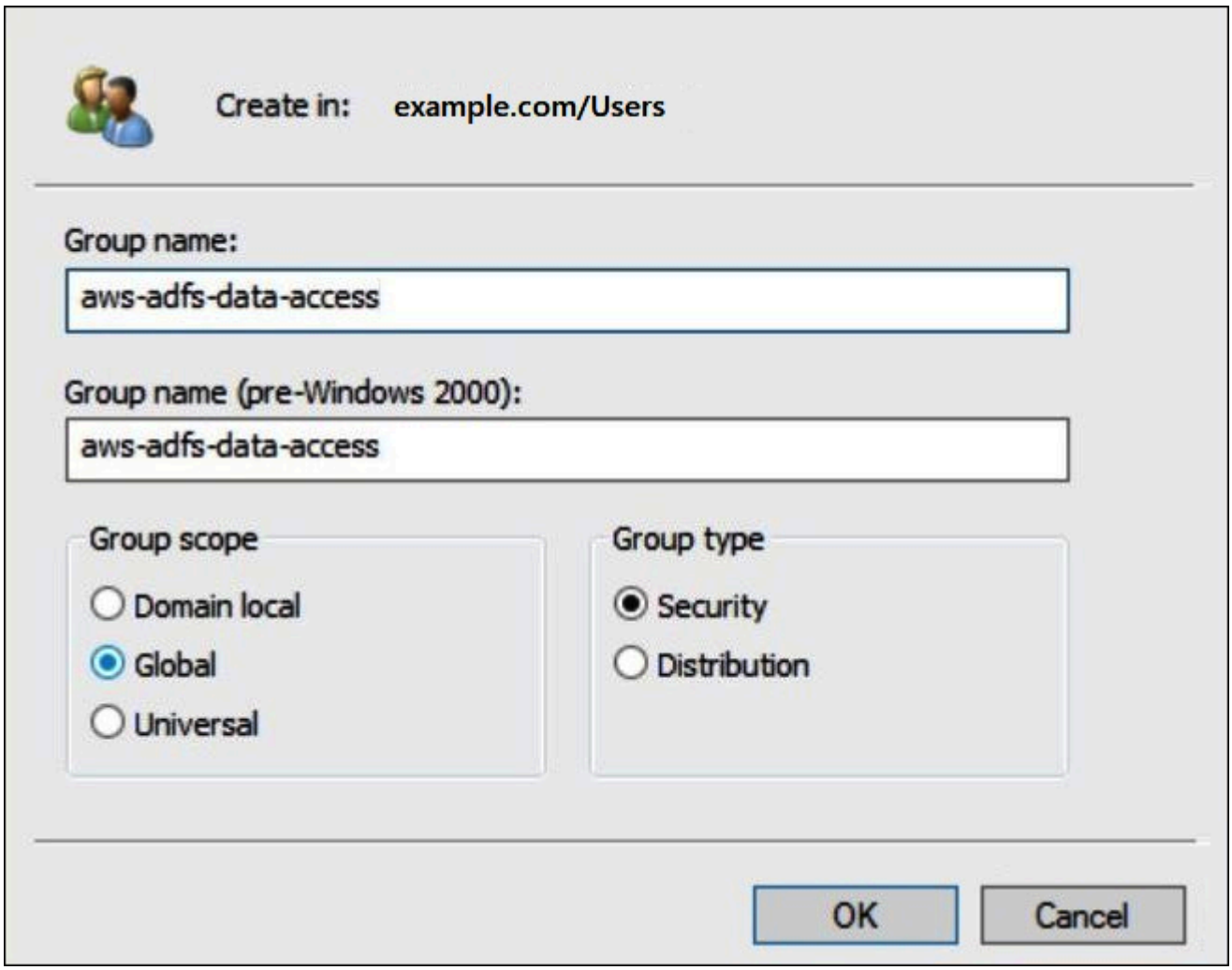
AD グループを作成するには

1. サーバーマネージャーダッシュボードの [Tools] (ツール) メニューで、[Active Directory Users and Computers] (Active Directory ユーザーとコンピュータ) を選択します。
2. ツールバーで、[Create new group] (新規グループを作成) オプションを選択します。



3. [New Object - Group] (新しいオブジェクト - グループ) ダイアログボックスで、次の情報を入力します。

   - [Group name] (グループ名) に、「**`aws-adfs-data-access`**」と入力します。
   - [Group scope] (グループスコープ) には、[Global] (グローバル) を選択します。
   - [Group type] (グループタイプ) には、[Security] (セキュリティ) を選択します。



4. [OK] を選択します。

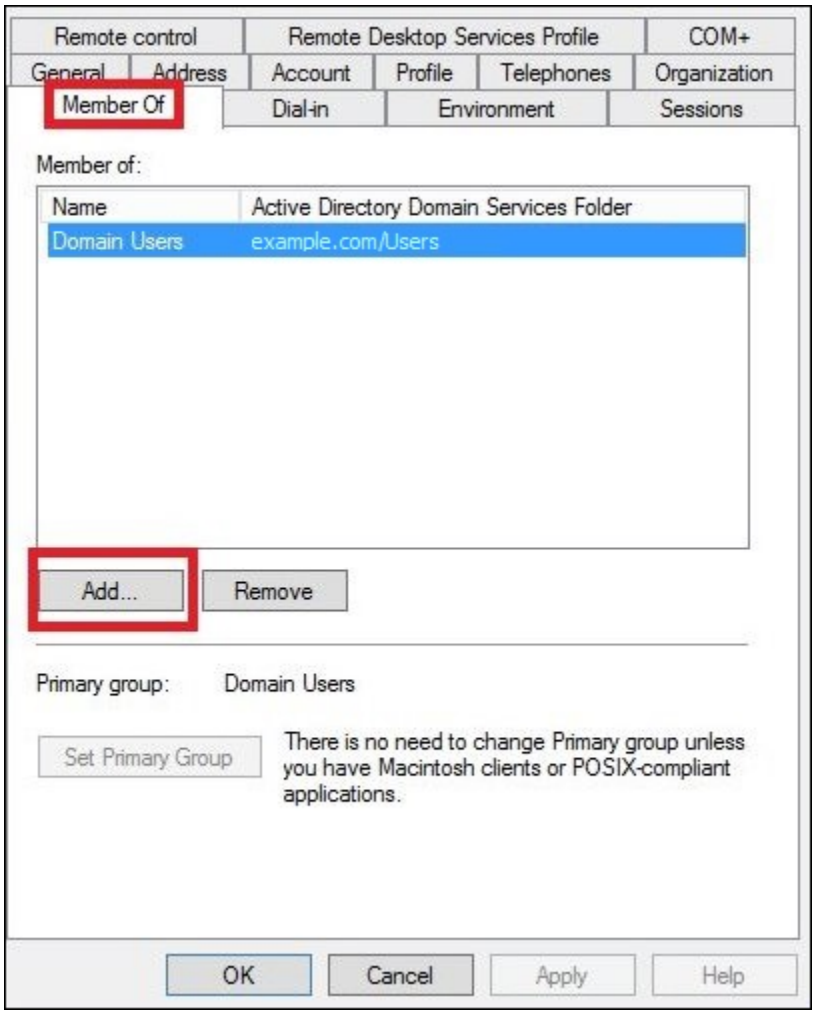
AD ユーザーを適切なグループに追加する

AD ユーザーと AD グループの両方を作成したため、ユーザーをグループに追加できるようになりました。

AD グループに AD ユーザーを追加するには

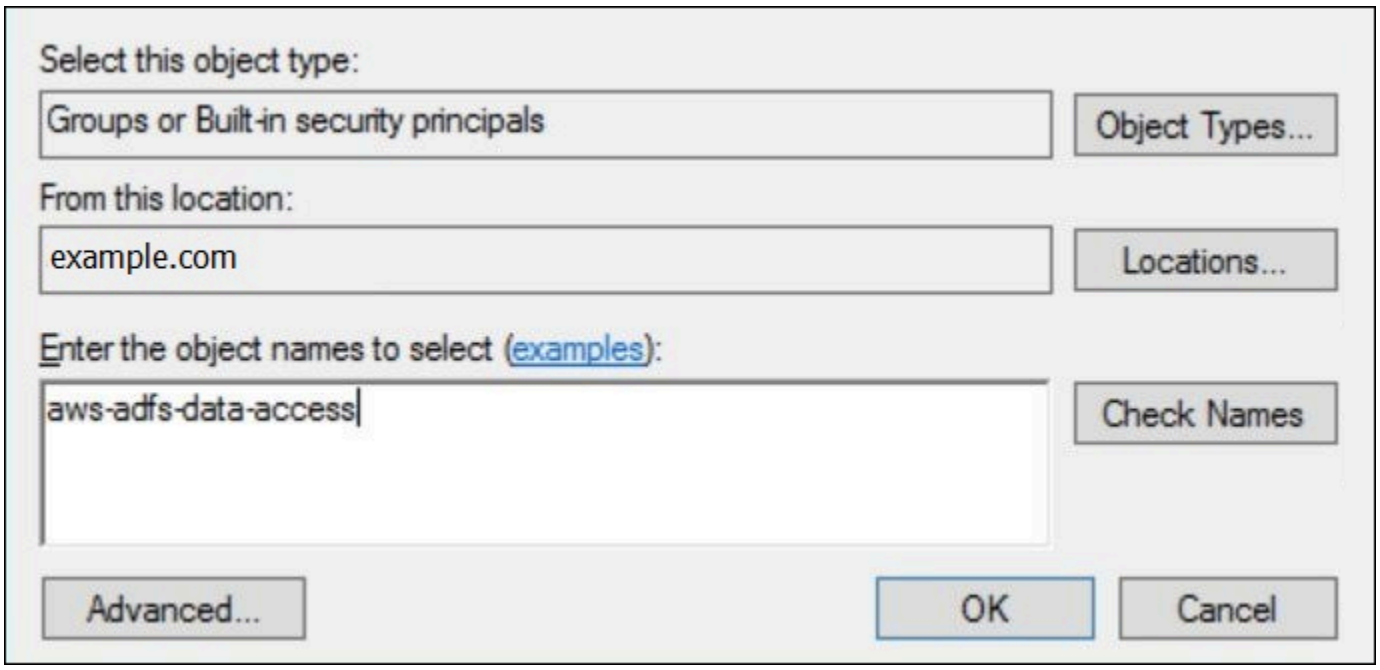
1. サーバーマネージャーダッシュボードで、[Tools] (ツール) メニューで、[Active Directory Users and Computers] (Active Directory ユーザーとコンピュータ) を選択します。
2. [First name] (ファーストネーム) と [Last name] (ラストネーム) には、[user] (ユーザー) ([Jane Doe] など) を選択します。

3. ユーザーの [Properties] (プロパティ) ダイアログボックスの [Member Of] (メンバー) タブで、[Add] (追加) を選択します。



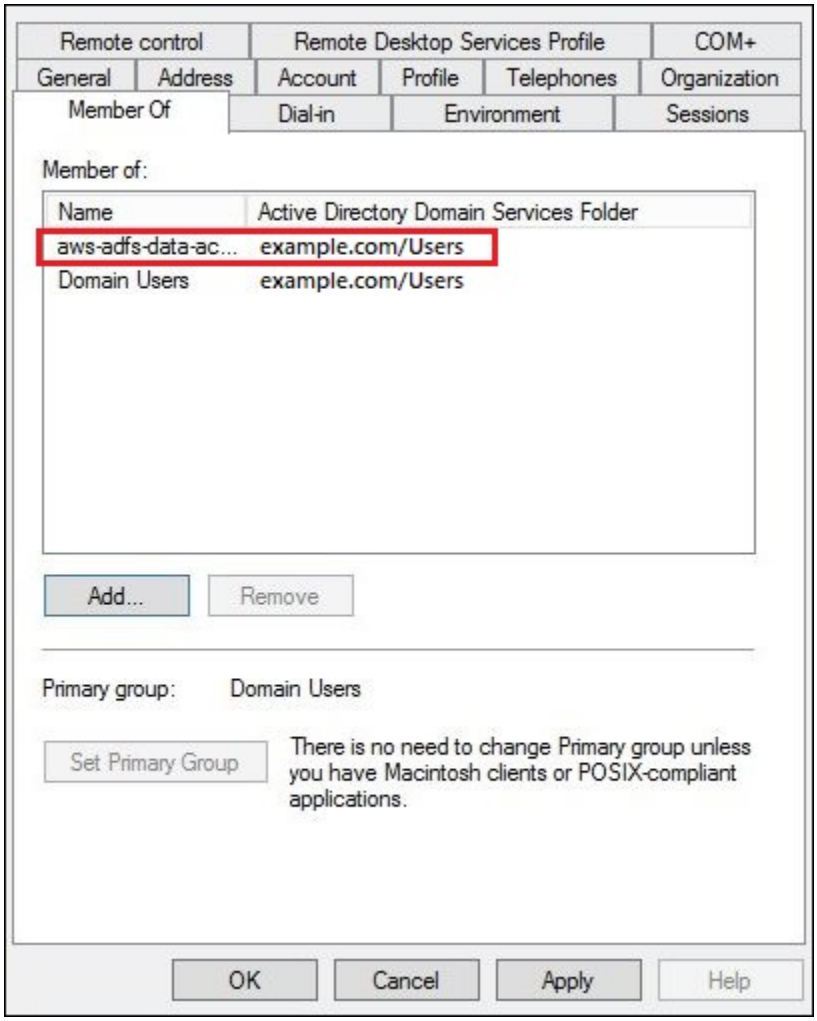
4. 要件に応じて、1 つ以上の AD FS グループを追加します。このチュートリアルでは、[aws-adfs-data-access] グループを追加します。

5. [Select Groups] (グループの選択) ダイアログボックスの [Enter the object names to select] (選択するオブジェクト名を入力) に、作成した AD FS グループの名前 (例: **`aws-adfs-data-access`**) を入力し、[Check Names] (名前の確認) を選択します。



6. [OK] を選択します。

   ユーザーの [Properties] (プロパティ) ダイアログボックスの [Member of] (メンバー) リストに AD グループの名前が表示されます。



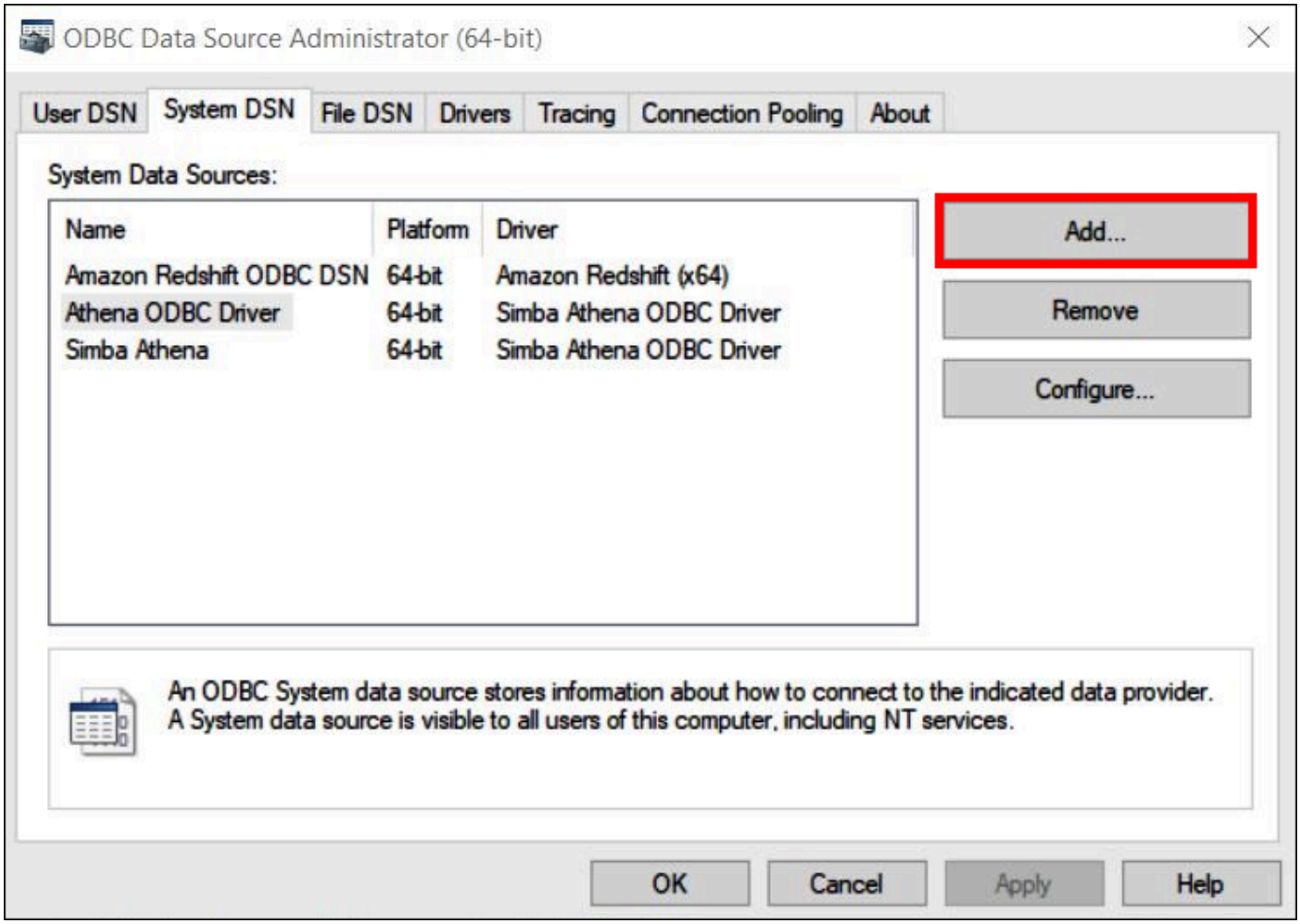
7. [Apply] (適用)、[OK] の順に選択します。

4. Athena への AD FS ODBC 接続を設定する

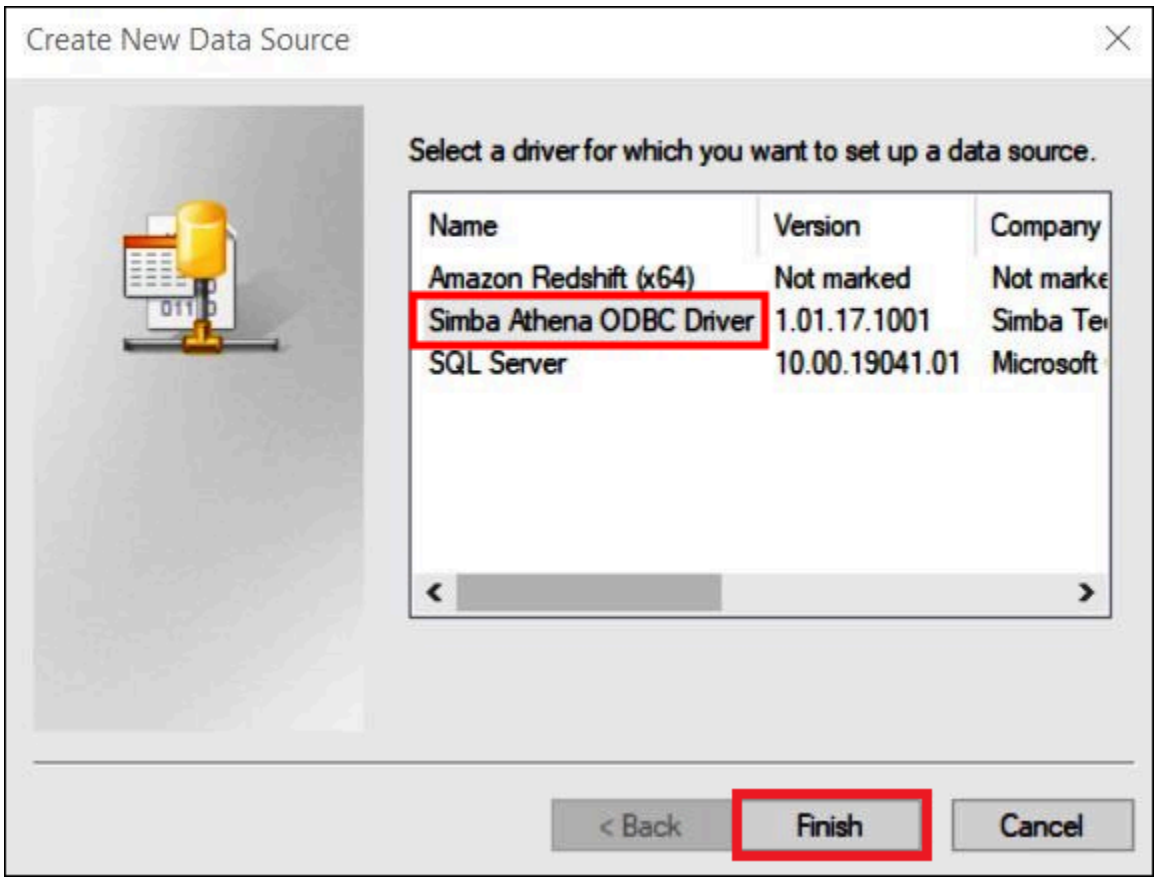
AD のユーザーとグループを作成すると、Windows で ODBC データソースプログラムを使用して、AD FS 用の Athena ODBC 接続を設定する準備が整います。

AD FS ODBC による Athena への接続を設定するには

1. Athena 用の ODBC ドライバーをインストールします。ダウンロードリンクについては、「ODBC を使用した Amazon Athena への接続」(ODBC を使用して Amazon Athena に接続する) を参照してください。
2. Windows で、[Start] (開始)、[ODBC Data Sources] (ODBC データソース) を選択します。
3. [ODBC Data Source Administrator] (ODBC データソースの管理者) プログラムで、[Add] (追加) をクリックします。

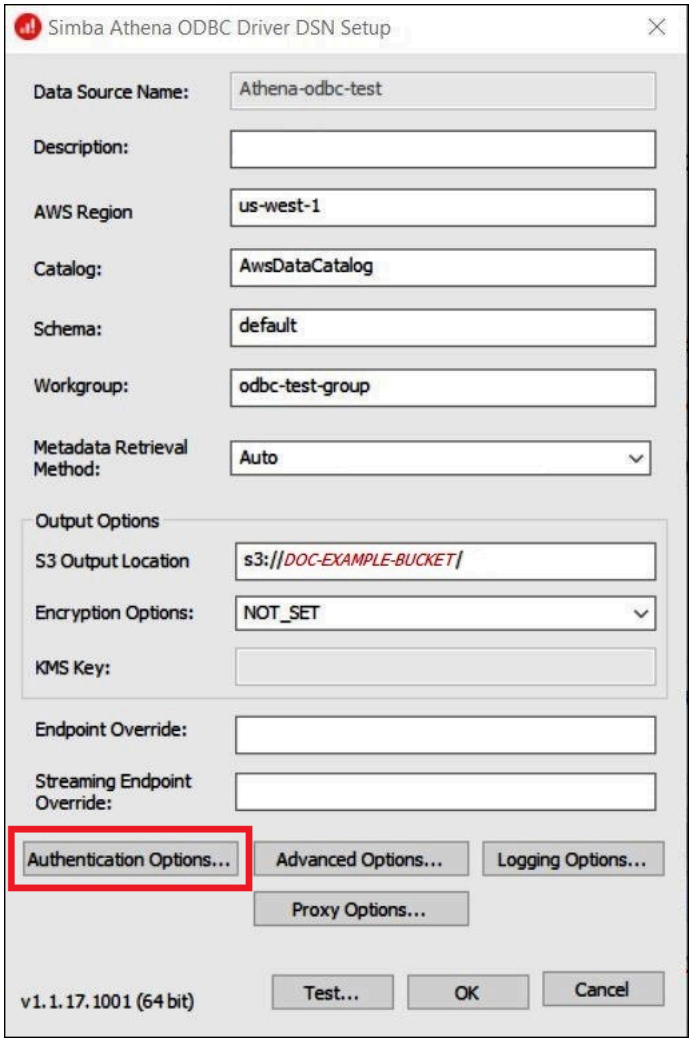


4. [Create New Data Source] (新しいデータソースの作成) ダイアログボックスで、[Simba Athena ODBC Driver] (Simba Athena ODBC ドライバー) を選択し、[Finish] (完了) を選択します。



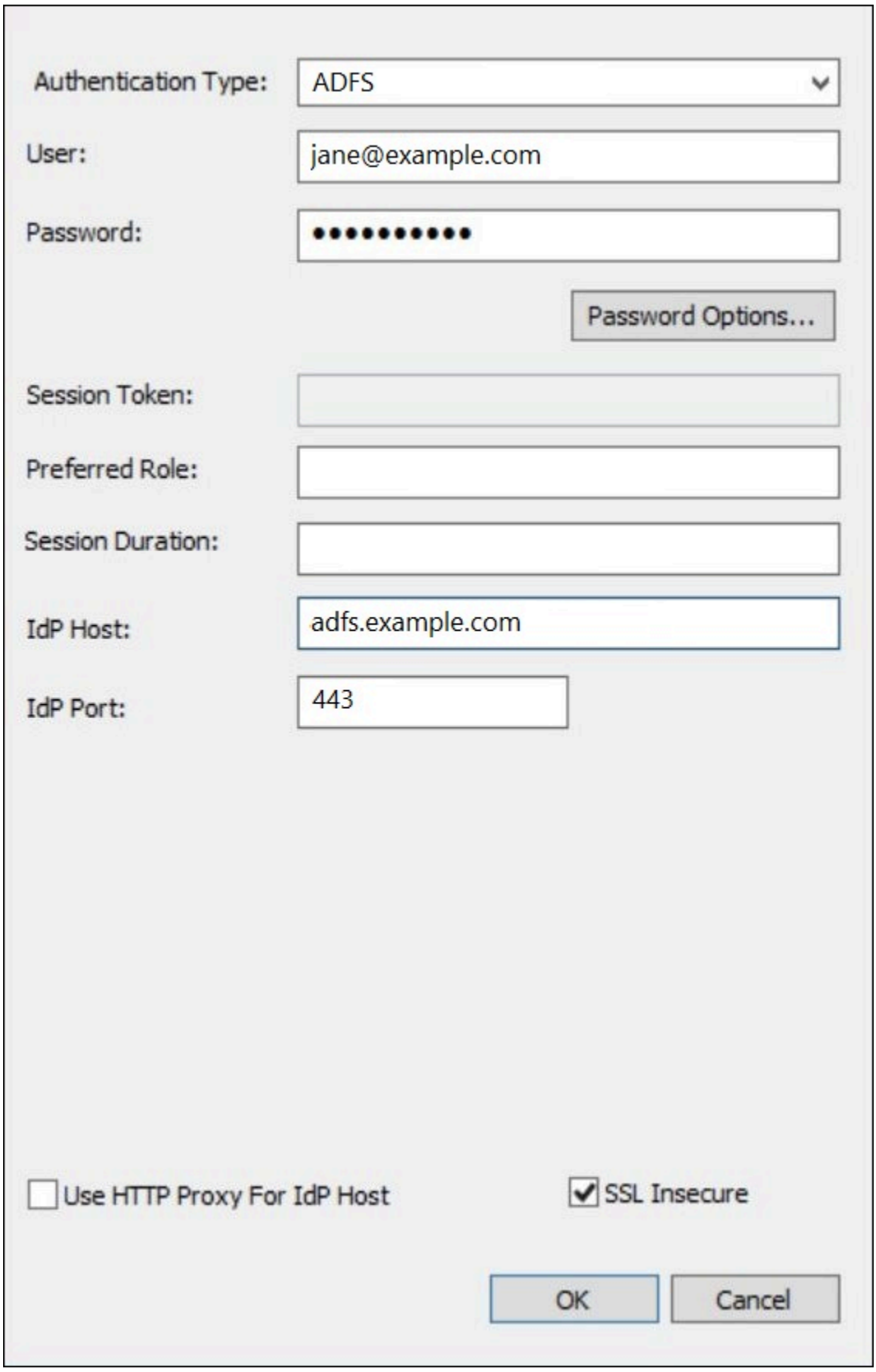
5. [Simba Athena ODBC Driver DSN Setup] (Simba Athena ODBC ドライバーの DSN 設定) ダイアログで、以下の値を入力します。

   - [Data Source Name] (データソース名) に、使用するデータソースの名前 (例えば、 **`Athena-odbc-test`**) を入力します。
   - [Description] (説明) に、データソースの説明を入力します。
   - [AWS リージョン] には、使用している AWS リージョン (例: **`us-west-1`**) を入力します
   - [S3 Output Location] (S3 出力場所) には、出力を保存する先の Amazon S3 パスを入力します。



6. [Authentication Options] (認証オプション) をクリックします。
7. [Authentication Options] (認証オプション) ダイアログボックスで、以下の値を指定します。

   - [Authentication Type] (認証タイプ) で、[ADFS] を選択します。
   - [User] (ユーザー) には、ユーザーのメールアドレス (例: **`jane@example.com`**) を入力します。
   - [Password] (パスワード) には、ユーザーの ADFS パスワードを入力します。
   - [IdP Host] (IdP ホスト) には、AD FS サーバー名 (例: **`adfs.example.com`**) を入力します。
   - [IdP Host] (IdP ポート) には、デフォルト値の [443] を使用します。
   - [SSL Insecure] (SSL 非セキュア) オプションを選択します。


Authentication Type:
ADFS
User:
jane@example.com
Password:
Password Options...
Session Token:
Preferred Role:
Session Duration:
IdP Host:
adfs.example.com
IdP Port:
443
Use HTTP Proxy For IdP Host
SSL Insecure
OK
Cancel

8. [OK] をクリックして、[Authentication Options] (認証オプション) を閉じます。
9. 接続をテストするには [Test] (テスト) を、終了するには [OK] をクリックします。

Okta プラグインと Okta ID プロバイダを使用して ODBC 用 の SSO を設定する

このページでは、Amazon Athena ODBC ドライバーと Okta プラグインを設定し、Okta の ID プロバイダを使用して Single Sign-On (SSO) 機能を追加する方法について説明します。

前提条件

このチュートリアルのステップを完了するには、以下が必要です。

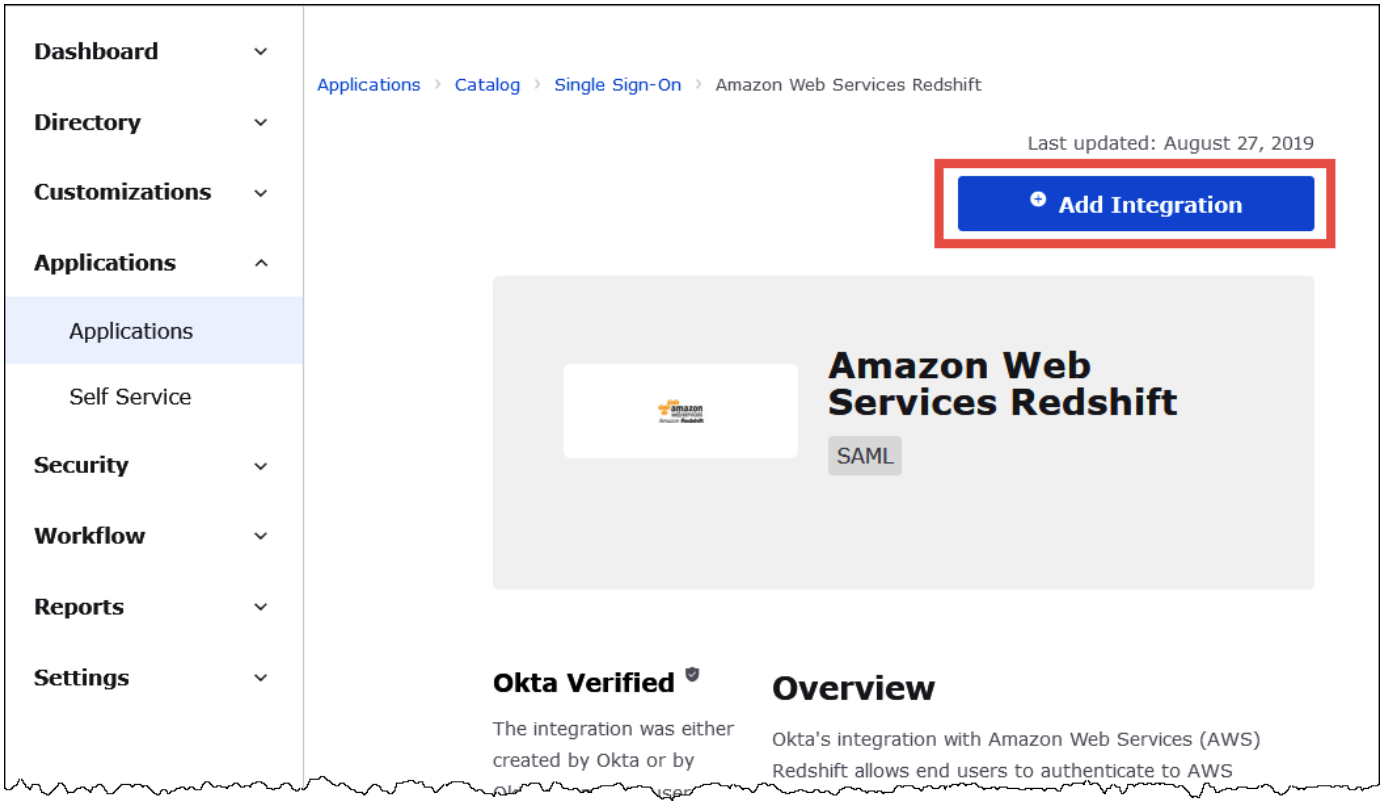
- Amazon Athena ODBC ドライバー ダウンロードリンクについては、「ODBC を使用した Amazon Athena への接続」(ODBC を使用して Amazon Athena に接続する) を参照してください。
- SAML で使用する IAM ロール。詳細については、「IAM ユーザーガイド」の「SAML 2.0 フェデレーション用のロールの作成 (コンソール)」を参照してください。
- Okta アカウント。詳細については、Okta.com を参照してください。

Okta でのアプリケーション統合の作成

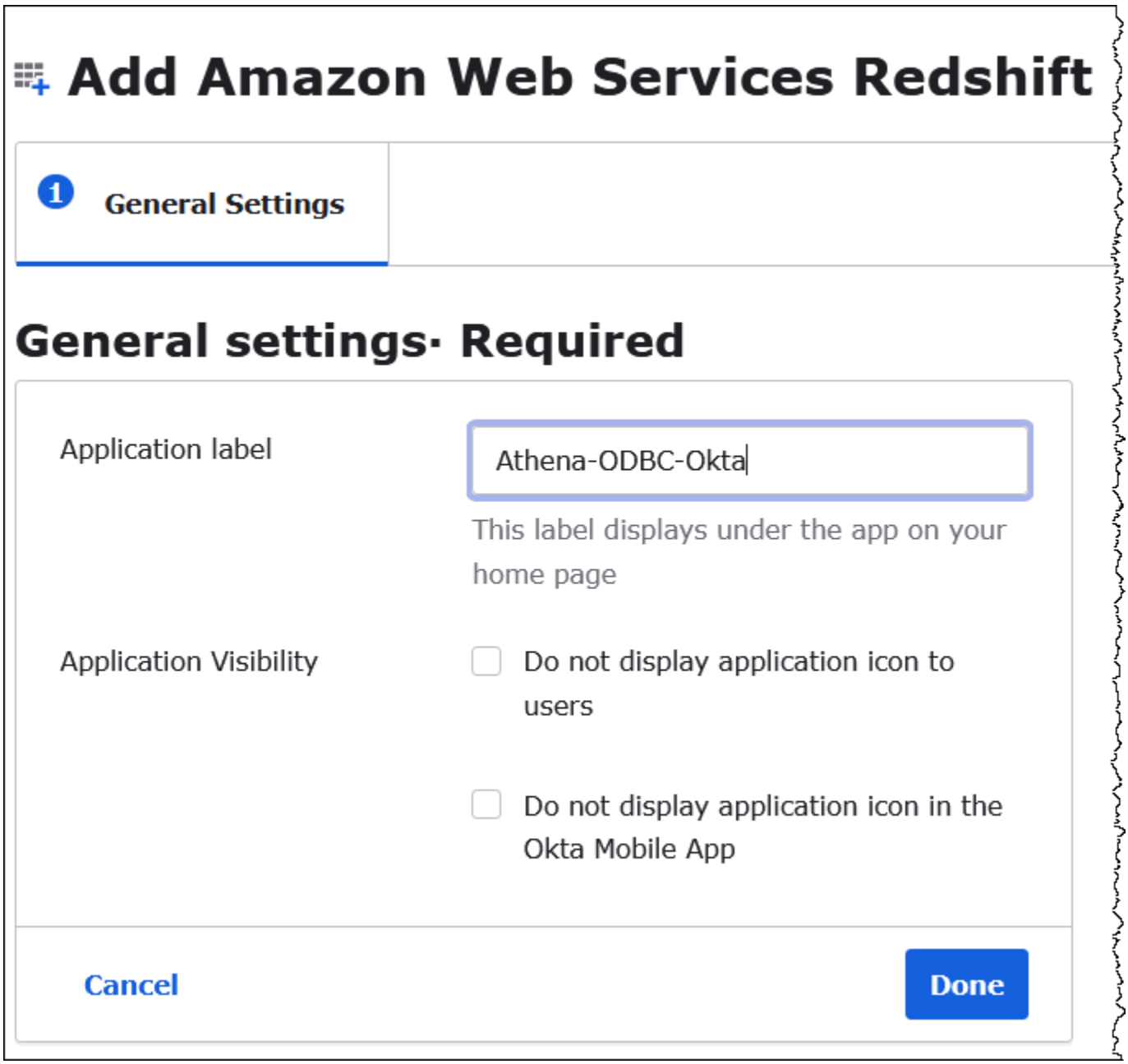
まず、Okta ダッシュボードを使用して、Athena へのシングルサインオン用の SAML 2.0 アプリケーションを作成して設定します。Okta で既存の Redshift アプリケーションを使用して、Athena へのアクセスを設定できます。

Okta でアプリケーション統合を作成するには

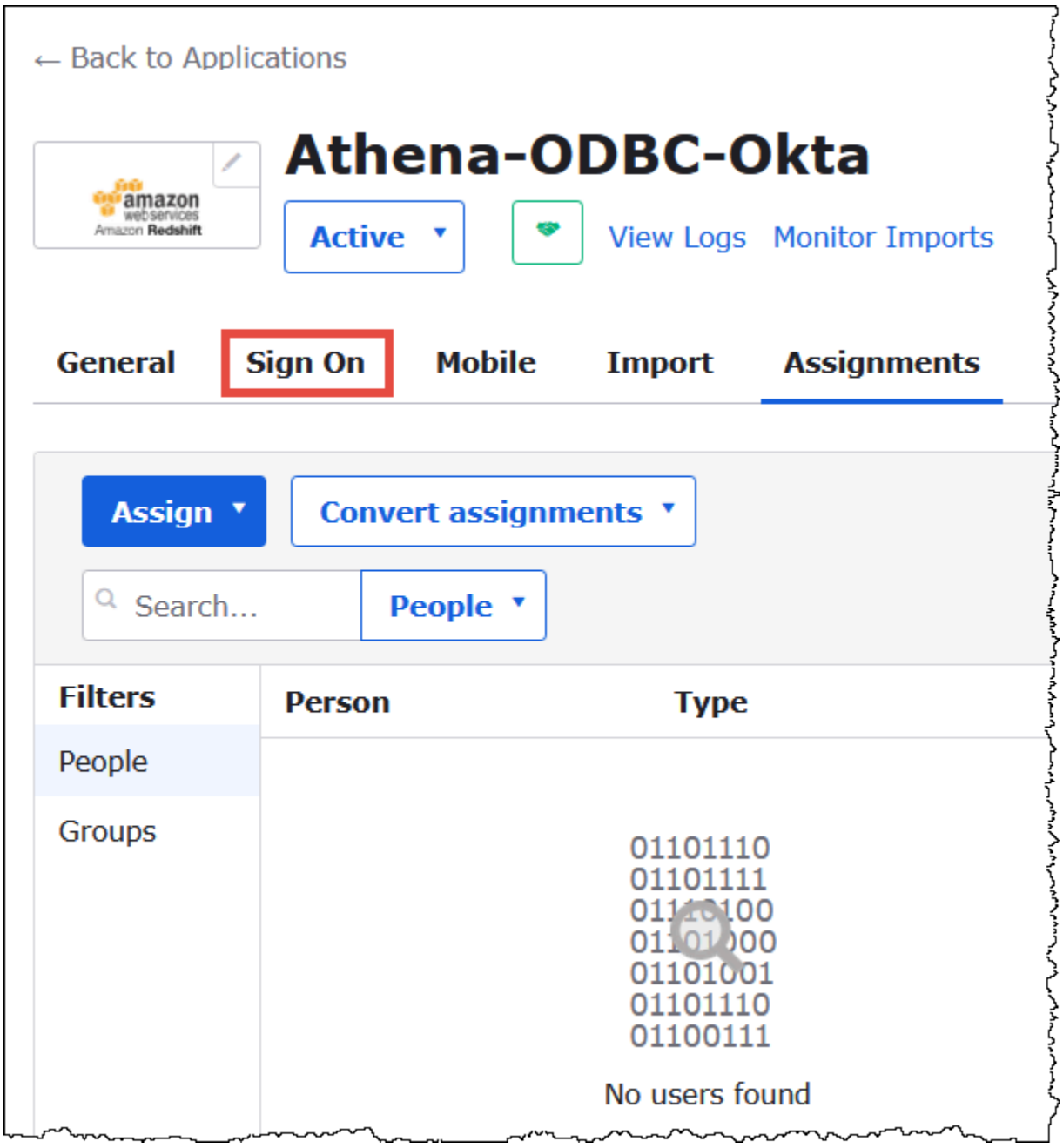
1. Okta.com で、アカウントの管理ページにログインします。
2. ナビゲーションパネルで、[Applications] (アプリケーション) を選択した後、もう一度 [Applications] (アプリケーション) を選択します。
3. [Applications] (アプリケーション) ページで、[Browse App Catalog] (アプリケーションカタログを参照) を選択します。
4. [Browse App Integration Catalog] (アプリケーションの統合カタログを表示) ページの [Use Case] (ユースケース) セクションで、[All Integrations] (すべての統合) を選択します。
5. 検索ボックスに Amazon Web Services Redshift と入力してから、[Amazon Web Services Redshift SAML] (アマゾン ウェブ サービス Redshift SAML) を選択します。
6. [Add integration] (統合の追加) を選択します。



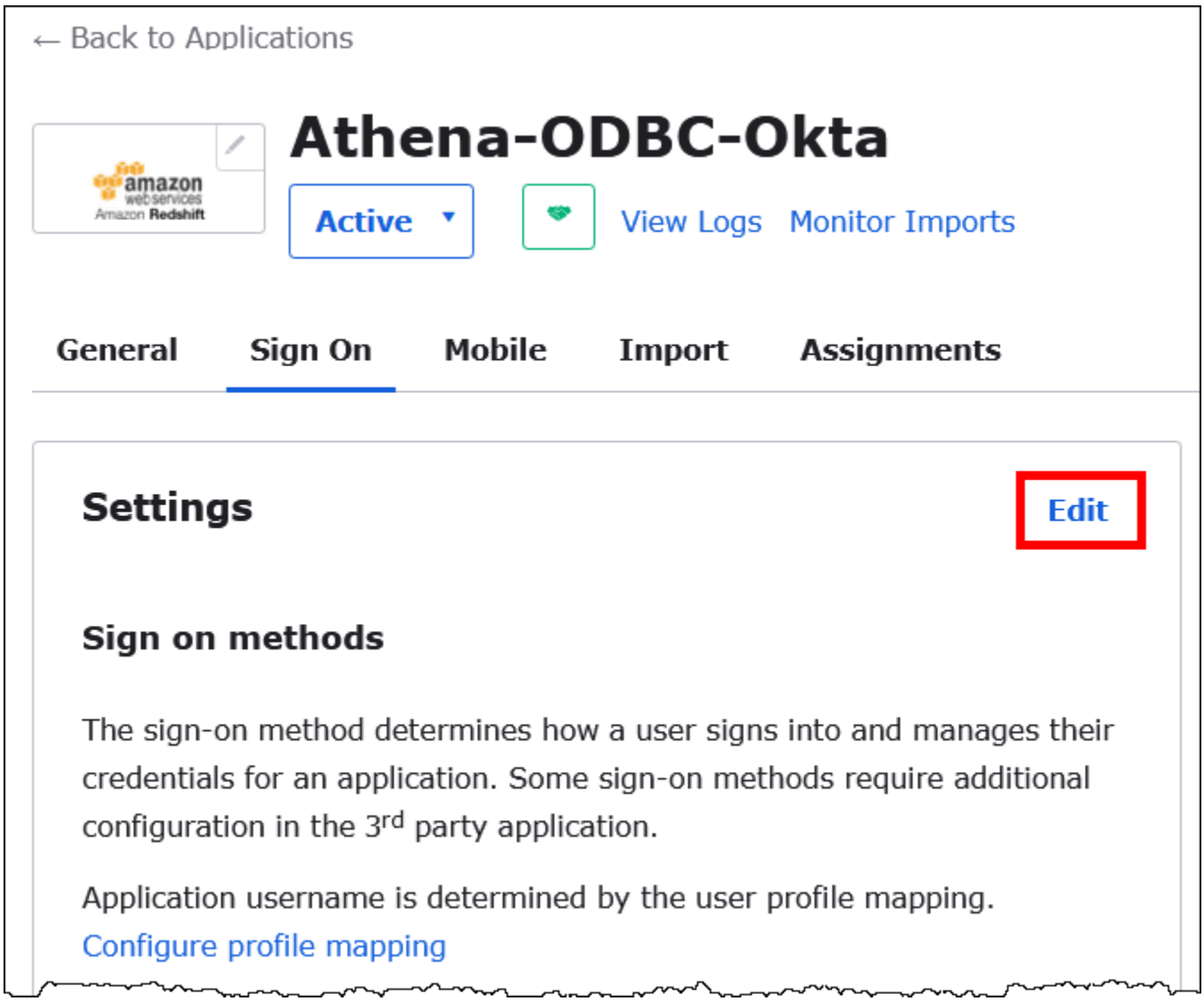
7. [General Settings Required] (必要な一般設定) セクションの [Application label] (アプリケーションラベル) に、アプリケーションの名前を入力します。このチュートリアルでは、Athena-ODBC-Okta という名前を使用します。



8. [完了] をクリックします。
9. Okta アプリケーションのページ (たとえば、[Athena-ODBC-Okta]) で、[Sign On] (サインオン) を選択します。

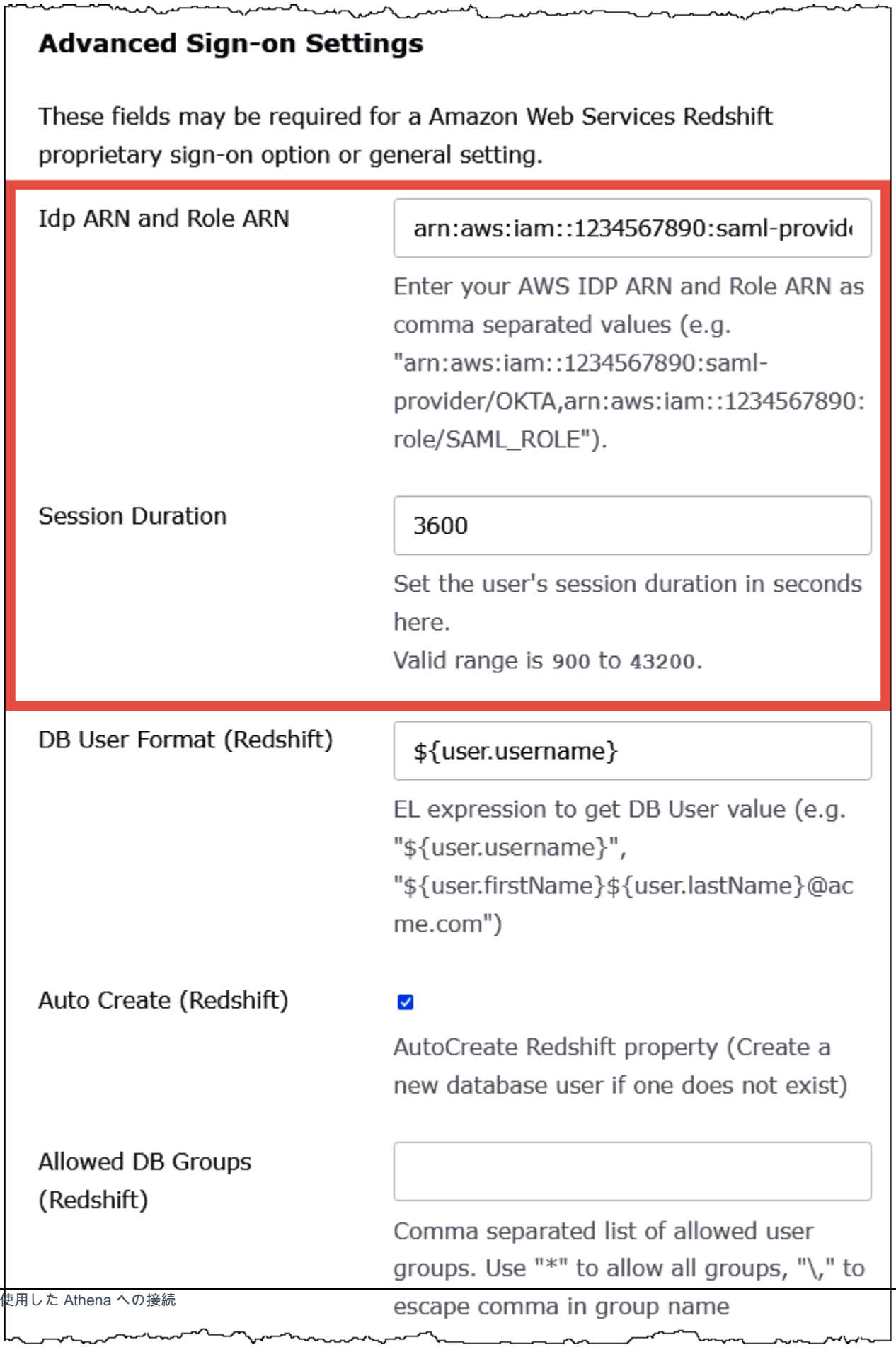


10. [Settings] (設定) セクションで、[Edit] (編集) を選択します。



11. [Advanced Sign-on Settings] (詳細なサインオン設定) セクションで、以下の値を設定します。

    - [IdP ARN and Role ARN] (IdP ARN とロール ARN) に、AWS IDP ARN とロール ARN をカンマ区切り値で入力します。IAM ロール形式の詳細については、「IAM ユーザーガイド」の「認証レスポンスの SAML アサーションを設定する」を参照してください。
    - [Session Duration] (セッション期間) に、900 ~ 43200 秒の値を入力します。このチュートリアルでは、デフォルトの 3600 (1 時間) を使用します。


Advanced Sign-on Settings
These fields may be required for a Amazon Web Services Redshift proprietary sign-on option or general setting.
Idp ARN and Role ARN
arn:aws:iam::1234567890:saml-provid
Enter your AWS IDP ARN and Role ARN as comma separated values (e.g. "arn:aws:iam::1234567890:saml-provider/OKTA,arn:aws:iam::1234567890:role/SAML_ROLE").
Session Duration
3600
Set the user's session duration in seconds here.
Valid range is 900 to 43200.
DB User Format (Redshift)
${user.username}
EL expression to get DB User value (e.g. "${user.username}", "${user.firstName}${user.lastName}@acme.com")
Auto Create (Redshift)
AutoCreate Redshift property (Create a new database user if one does not exist)
Allowed DB Groups (Redshift)
Comma separated list of allowed user groups. Use "*" to allow all groups, "\," to escape comma in group name

Athena では、[DbUser Format] (DbUser 形式)、[AutoCreate]、[Allowed DBGroups] (許可された DBGroups) の設定は使用されません。ユーザーがこれらを設定する必要はありません。

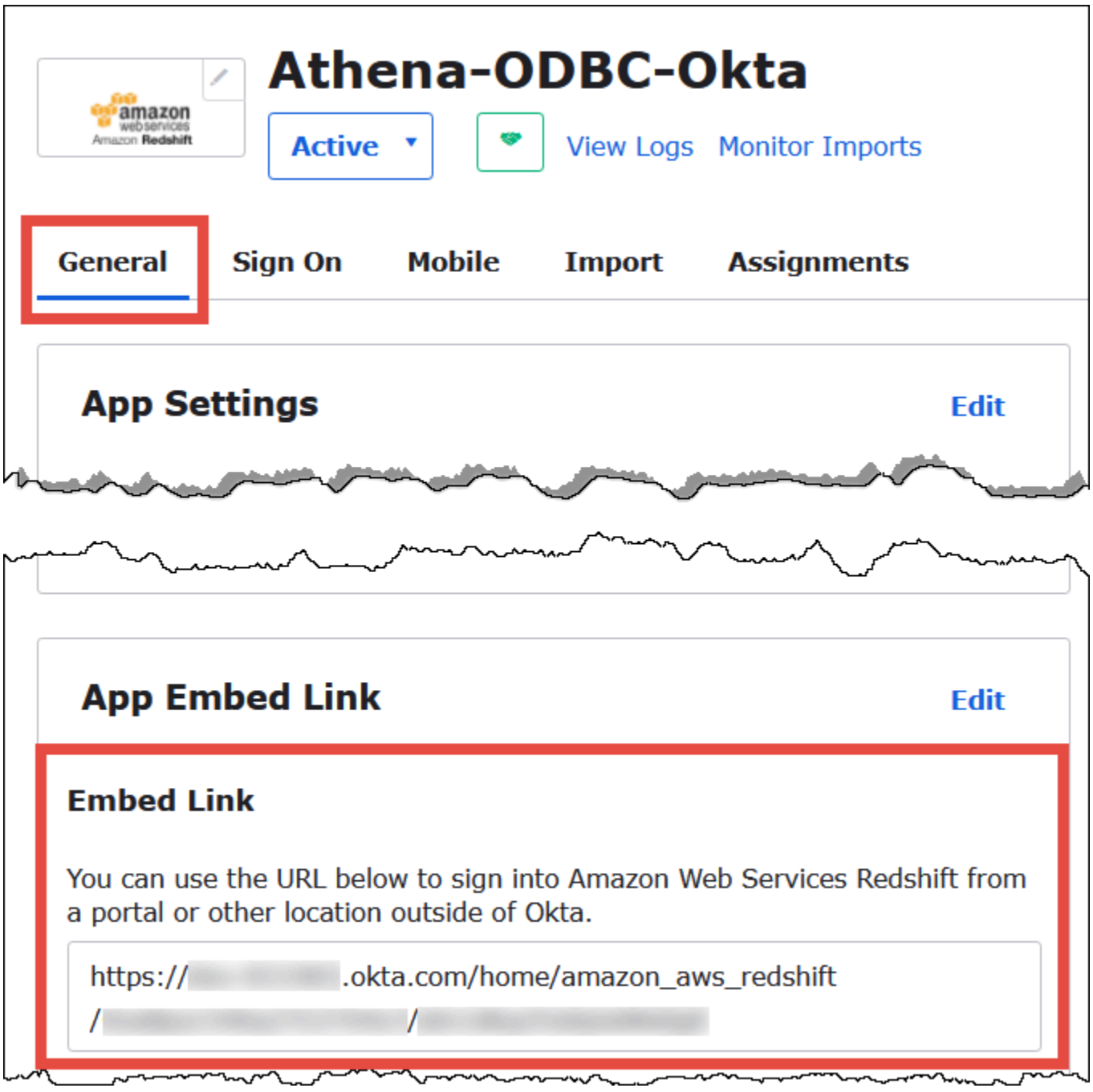
12. [Save] を選択します。

Okta から ODBC の設定情報を取得する

Okta アプリケーションを作成し、アプリケーションの ID と IdP ホスト URL を取得する準備ができました。これらは、後で Athena への接続用に ODBC を設定するときに必要になります。

Okta から ODBC の設定情報を取得するには

1. Okta アプリケーションの [General] (一般) タブを選択してから、[App Embed Link] (アプリケーション埋め込みリンク) セクションまでスクロールダウンします。



[Embed Link] (埋め込みリンク) の URL の形式は以下のとおりです。

```
https://trial-1234567.okta.com/home/amazon_aws_redshift/Abc1de2fghi3J45kL678/
abc1defghij2klmNo3p4
```

2. [Embed Link] (埋め込みリンク) の URL から、次の部分を抽出して保存します。

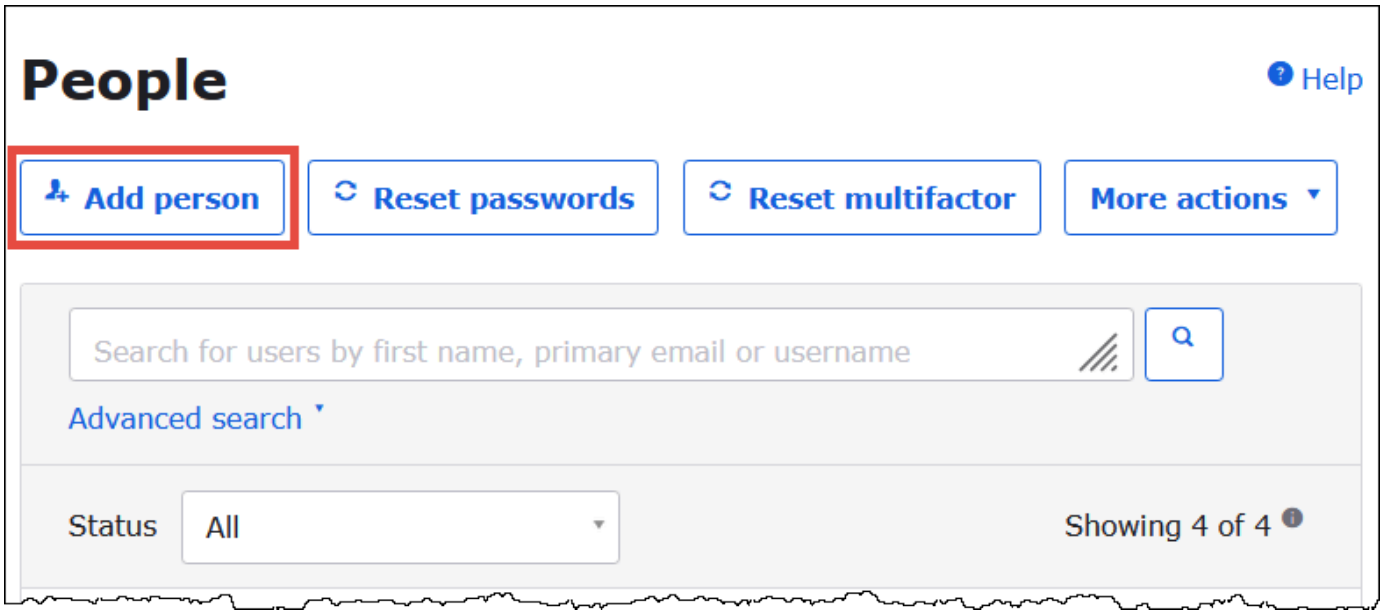
- `https://` に続く最初のセグメント、`okta.com` までです (たとえば、trial-1234567.okta.com)。これは IdP ホストです。
- URL に含まれる最後の 2 つのセグメント、中央のスラッシュを含みます。これら 2 つのセグメントは、数字と大文字と小文字が混在する 20 文字の文字列 2 組です (たとえば、Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4)。これはアプリケーション ID です。

Okta アプリケーションにユーザーを追加します。

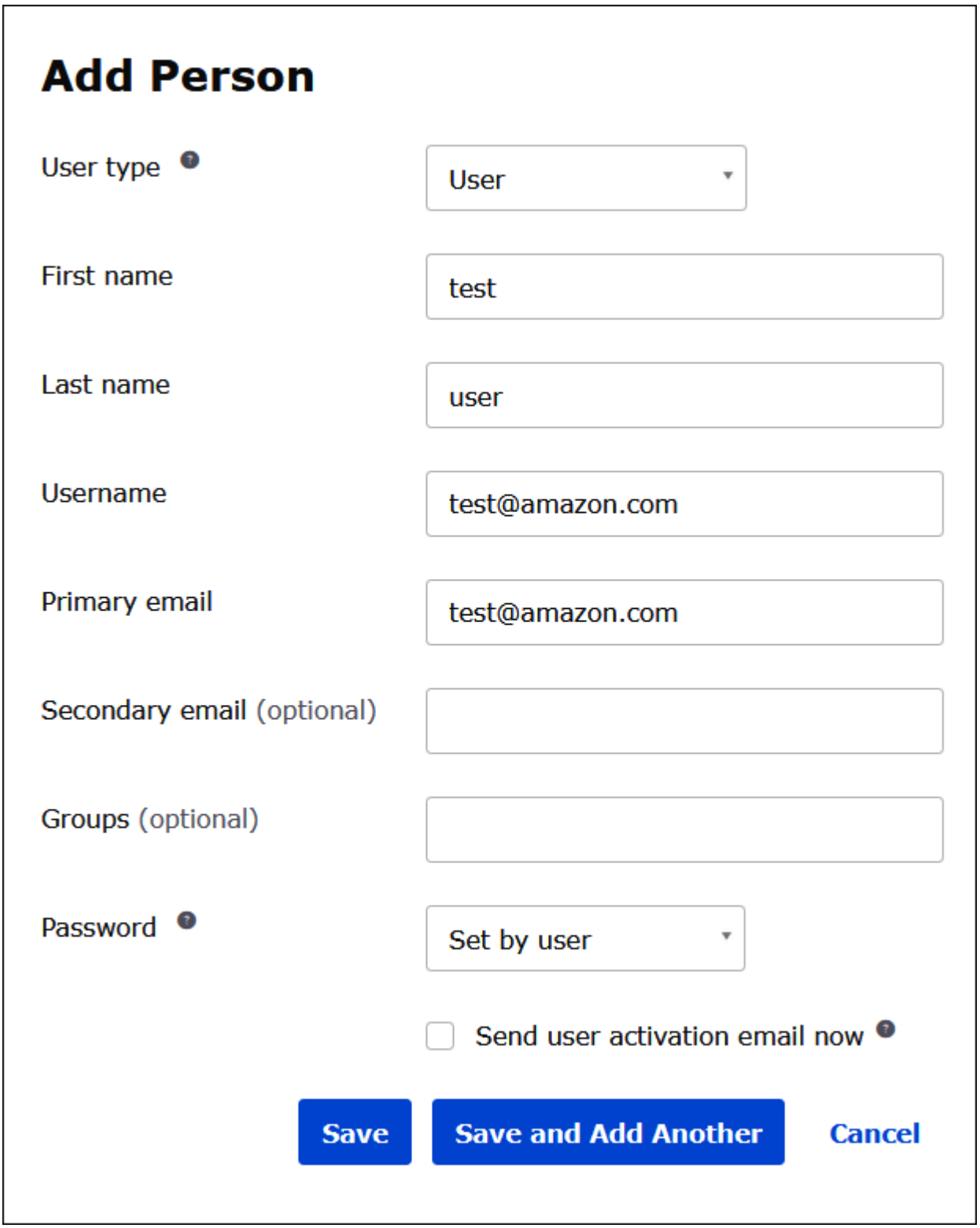
Okta アプリケーションにユーザーを追加する準備ができました。

Okta アプリケーションにユーザーを追加するには

1. 左側のナビゲーションペインで、[ディレクトリ] 、[ピープル] の順に選択します。
2. [Add Person] (ユーザーを追加) を選択します。



3. [Add Person] (ユーザーを追加) ダイアログボックスで、以下の情報を入力します。
   - [First name] (名) と [Last name] (姓) に値を入力します。このチュートリアルでは、**`test user`** を使用します。
   - [Username] (ユーザー名) と [Primary email] (プライマリ E メール) に値を入力します。このチュートリアルでは、両方に **`test@amazon.com`** を使用します。パスワードのセキュリティ要件はユーザーによって異なる可能性があります。

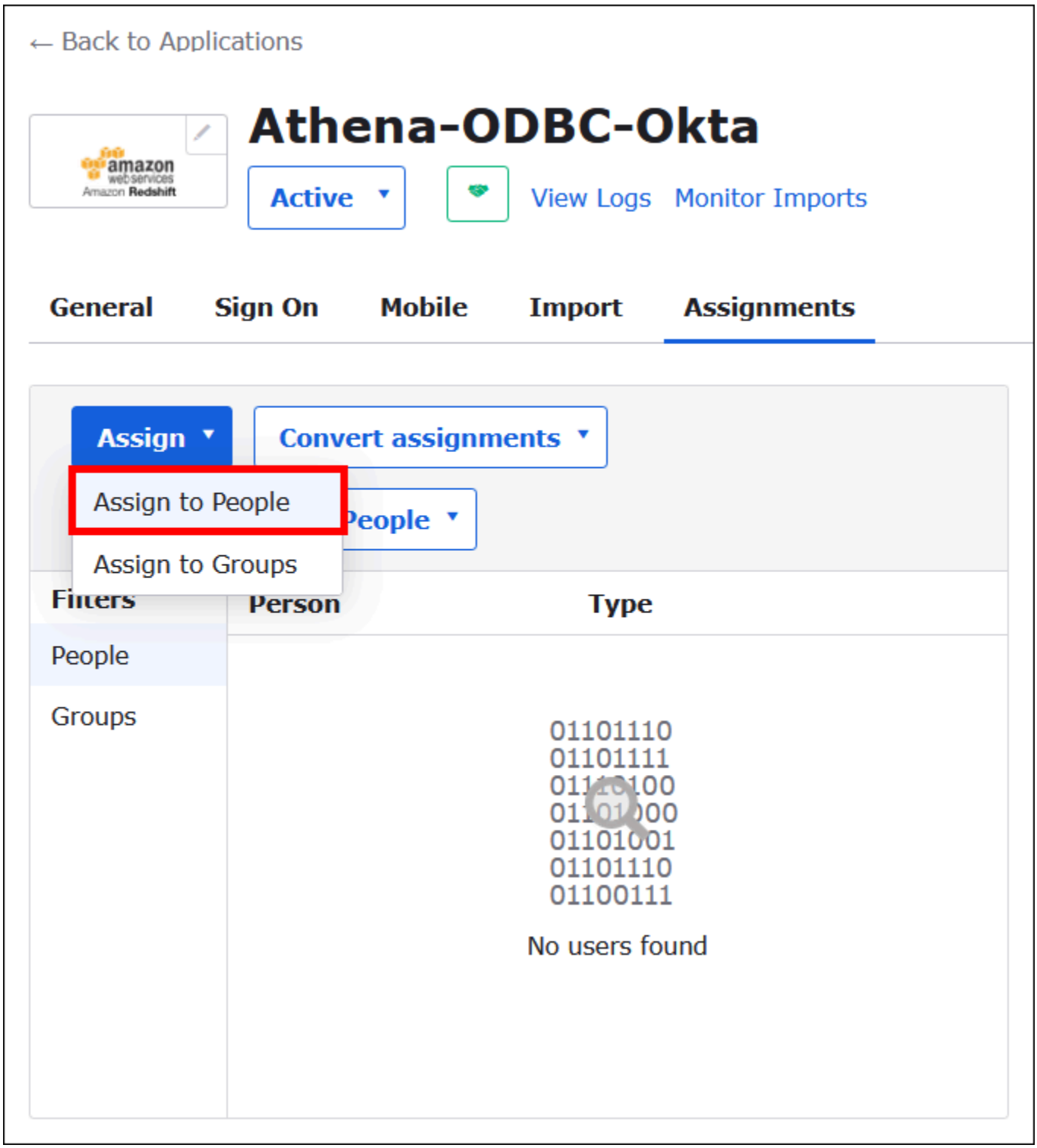


4. [Save] を選択します。

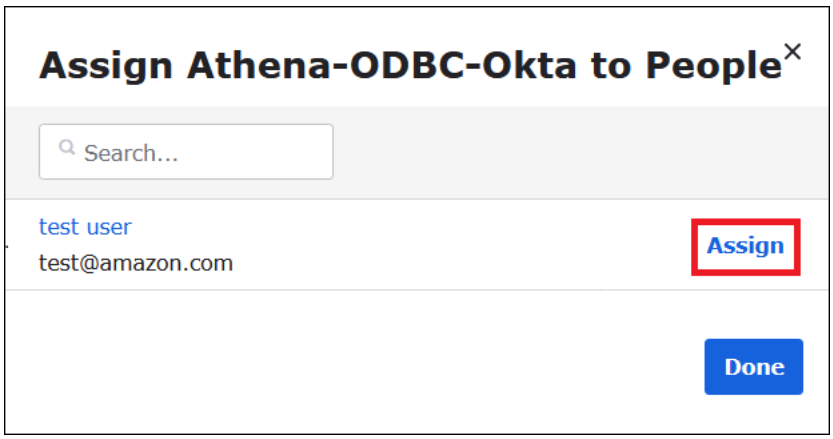
作成したユーザーをアプリケーションに割り当てる準備ができました。

アプリケーションにユーザーを割り当てるには

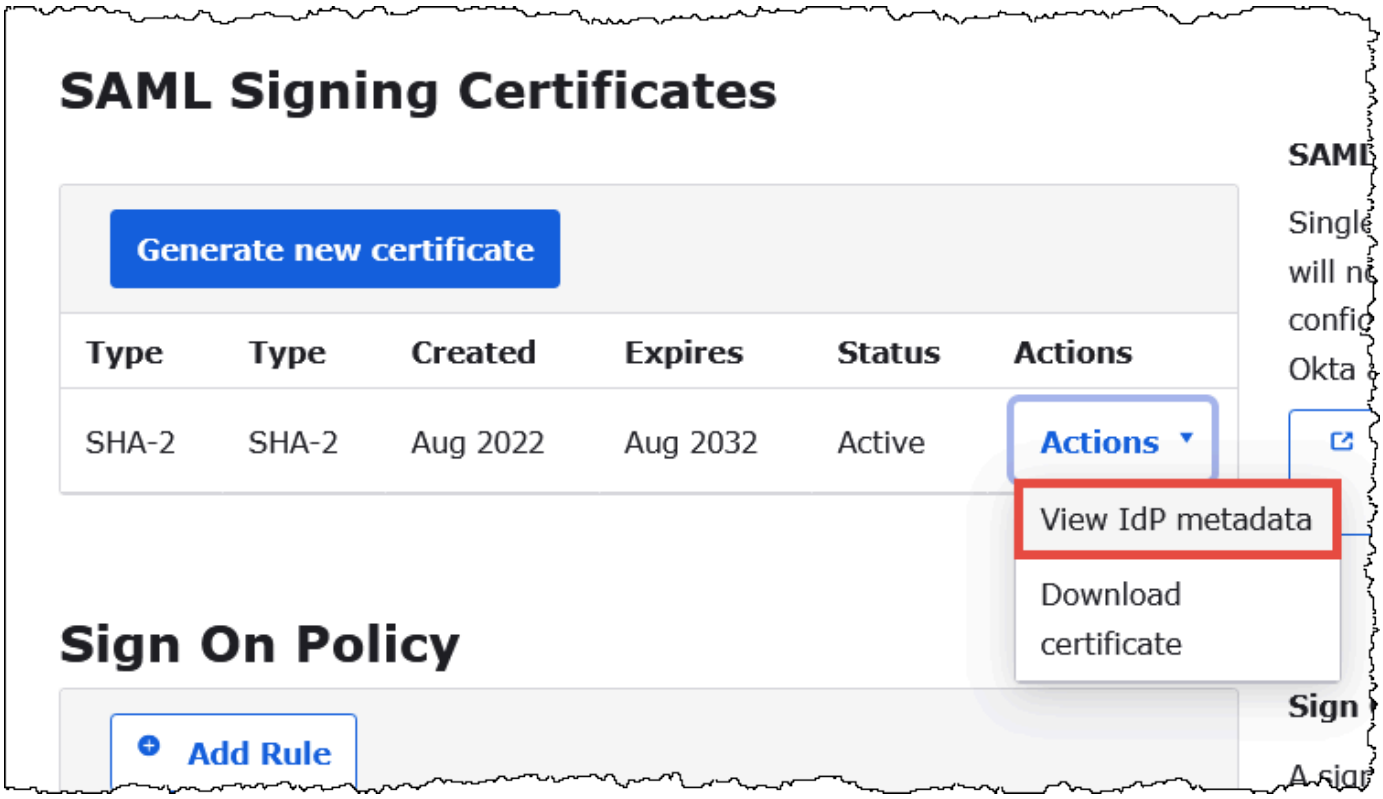
1. ナビゲーションペインで、[Applications] (アプリケーション) を選択してから、もう一度 [Applications] (アプリケーション) を選択し、アプリケーションの名前を選択します (たとえば、[Athena-ODBC-Okta])。
2. [Assign] (割り当て) を選択してから、[Assign to People] (ユーザーに割り当て) を選択します。



3. ユーザーの [Assign] (割り当て) オプションを選択し、[Done] (完了) を選択します。



4. プロンプトで、[Save and Go Back] (保存して戻る) を選択します。ダイアログボックスで、ユーザーのステータスが [Assigned] (割り当て済み) と表示されます。
5. [完了] をクリックします。
6. [Sign On] (サインオン) タブを選択します。
7. [SAML Signing Certificates] (SAML 署名証明書) セクションまでスクロールダウンします。
8. [アクション] を選択します。
9. [View IdP metadata] (IdP メタデータを表示) コンテキストメニューを開き (右クリック)、ファイルを保存するためのブラウザオプションを選択します。
10. 拡張子 (`.xml`) を付けてファイルを保存します。

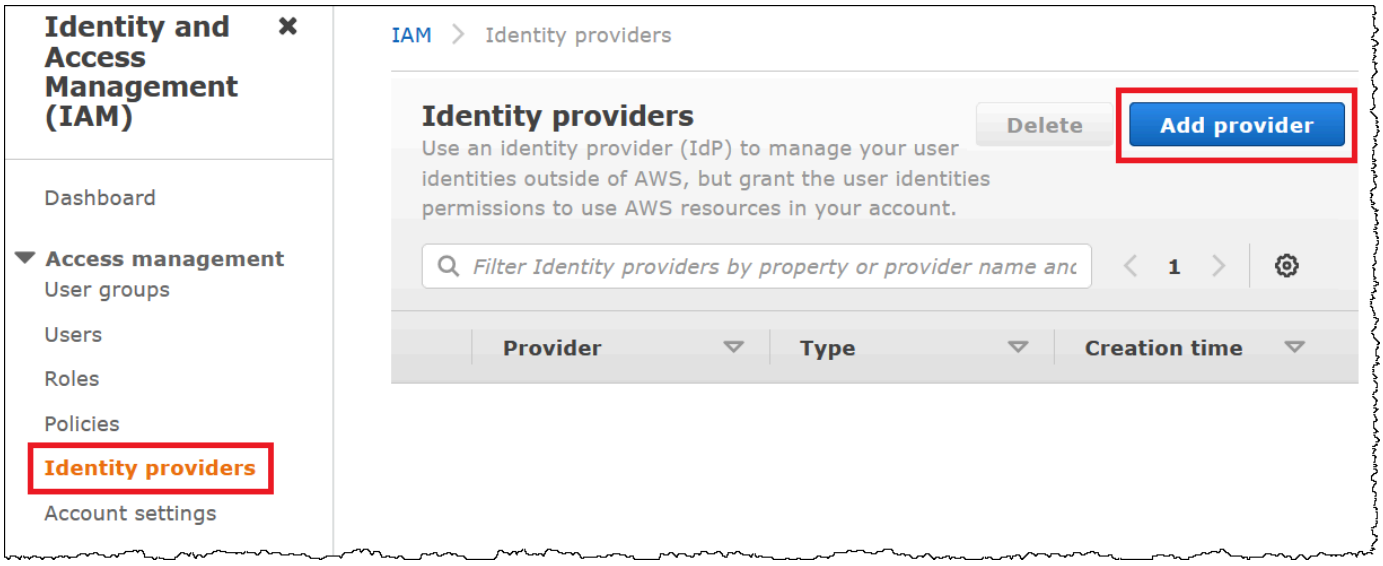


AWS SAML ID プロバイダとロールを作成する

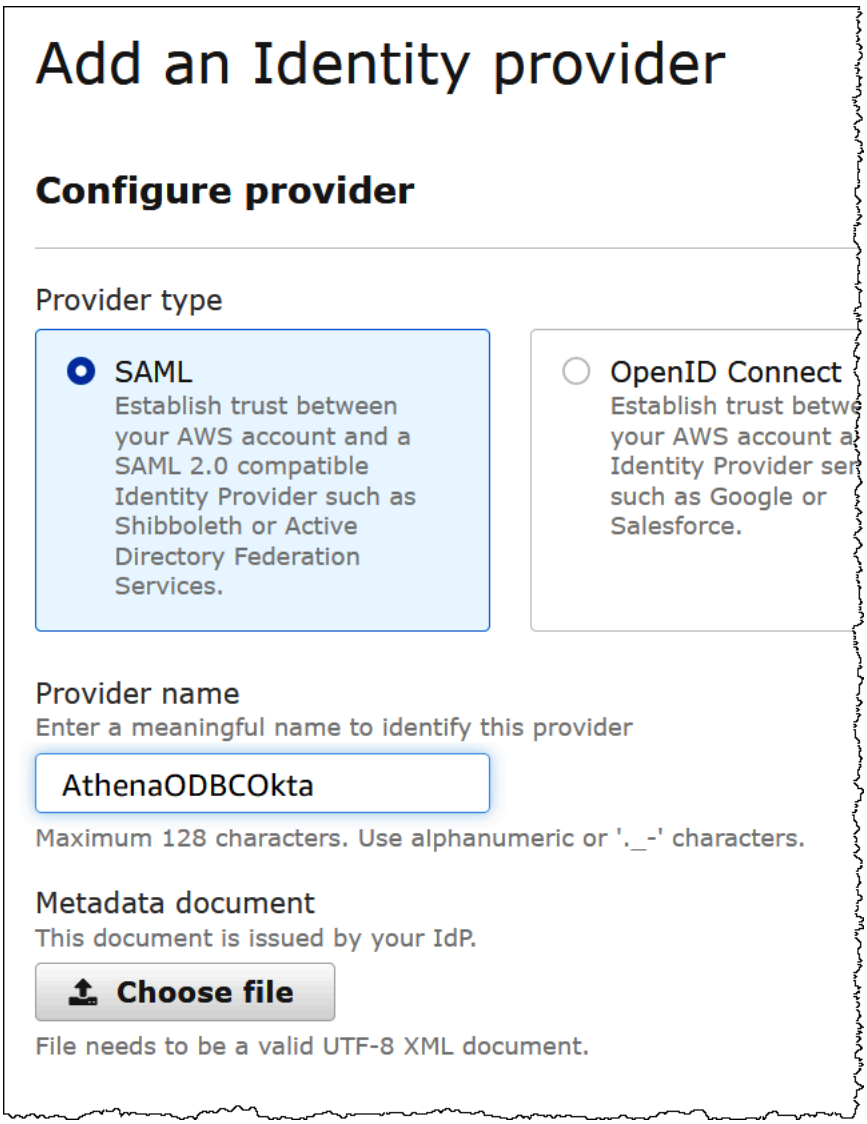
メタデータ XML ファイルを AWS の IAM コンソールにアップロードする準備ができました。このファイルを使用して、AWS SAML ID プロバイダとロールを作成します。これらの手順を実行するには、AWS のサービスの管理者アカウントを使用します。

AWS で SAML ID プロバイダとロールを作成するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/IAM/ の IAM コンソールを開きます。
2. ナビゲーションペインで、[Identity providers] (ID プロバイダ) を選択し、[Add provider] (プロバイダを追加) を選択します。



3. [Add an Identity provider] (ID プロバイダ) ページで、[Configure provider] (プロバイダの設定) に以下の情報を入力します。
   - [プロバイダのタイプ] では [SAML] を選択します。
   - [Provider name] (プロバイダ名) に、プロバイダの名前を入力します (例: **`AthenaODBCOkta`**)。
   - [メタデータドキュメント] では、[ファイルを選択] オプションを使用して、ダウンロードした ID プロバイダ (IdP) メタデータ XML ファイルをアップロードします。



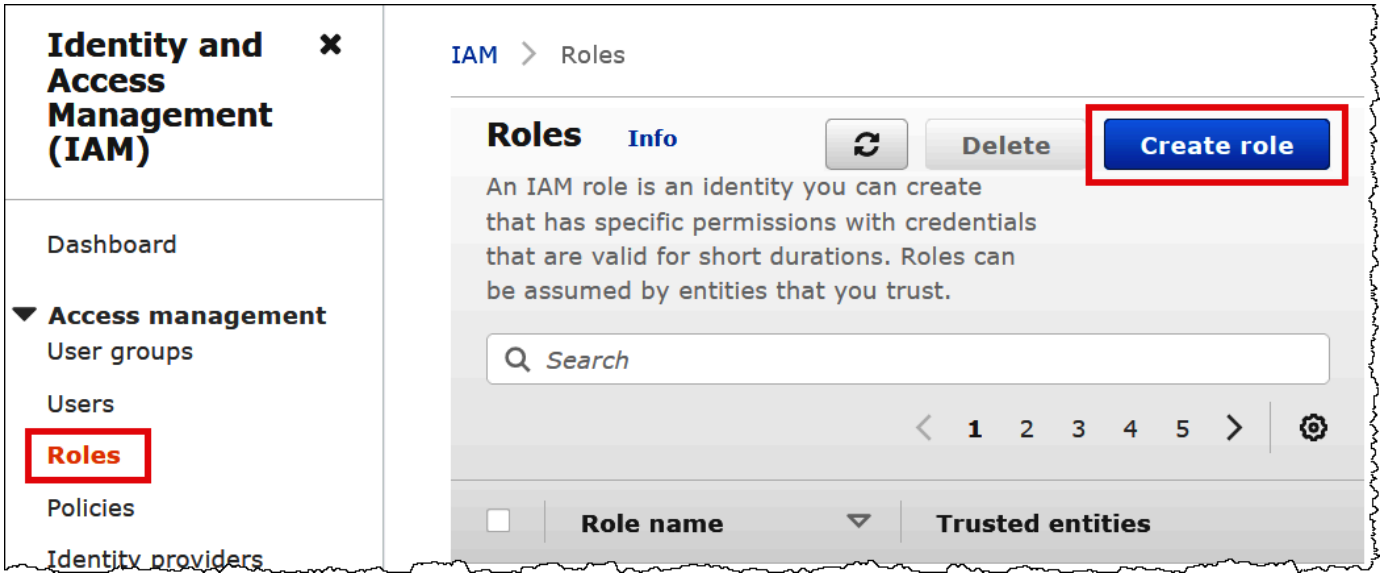
4. [プロバイダーを追加] をクリックします。

Athena および Amazon S3 にアクセスするための IAM ロールを作成する

Athena および Amazon S3 にアクセスするための IAM ロールを作成する準備ができました。このロールをユーザーに割り当てます。これにより、ユーザーに Athena へのシングルサインオンアクセスを提供できます。

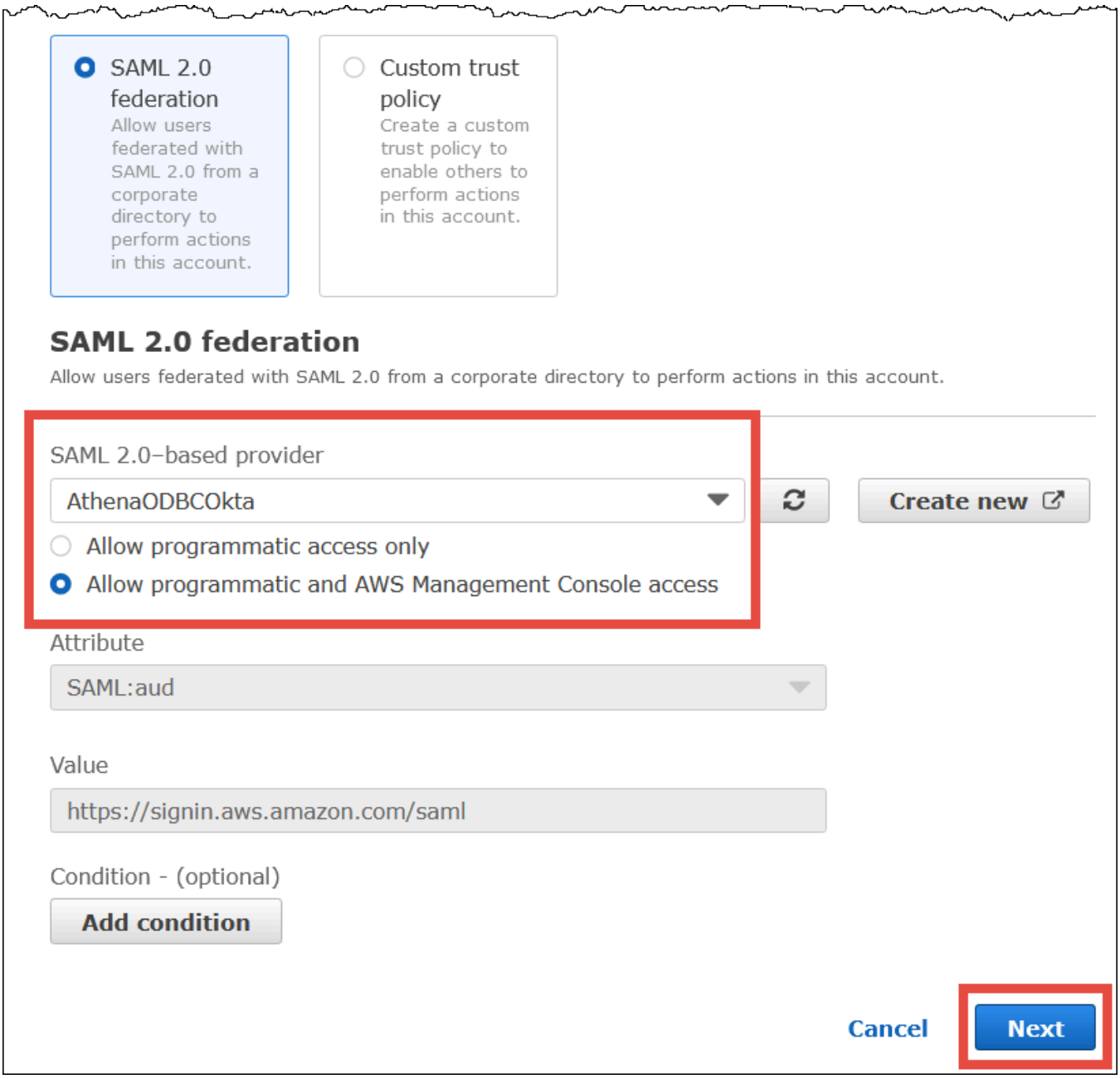
ユーザー用の IAM ロールを作成するには

1. IAM コンソールのナビゲーションペインで、[Roles] (ロール)、[Create role] (ロールの作成) の順にクリックします。

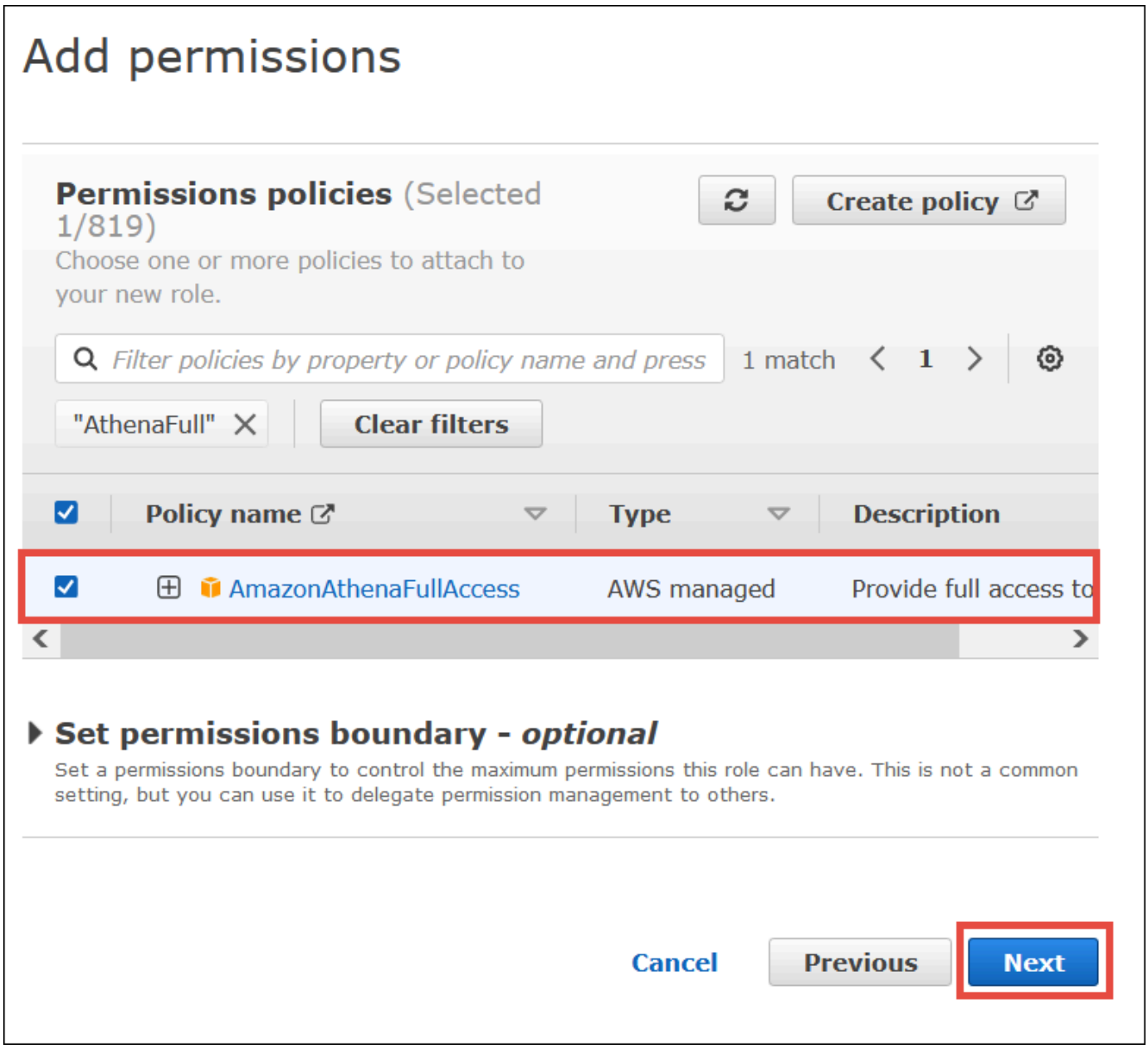


2. [Create role] (ロールを作成) ページで、次のオプションを選択します。

   - [Select type of trusted entity] (信頼されたエンティティのタイプを選択) で、[SAML 2.0 Federation] (SAML 2.0 フェデレーション) を選択します。
   - [SAML 2.0–based provider] (SAML 2.0 ベースのプロバイダ) で、作成した SAML ID プロバイダを指定します (たとえば、AthenaODBCOkta)。
   - [プログラムによるアクセスとコンソールによるアクセスを許可] を選択します。



3. [Next] を選択します。
4. [Add Permissions] (許可を追加) ページの [Filter policies] (ポリシーをフィルタリング) に、**`AthenaFull`** と入力してから Enter キーを押します。
5. `AmazonAthenaFullAccess` 管理ポリシーを選択してから、[Next] (次へ) を選択します。



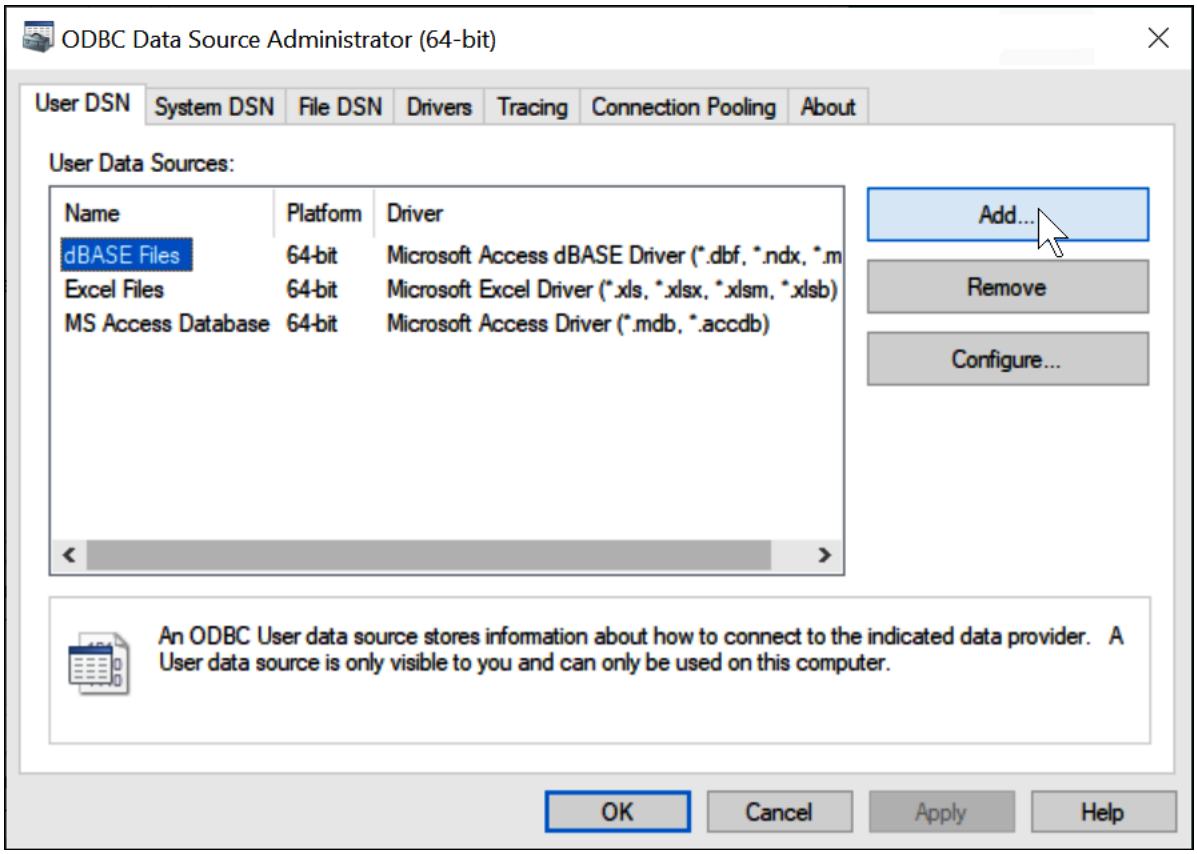
6. [Name, review, and create] (名前、確認、および作成) ページで、[Role name] (ロール名) にロールの名前 (たとえば、**`Athena-ODBC-OktaRole`**) を入力し、[Create role] (ロールを作成) を選択します。

Athena への Okta ODBC 接続を設定する

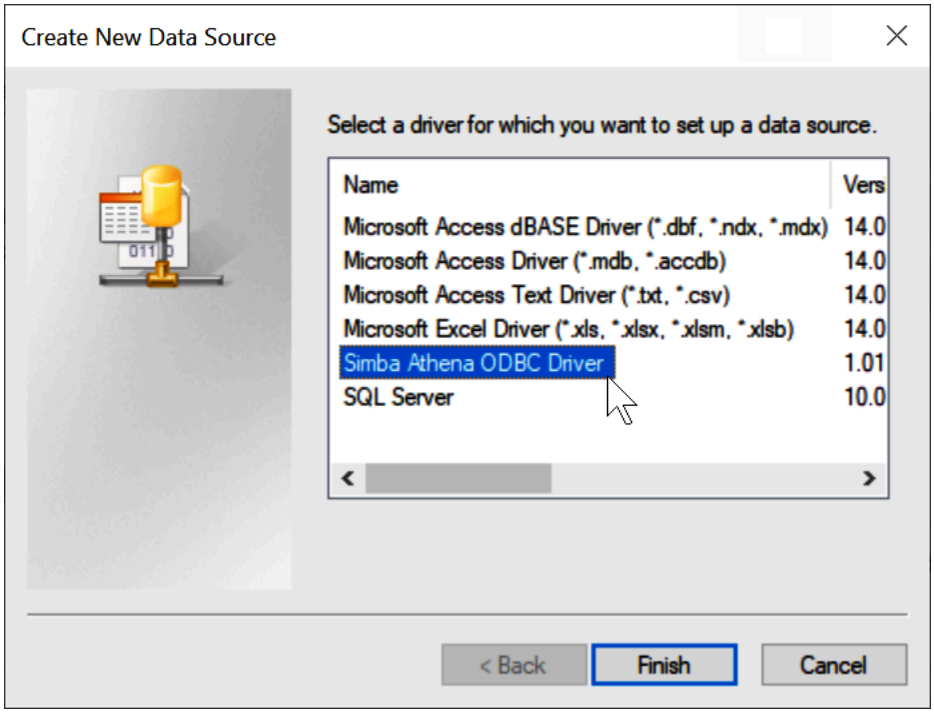
Windows の ODBC データソースプログラムを使用して、Athena への Okta ODBC 接続を設定する準備ができました。

Athena への Okta ODBC 接続を設定するには

1. Windows で、[ODBC Data Sources] (ODBC データソース) プログラムを起動します。
2. [ODBC Data Source Administrator] (ODBC データソースの管理者) プログラムで、[Add] (追加) をクリックします。

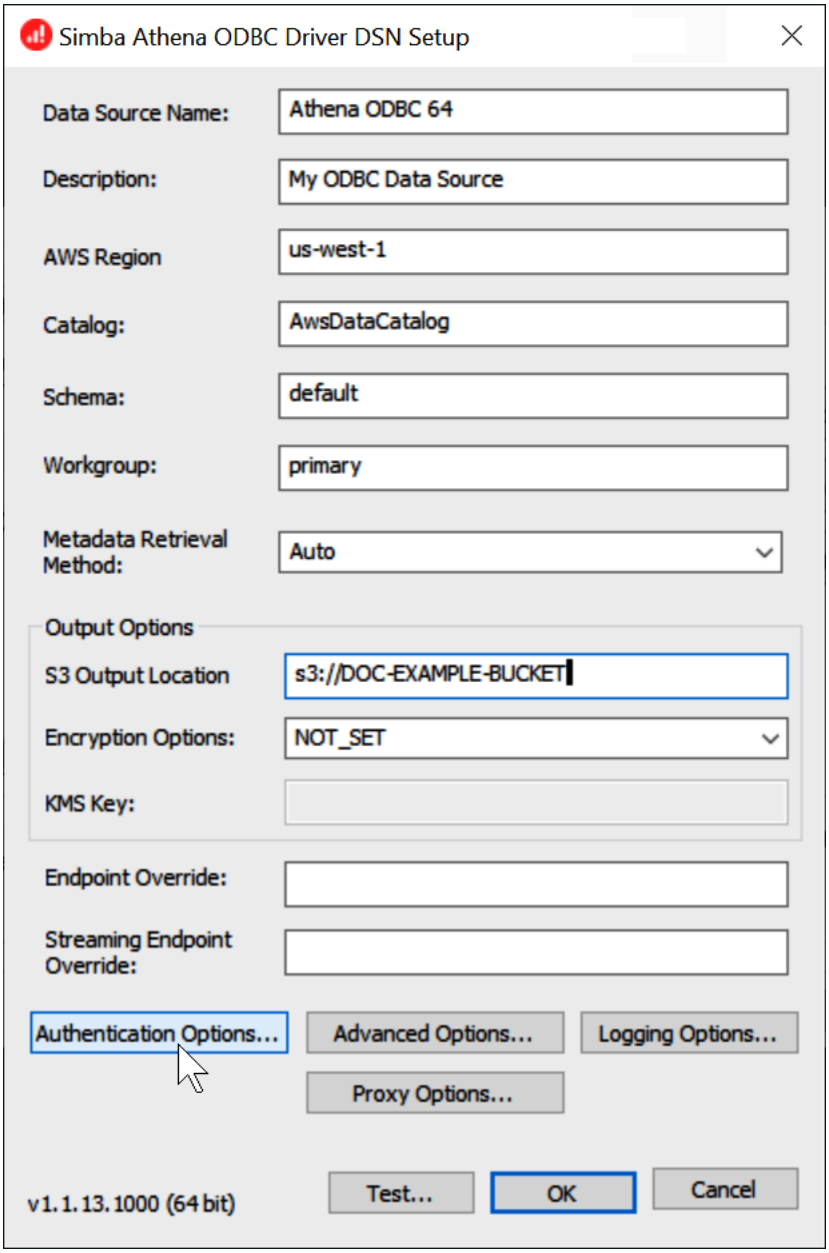


3. [Simba Athena ODBC Driver] (Simba Athena ODBC ドライバー) を選択し、[Finish] (完了) をクリックします。



4. 左[Simba Athena ODBC Driver DSN Setup] (Simba Athena ODBC ドライバーの DSN セットアップ) ダイアログで、以下に記述されている値を入力します。

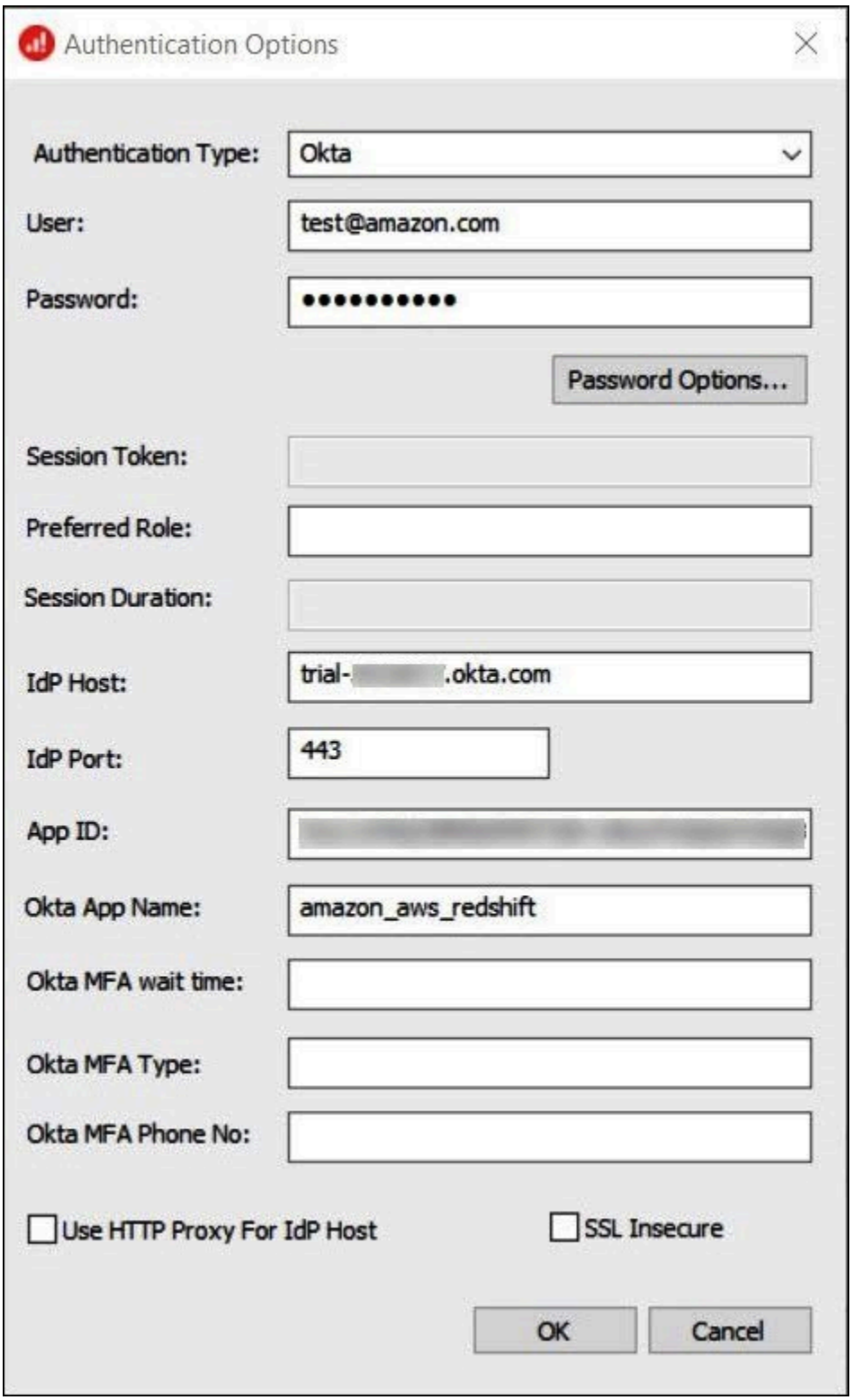
   - [Data Source Name] (データソース名) に、使用するデータソースの名前 (例えば、**`Athena ODBC 64`**) を入力します。
   - [Description] (説明) に、データソースの説明を入力します。
   - [AWS リージョン] に、使用している AWS リージョン (たとえば、**`us-west-1`**) を入力します。
   - [S3 Output Location] (S3 出力場所) には、出力を保存する先の Amazon S3 パスを入力します。



5. [Authentication Options] (認証オプション) をクリックします。
6. [Authentication Options] (認証オプション) ダイアログボックスで、以下の値を選択または入力します。

   - [Authentication Type] (認証タイプ) で、[Okta] を選択します。
   - ユーザーには、Okta ユーザー名を入力します。
   - パスワードには、Okta パスワードを入力します。
   - [IdP Host] (IdP ホスト) に、以前記録した値を入力します (たとえば、**`trial-1234567.okta.com`**)。

- [IdP Port] (IdP ポート) に、**443** と入力します。
- [App ID] (アプリケーション ID) に、以前記録した値 (Okta 埋め込みリンクに含まれる最後の 2 つのセグメント) を入力します。
- [Okta App Name] (Okta アプリケーション名) に、**amazon_aws_redshift** と入力します。



7. [OK] を選択します。
8. 接続をテストするには [Test] (テスト) を、終了するには [OK] を選択します。

ODBC、SAML 2.0、および Okta ID プロバイダを使用したシングルサインオンの設定

データソースに接続する際、 Amazon Athena では、PingOne、Okta、OneLogin を初めとしたアイデンティティプロバイダ (IdPs) を使用できます。Athena ODBC ドライバーのバージョン 1.1.13 および Athena JDBC ドライバーのバージョン 2.0.25 以降には、ブラウザの SAML プラグインが含まれており、任意の SAML 2.0 プロバイダで動作するように設定できます。このトピックでは、Okta の ID プロバイダを使用して Single Sign-On (SSO) 機能を追加するために、Amazon Athena ODBC ドライバーとブラウザベースの SAML プラグインを設定する方法について説明します。

前提条件

このチュートリアルのステップを完了するには、以下が必要です。

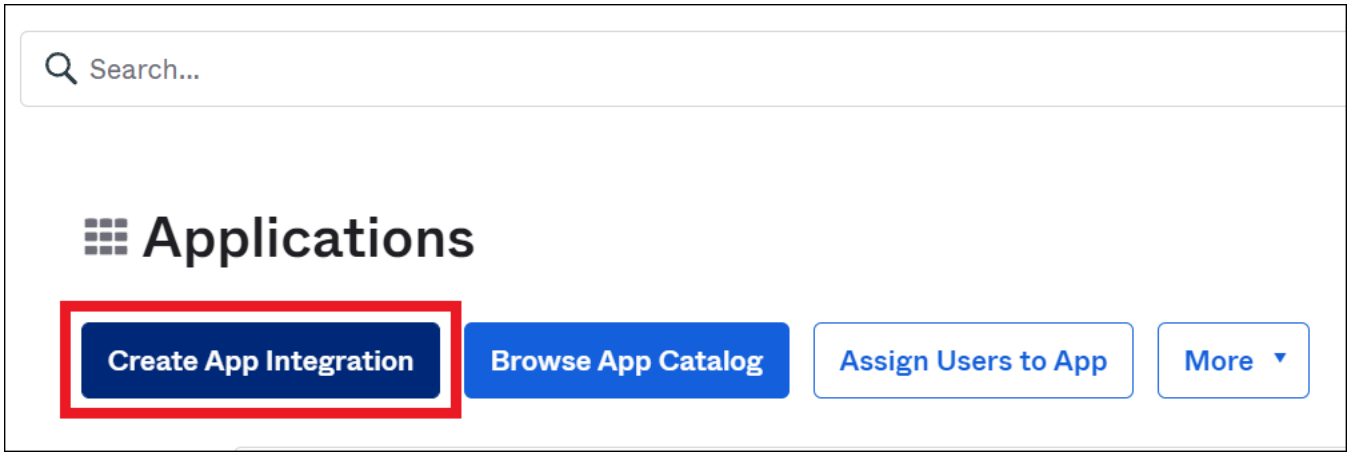
- Athena ODBC ドライバーバージョン 1.1.13 以降。バージョン 1.1.13 以降には、ブラウザの SAML サポートが含まれています。ダウンロード用のリンクは、「ODBC を使用した Amazon Athena への接続」でご確認ください。
- SAML で使用する IAM ロール。詳細については、「IAM ユーザーガイド」の「SAML 2.0 フェデレーション用のロールの作成 (コンソール)」を参照してください。
- Okta アカウント。詳細については、okta.com を参照してください。

Okta でのアプリケーション統合の作成

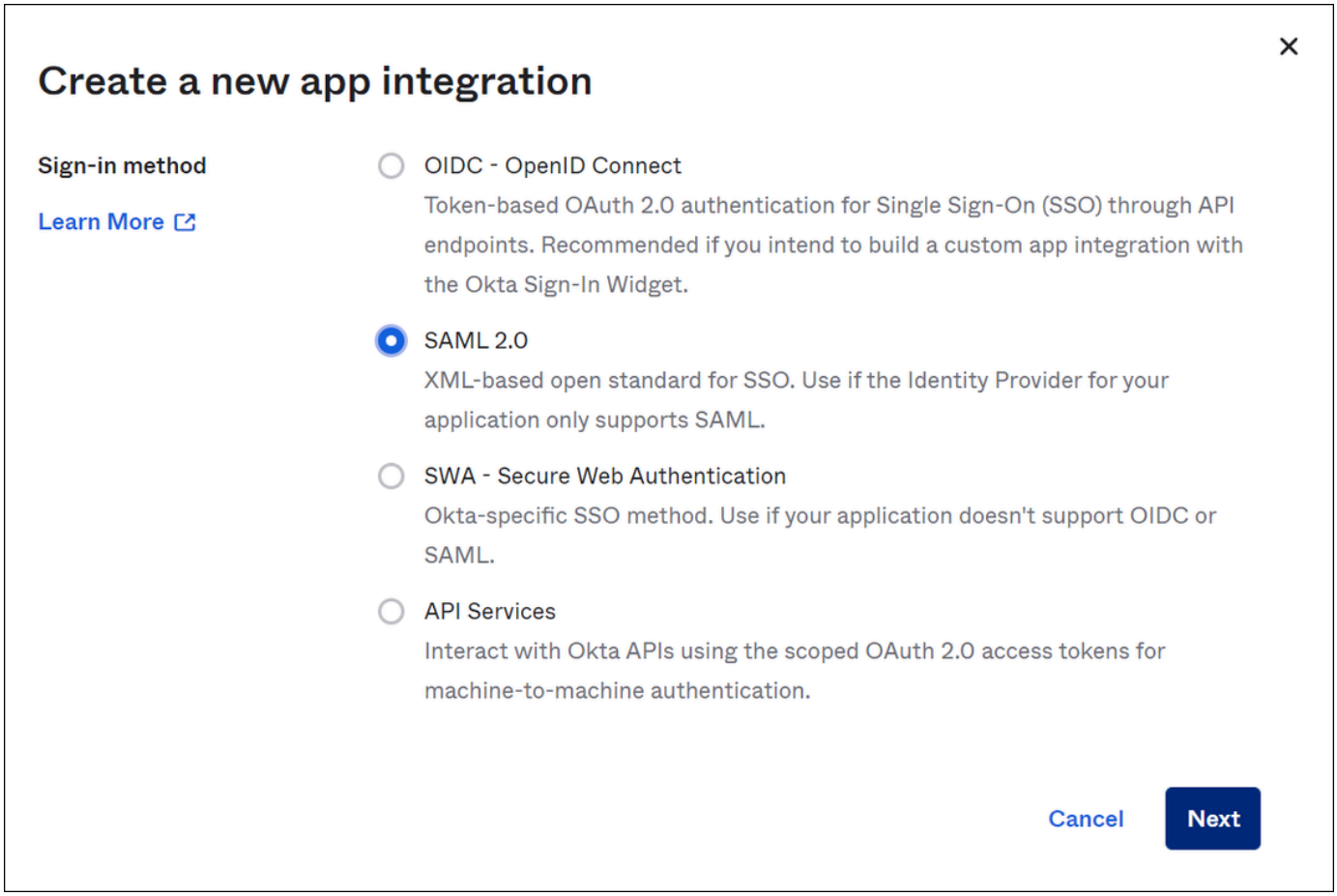
まず、Okta ダッシュボードを使用して、Athena へのシングルサインオン用の SAML 2.0 アプリケーションを作成して設定します。

Okta ダッシュボードを使用して Athena のシングルサインオンをセットアップするには

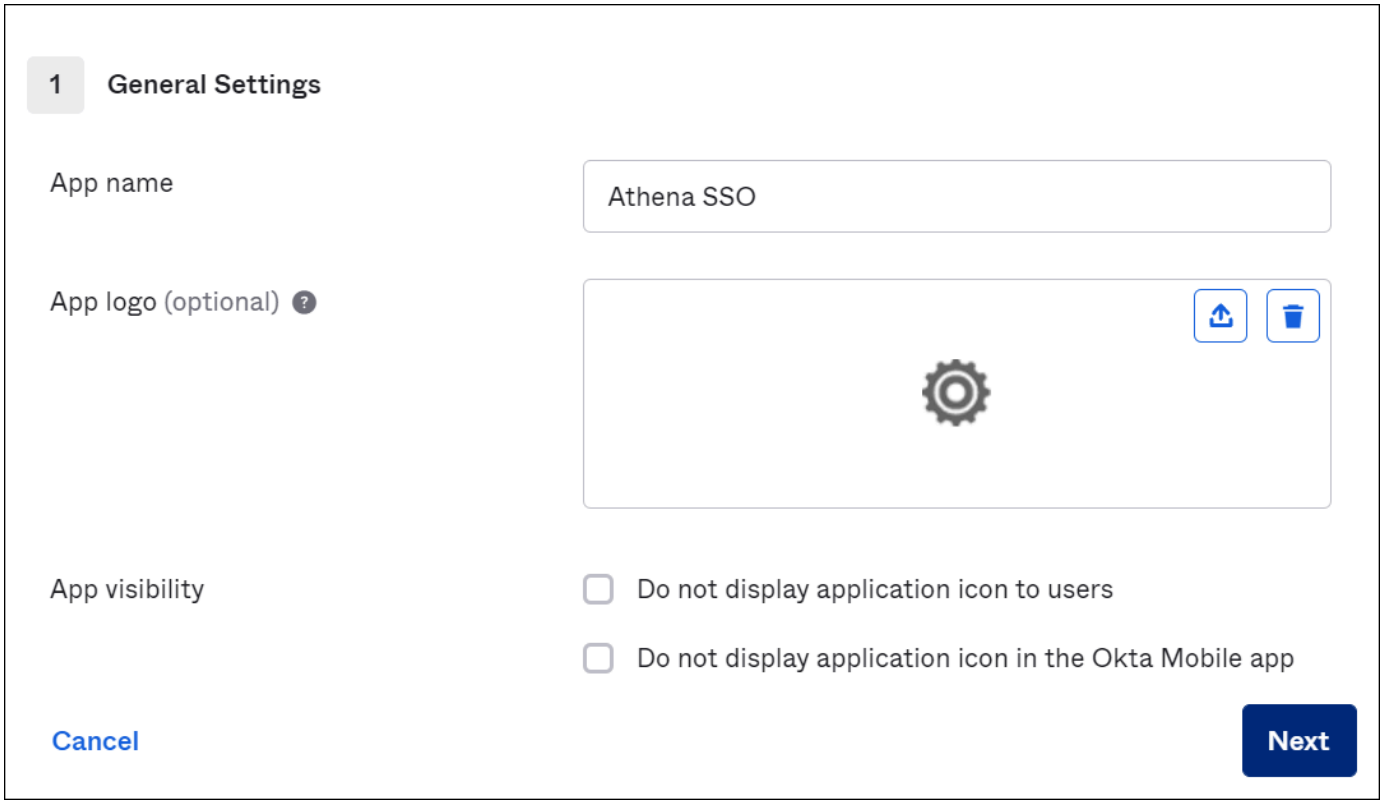
1. `okta.com` の Okta 管理ページにログインします。
2. [ナビゲーションペイン] で、[Applications] (アプリケーション) をクリックした後に、もう一度 [Applications] をクリックします。
3. [Applications] (アプリケーション) ページで、[Create App Integration] (アプリケーション統合を作成) をクリックします。



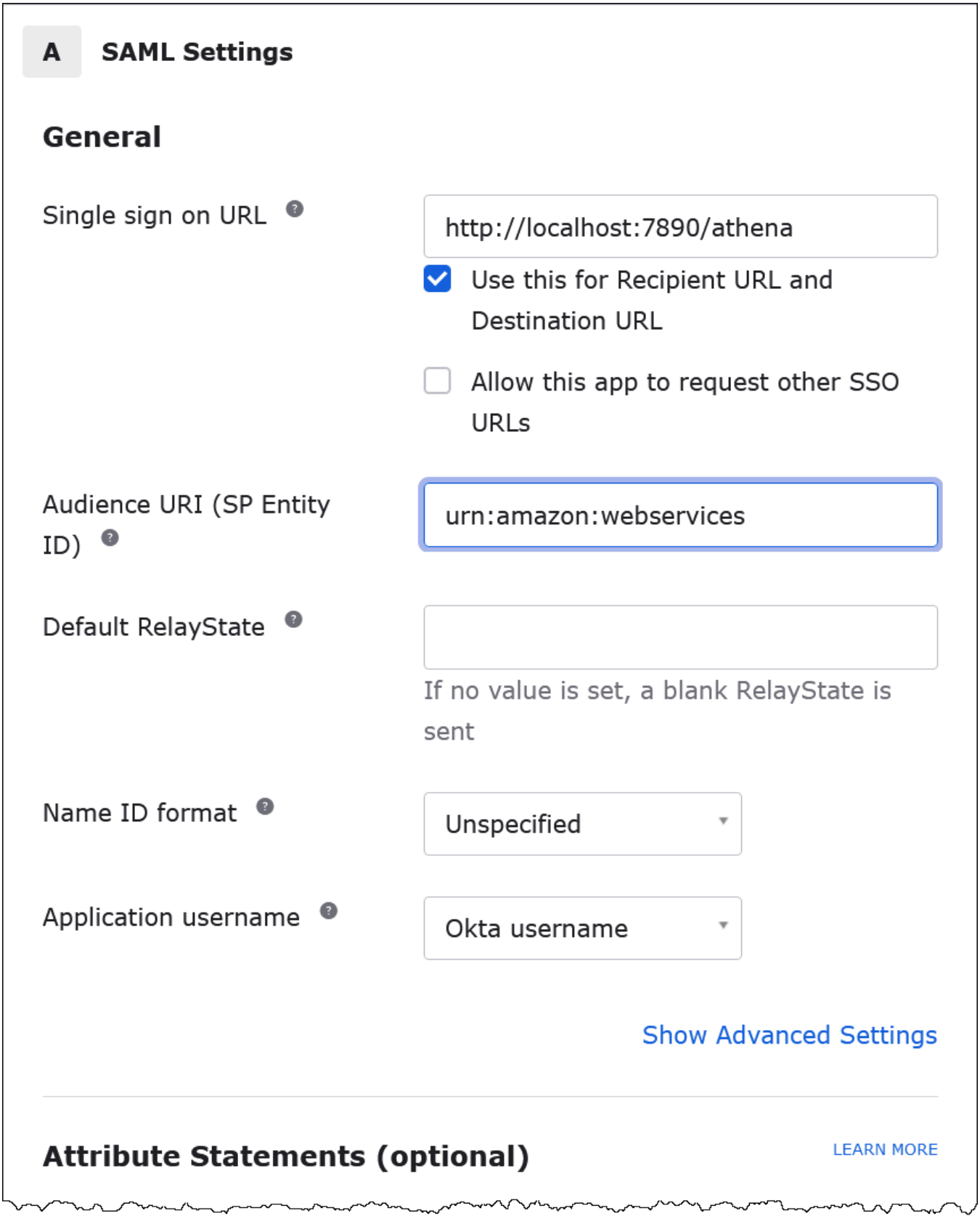
4. [Create a new app integration] (新しいアプリケーション統合の作成) ダイアログボックスにある [Sign-in method] (サインイン方法) で、[SAML 2.0] を選択した上で、[Next] (次へ) をクリックします。



5. [Create SAML Integration] (SAML 統合の作成) ページの [General Settings] (全般設定) セクションで、アプリケーションの名前を入力します。このチュートリアルでは、この名前として [Athena SSO] を使用します。



6. [Next] を選択します。
7. [Configure SAML] (SAML の設定) ページの [SAML Settings] (SAML 設定) セクションで、以下の値を入力します。
   - [Single sign on URL] (シングルサインオン URL) には **`http://localhost:7890/athena`** を入力します。
   - [Audience URI] (対象者の URI) には **`urn:amazon:webservices`** を入力します。


A
SAML Settings
General
Single sign on URL
http://localhost:7890/athena
Use this for Recipient URL and Destination URL
Allow this app to request other SSO URLs
Audience URI (SP Entity ID)
urn:amazon:webservices
Default RelayState
If no value is set, a blank RelayState is sent
Name ID format
Unspecified
Application username
Okta username
Show Advanced Settings
Attribute Statements (optional)
LEARN MORE

8. [Attribute Statements (optional)] (属性ステートメント (オプション)) には、次の名前と値のペアを入力します。これらのマッピング属性は必須です。

   - [Name] (名前) に、次の URL を入力します。

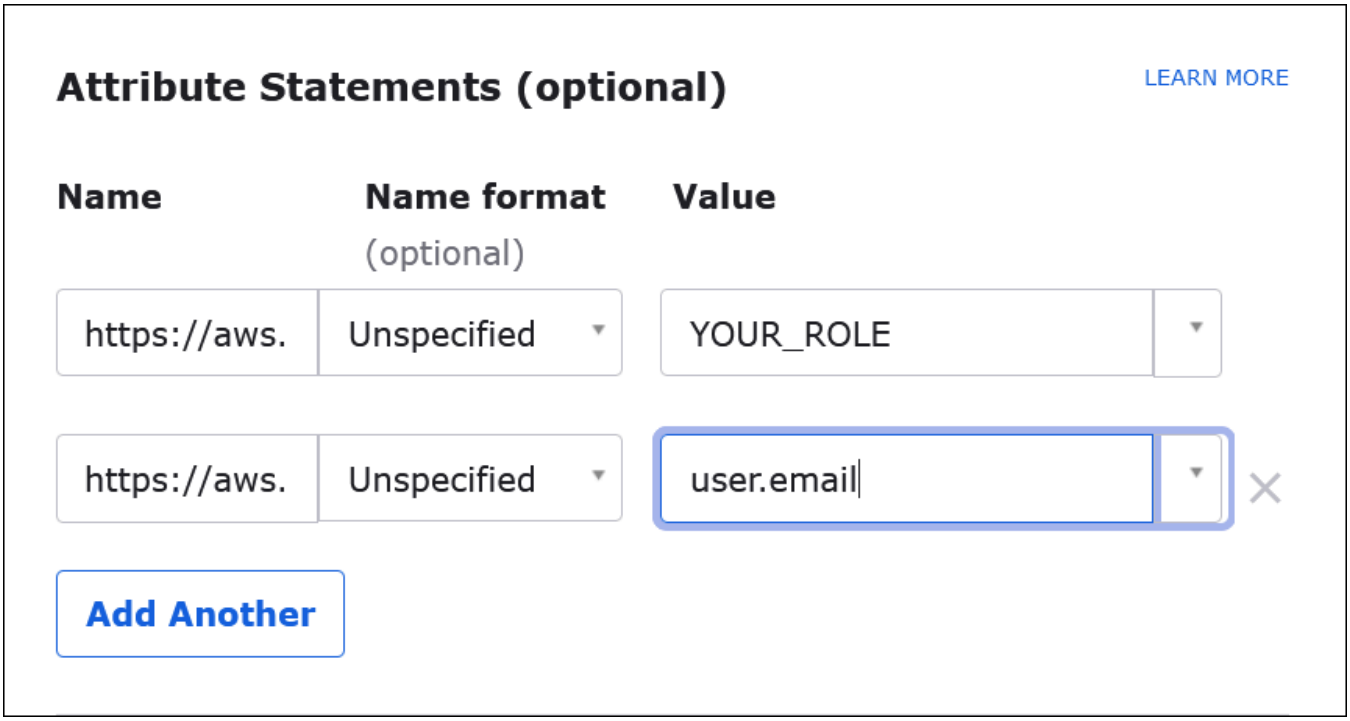
     `https://aws.amazon.com/SAML/Attributes/Role`

     [Value] (値) に、IAM ロールの名前を入力します。IAM ロール形式の詳細については、「IAM ユーザーガイド」の「認証レスポンスの SAML アサーションを設定する」を参照してください。

   - [Name] (名前) に、次の URL を入力します。

     `https://aws.amazon.com/SAML/Attributes/RoleSessionName`

     [値] に「**`user.email`**」と入力します。



9. [Next] (次へ) を選択し、[Finish] (完了) を選択します。

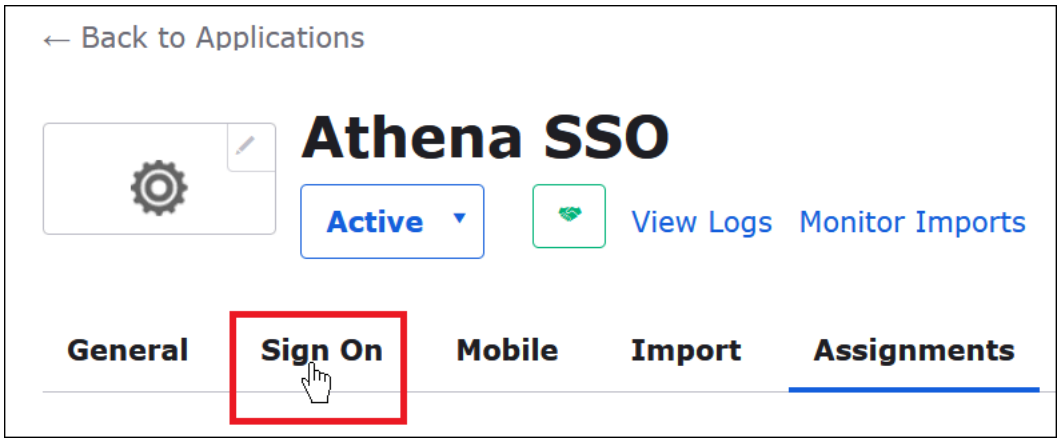
   Okta がアプリケーションを作成する際、同時にログイン URL も作成され、これは次で取得できます。

Okta ダッシュボードからのログイン URL の取得

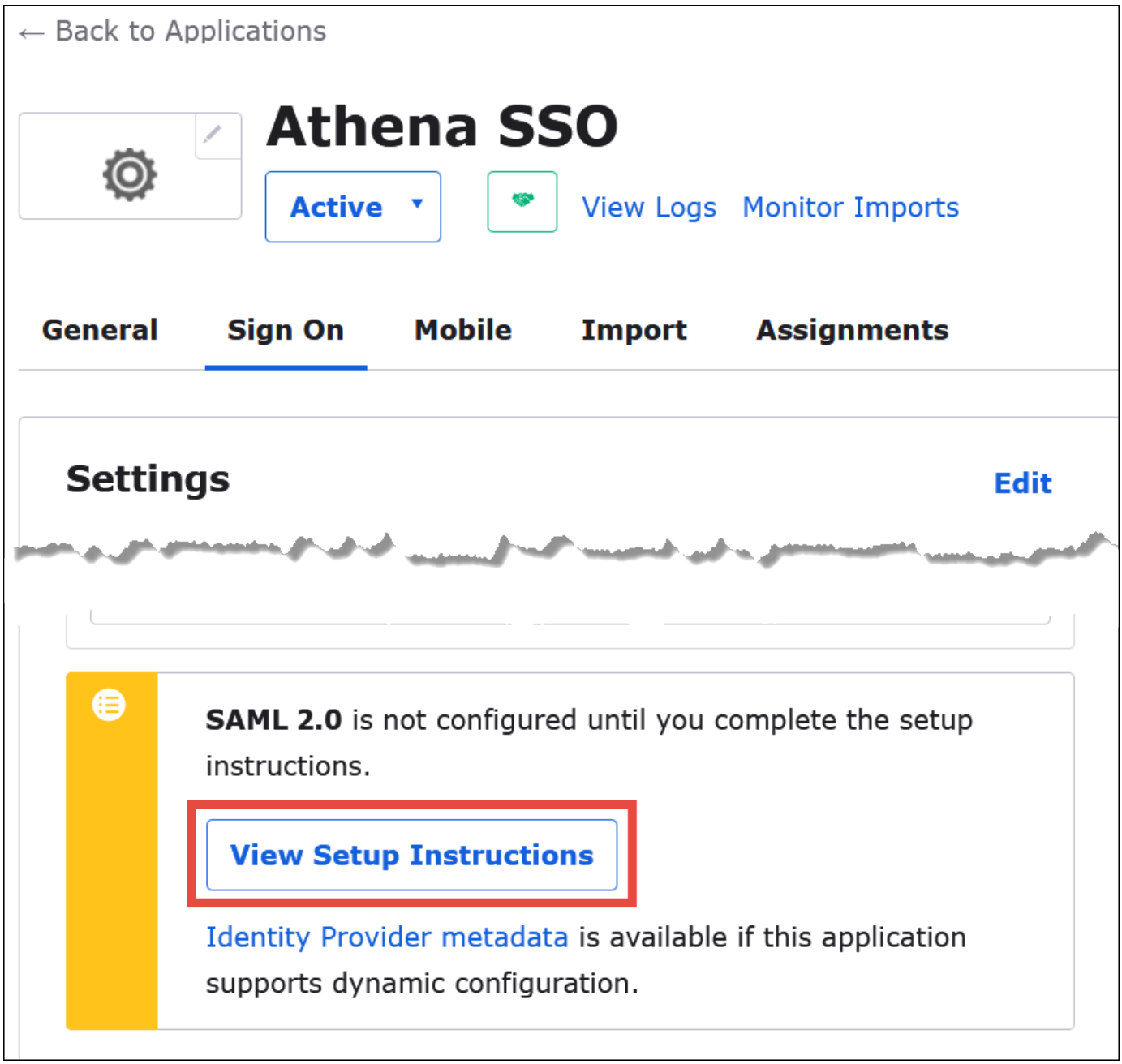
この段階まででアプリケーションが作成されたので、Okta ダッシュボードからログイン URL およびその他のメタデータを取得できます。

Okta ダッシュボードからログイン URL を取得するには

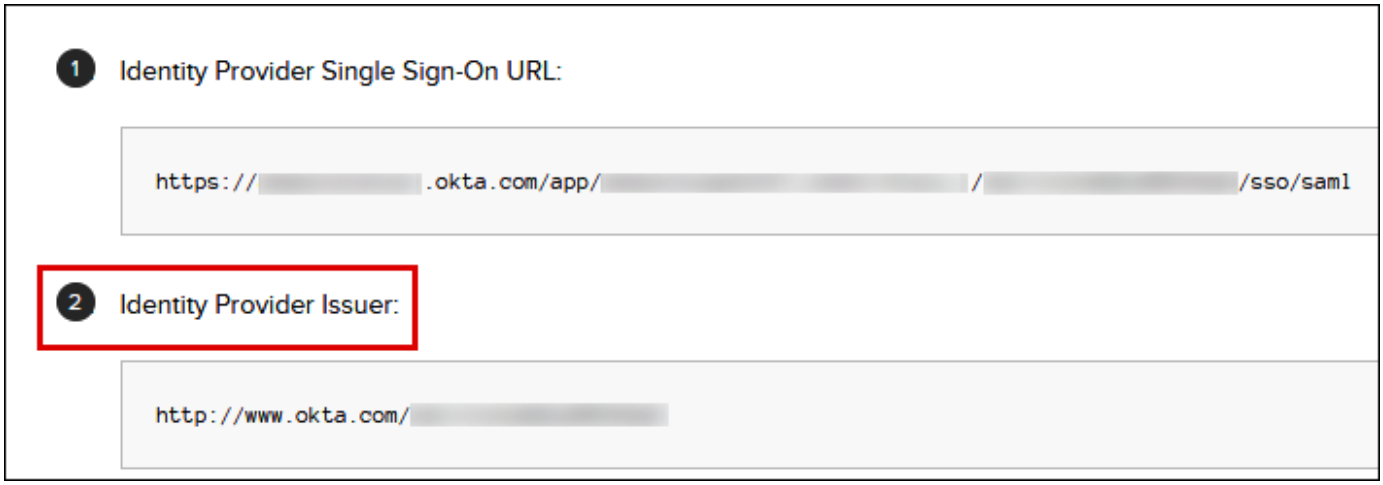
1. Okta のナビゲーションペインで、[Applications] (アプリケーション) をクリックした後、もう一度 [Applications] をクリックします。
2. ログイン URL を確認するアプリケーション (たとえば、AthenASSO) を選択します。
3. アプリケーションのページで、[Sign On] (サインオン) をクリックします。



4. [View Setup Instructions] (セットアップ手順を表示) をクリックします。



5. [How to Configure SAML 2.0 for Athena SSO] (Athena SSO 用の SAML 2.0 を設定するには) ページで、[Identity Provider Issuer] (ID プロバイダの発行者) の URL を探します。Okta ダッシュボード内の一部では、この URL を [SAML issuer ID] (SAML 発行者 ID) として参照します。



6. ID プロバイダシングルサインオン URL の値をコピーまたは保存します。

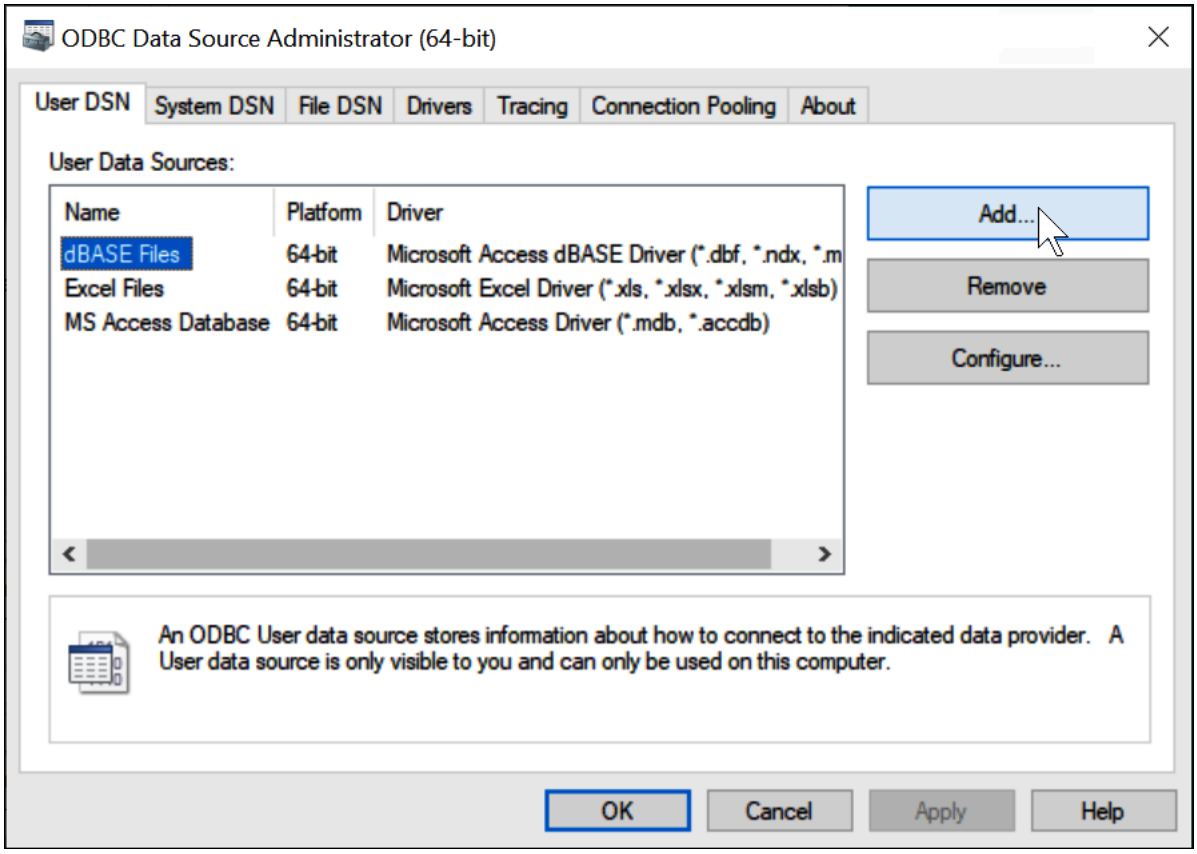
   次のセクションで ODBC 接続を設定する際に、ブラウザの SAML プラグイン用の接続パラメータ [Login URL] (ログインURL) として、この値を指定します。

ブラウザ SAML ODBC による Athena への接続の設定

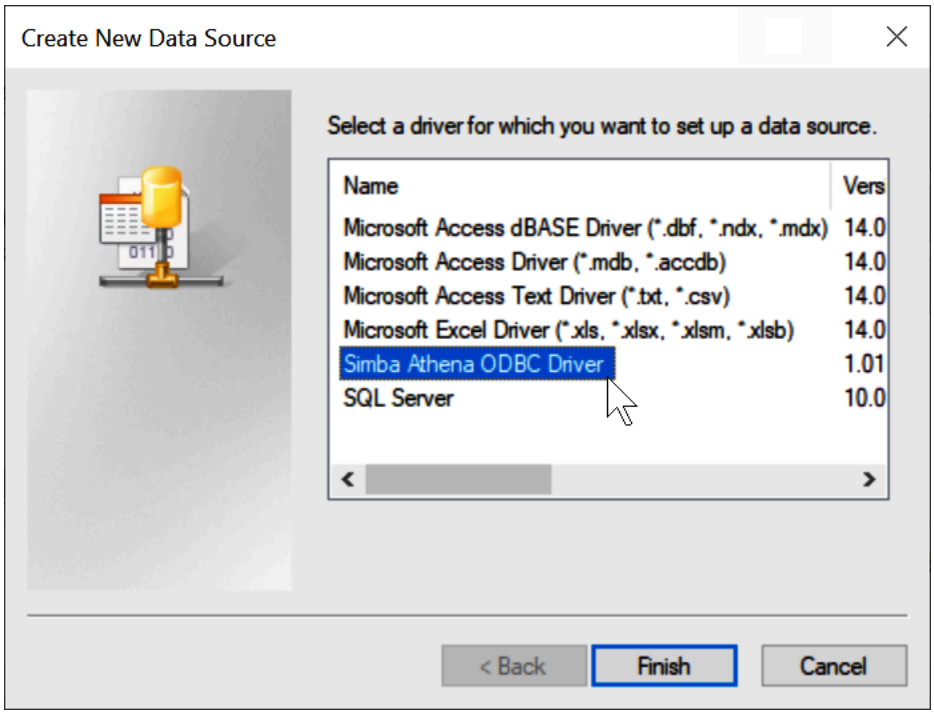
この段階で、Windows の ODBC データソースプログラムを使用して、Athena へのブラウザの SAML 接続を設定するための準備が整いました。

ブラウザの SAML ODBC による Athena への接続を設定するには

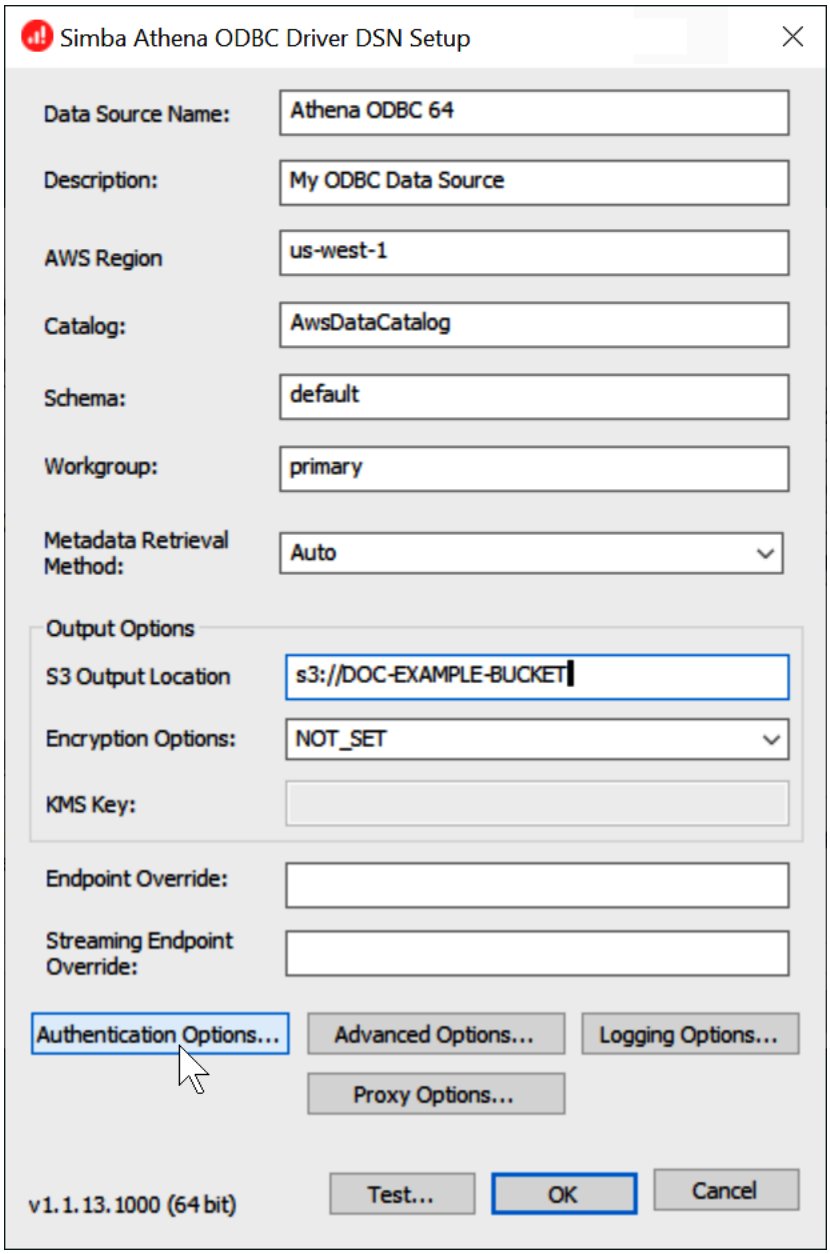
1. Windows で、[ODBC Data Sources] (ODBC データソース) プログラムを起動します。
2. [ODBC Data Source Administrator] (ODBC データソースの管理者) プログラムで、[Add] (追加) をクリックします。



3. [Simba Athena ODBC Driver] (Simba Athena ODBC ドライバー) を選択し、[Finish] (完了) をクリックします。

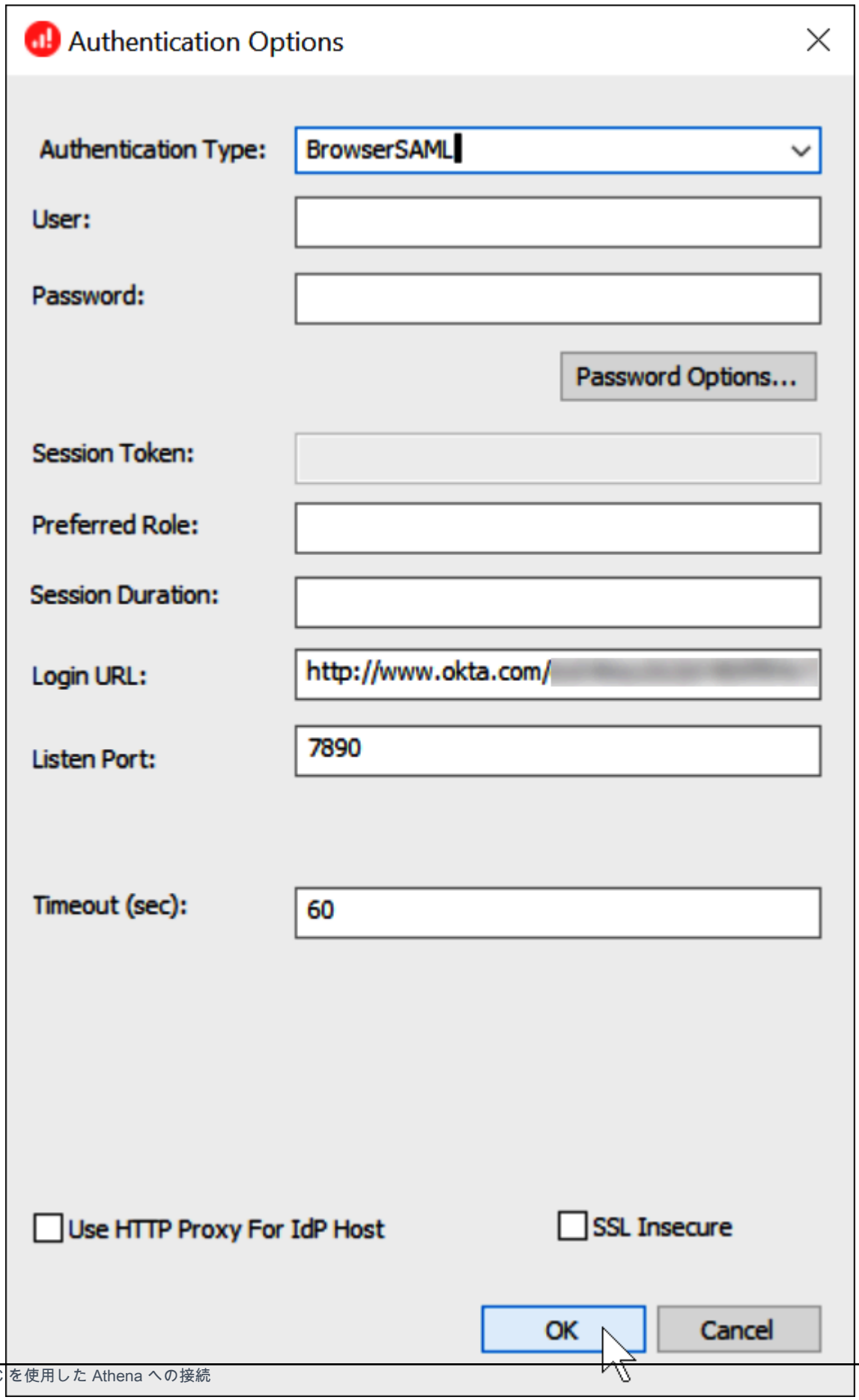


4. 左[Simba Athena ODBC Driver DSN Setup] (Simba Athena ODBC ドライバーの DSN セットアップ) ダイアログで、以下に記述されている値を入力します。



   - [Data Source Name] (データソース名) には、使用するデータソースの名前 (例:Athena ODBC 64) を入力します。
   - [Description] (説明) に、データソースの説明を入力します。
   - [AWS リージョン] には、使用している AWS リージョン (例: **`us-west-1`**) を入力します
   - [S3 Output Location] (S3 出力場所) には、出力を保存する先の Amazon S3 パスを入力します。
5. [Authentication Options] (認証オプション) をクリックします。

6. [Authentication Options] (認証オプション) ダイアログボックスで、以下の値を選択または入力します。


Authentication Options
Authentication Type:
BrowserSAML
User:
Password:
Password Options...
Session Token:
Preferred Role:
Session Duration:
Login URL:
http://www.okta.com/
Listen Port:
7890
Timeout (sec):
60
Use HTTP Proxy For IdP Host
SSL Insecure
OK
Cancel

- [Authentication Type] (認証タイプ) で、[BrowserSAML] を選択します。
- [Login URL] (ログインURL) には、Okta ダッシュボードから取得した、[Identity Provider Single Sign-On URL] (ID プロバイダのシングルサインオン URL) を入力します。
- [Listen Port] (リッスンするポート) には、「7890」と入力します。
- [Timeout (sec)] (タイムアウト (秒)) に、接続のタイムアウト値を秒単位で入力します。

7. [OK] をクリックして、[Authentication Options] (認証オプション) を閉じます。
8. 接続をテストするには [Test] (テスト) を、終了するには [OK] をクリックします。

Amazon Athena Power BI コネクタの使用

Windows オペレーティングシステムでは、Amazon Athena 向けの Microsoft Power BI コネクタを使用して、Microsoft Power BI Desktop で Amazon Athena からのデータを分析することができます。Power BI の詳細については、「Microsoft Power BI」を参照してください。Power BI サービスにコンテンツを公開した後は、2021 年 7 月以降にリリースされた Power BI ゲートウェイを使用して、オンデマンドまたはスケジュールされた更新によって、コンテンツを最新の状態に保つことができます。

前提条件

使用を開始する前に、環境が次の要件を満たしていることを確認してください。Amazon Athena ODBC ドライバーが必要です。

- AWS アカウント
- Athena を使用するためのアクセス許可
- Amazon Athena ODBC ドライバー
- Power BI Desktop

サポートされる機能

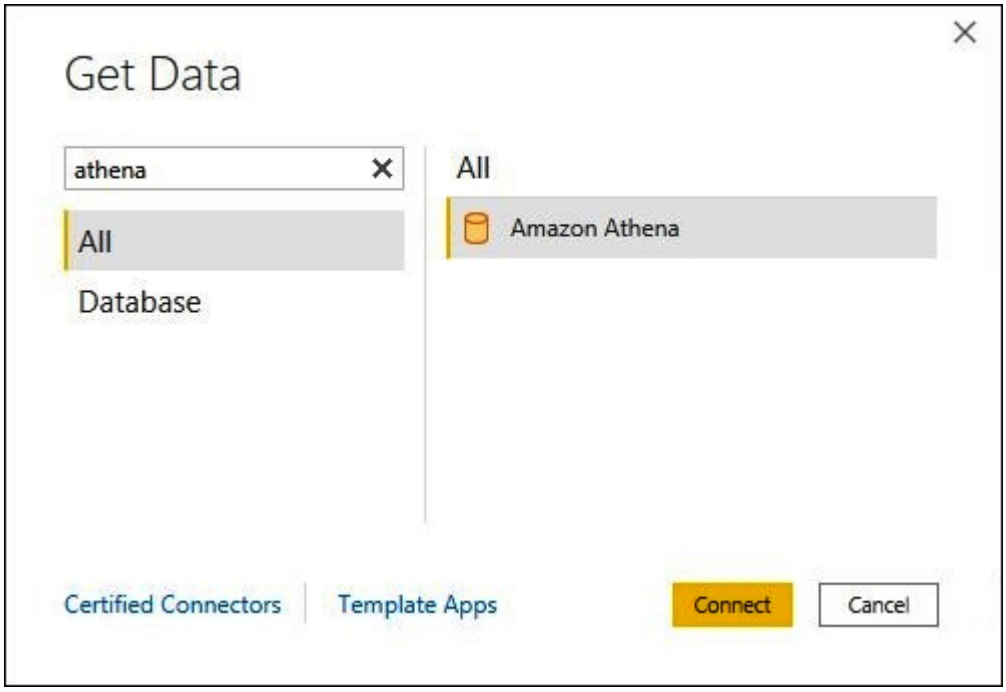
- インポート - 選択したテーブルと列がクエリ実行用に Power BI Desktop にインポートされます。
- DirectQuery - Power BI Desktop にデータがインポートまたはコピーされることはありません。Power BI Desktop が、基盤となるデータソースに直接クエリを実行します。
- Power BI ゲートウェイ – Microsoft Power BI サービスと Athena の間の橋渡しのように機能する、AWS アカウント 内のオンプレミスデータゲートウェイです。このゲートウェイは、Microsoft Power BI サービスでデータを表示するために必要です。

Amazon Athena への接続

Power BI Desktop を Amazon Athena データに接続するには、次の手順を実行します。

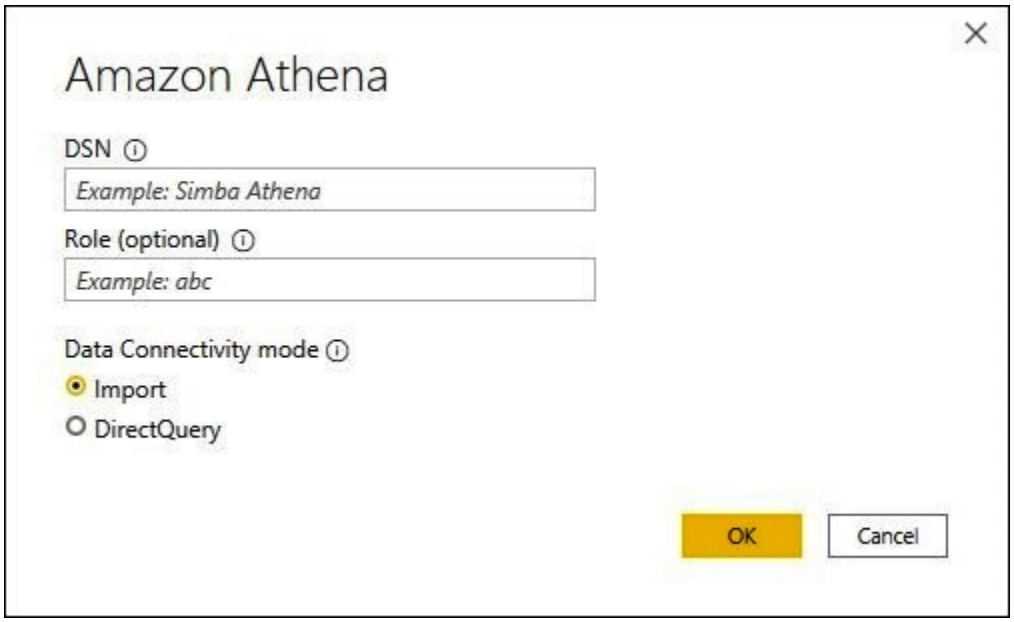
Power BI Desktop から Athena データに接続する

1. Power BI Desktop を起動します。
2. 次のいずれかを行ってください。
   - [File] (ファイル) 、[Get Data] (データを取得) の順に選択します。
   - [Home] (ホーム) リボンから、[Get Data] (データを取得) を選択します。
3. 検索ボックスに「Athena」と入力します。
4. [Amazon Athena] を選択してから、[Connect] (接続) をクリックします。

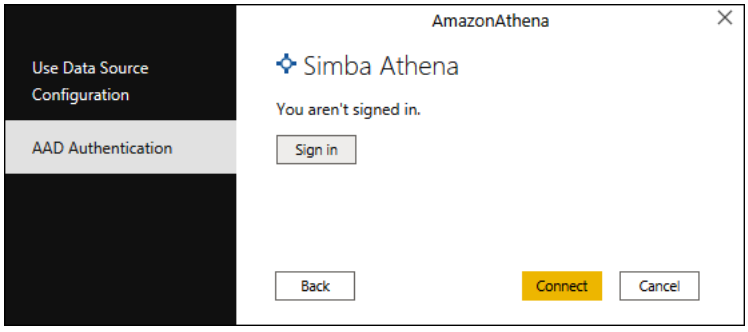


5. Amazon Athena 接続ページで、次の情報を入力します。
   - [DSN] では、使用する ODBC DSN の名前を入力します。DSN の設定手順については、ODBC ドライバーのドキュメントをご覧ください。
   - [Data Connectivity] (データ接続モード) では、次の一般的なガイドラインに従って、ユースケースに適したモードを選択します。
     - 小さいデータセットの場合は、[インポート] を選択します。インポートモードを使用する場合、Power BI が Athena と連携して、可視化に使用するデータセット全体のコンテンツをインポートします。

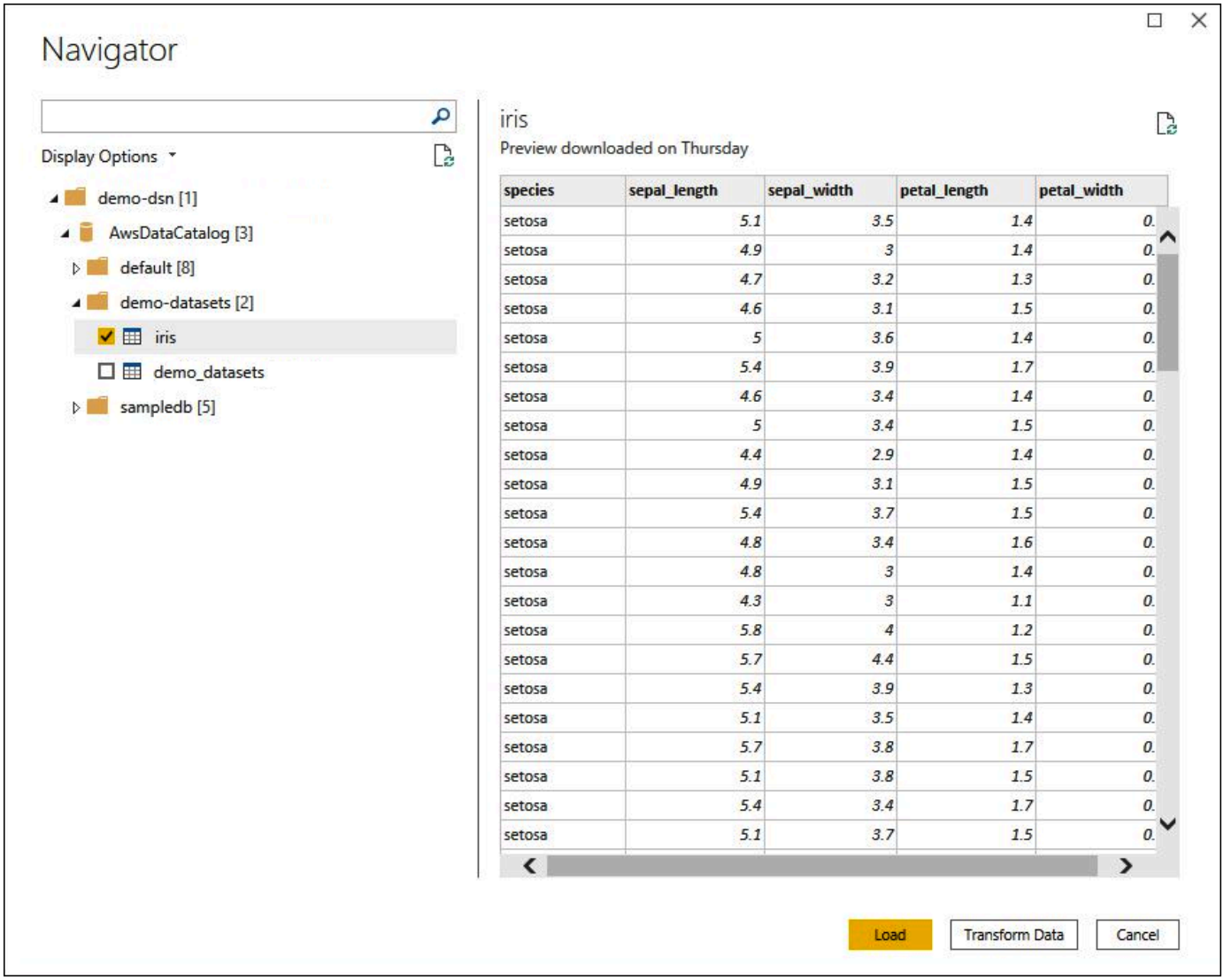
- 大規模なデータセットの場合は、[DirectQuery] を選択します。DirectQuery モードでは、データがワークステーションにダウンロードされることはありません。可視化を作成または操作している間、Microsoft Power BI が Athena と連携して基盤となるデータソースに動的にクエリを実行するため、常に最新のデータが表示されます。DirectQuery の詳細については、Microsoft のドキュメントの「Power BI Desktop の DirectQuery を使用する」を参照してください。



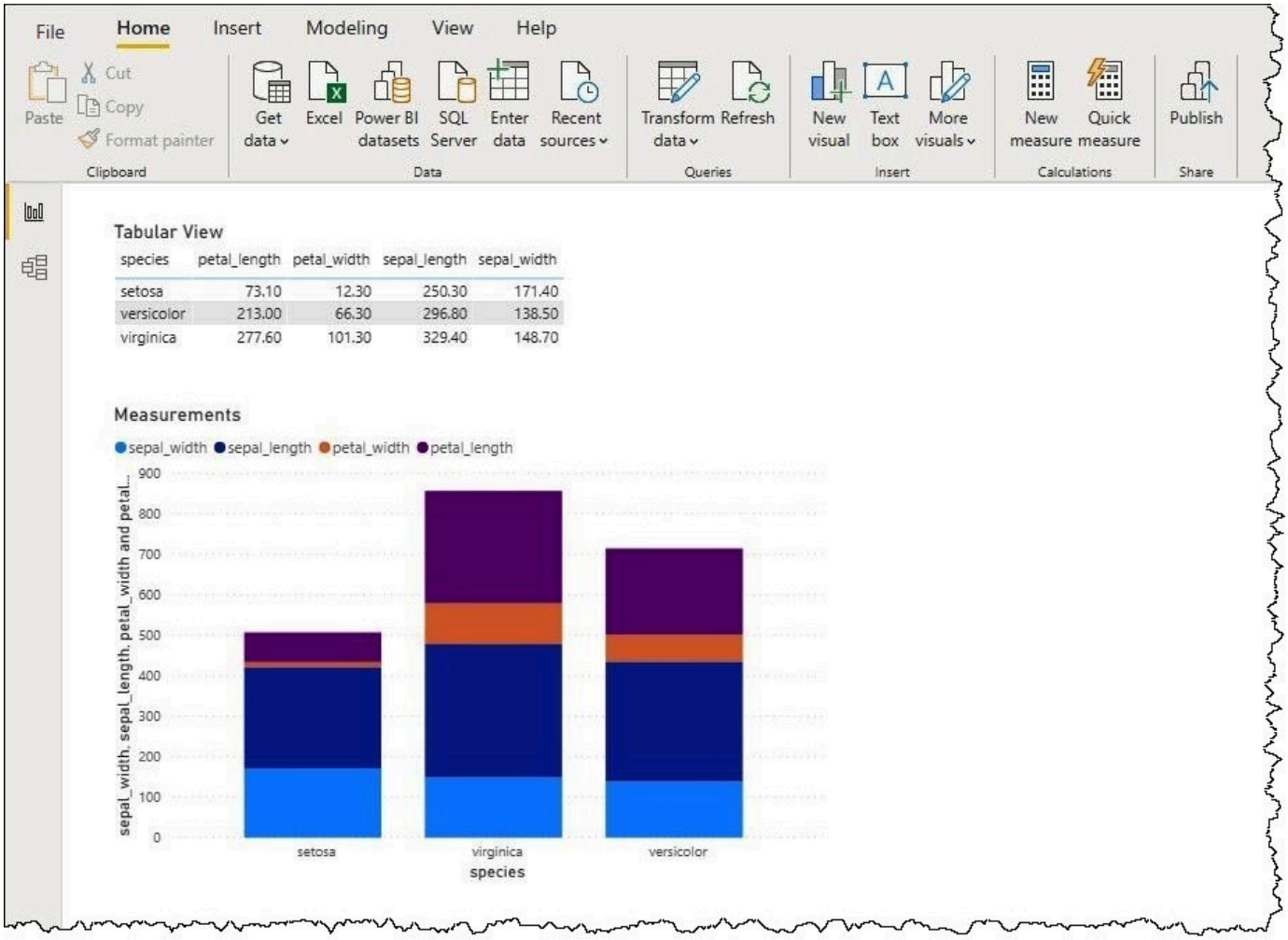
6. [OK] を選択します。
7. データソース認証を設定するプロンプトで、[Use Data Source Configuration] (データソース設定を使用) または [AAD Authentication] (AAD 認証) を選択してから、[Connect] (接続) をクリックします。



データカタログ、データベース、およびテーブルが [Navigator] (ナビゲータ) ダイアログボックスに表示されます。



8. [Display Options] (表示オプション) ペインで、使用するデータセットのチェックボックスをオンにします。
9. インポートする前にデータセットを変換する場合は、ダイアログボックスの下部に移動し、[Transform Data] (データを変換する) をクリックします。これにより、Power Query エディターが開き、使用するデータのセットをフィルタリングして絞り込むことができます。
10. ロード] を選択します。ロードが完了すると、次の画像のような可視化を作成できます。DirectQuery をインポートモードとして選択した場合、Power BI はユーザーがリクエストした可視化のクエリを Athena に対して発行します。



オンプレミスゲートウェイの設定

ダッシュボードとデータセットを Power BI サービスに公開すると、他のユーザーがウェブアプリ、モバイルアプリ、埋め込みアプリを使用してダッシュボードやデータセットを操作できるようになります。Microsoft Power BI サービスでデータを表示するには、Microsoft Power BI オンプレミスデータゲートウェイを AWS アカウント にインストールします。ゲートウェイは、Microsoft Power BI サービスと Athena の間の橋渡しのように機能します。

オンプレミスデータゲートウェイをダウンロードしてインストールし、テストを行う

1. 「Microsoft Power BI ゲートウェイのダウンロード」ページにアクセスし、個人モードまたは標準モードを選択します。個人モードは、Athena コネクタをローカルでテストする場合に便利です。標準モードは、マルチユーザーの本番稼働の設定に適しています。

2. オンプレミスゲートウェイ (個人モードまたは標準モード) をインストールするには、Microsoft のドキュメントの「オンプレミス データ ゲートウェイをインストールする」を参照してください。
3. ゲートウェイをテストするには、Microsoft のドキュメントの「オンプレミス データ ゲートウェイでカスタム データ コネクタを使用する」を参照してください。

オンプレミスデータゲートウェイの詳細については、次の Microsoft リソースを参照してください。

- オンプレミスデータゲートウェイとは?
- Power BI 用のデータゲートウェイのデプロイに関するガイダンス

Athena で使用するための Power BI ゲートウェイの設定例については、AWS Big Data ブログの記事「Creating dashboards quickly on Microsoft Power BI using Amazon Athena」を参照してください。

データベースとテーブルの作成

Amazon Athena では、データが Amazon Simple Storage Service に格納されている外部のテーブルを定義してクエリするために、データ定義言語 (DDL) ステートメントのサブセット、および ANSI SQL 関数と演算子がサポートされています。

Athena でデータベースとテーブルを作成するときは、スキーマとデータの場所を記述して、テーブル内のデータをリアルタイムのクエリ向けに準備します。

クエリパフォーマンスを向上させてコストを削減するため、データはパーティションして、Amazon S3 での保存のためにオープンソースの列形式 (Apache Parquet または ORC など) を使用することが推奨されます。

トピック

- Athena でのデータベースの作成
- Athena でのテーブルの作成
- テーブル、データベース、および列の名前
- 予約済みキーワード
- Amazon S3 のテーブルの場所
- 列指向ストレージ形式
- 列指向形式への変換
- Athena でのデータのパーティション化

- Amazon Athena でのパーティション射影

Athena でのデータベースの作成

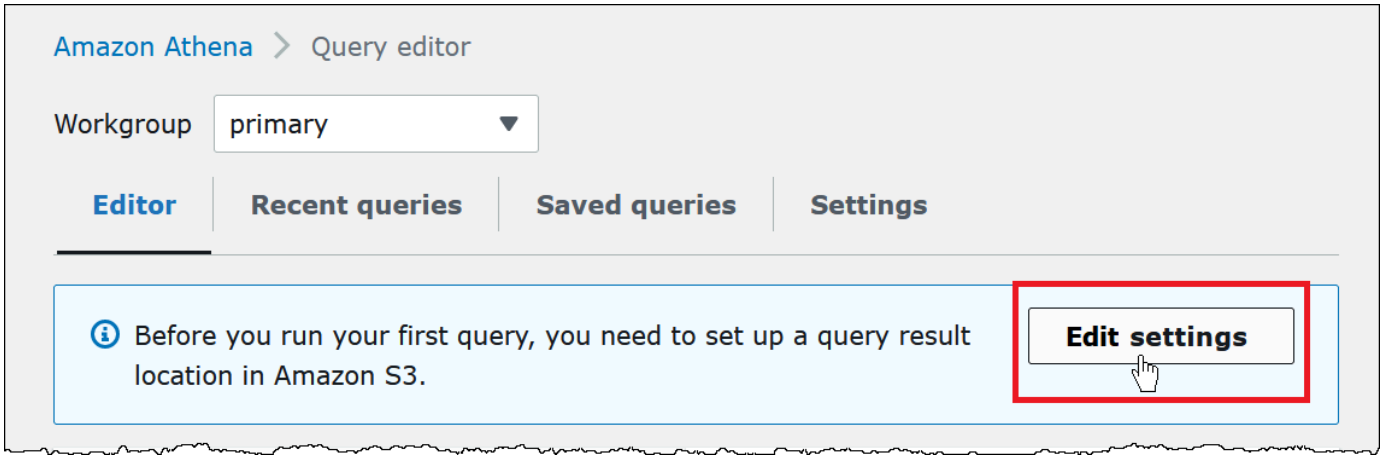
Athena のデータベースは、Athena で作成するテーブルの論理的なグループです。

前提条件

Amazon S3 でクエリ出力の場所をまだ設定していない場合は、以下の前提条件となるステップを実行して設定してください。

クエリ出力の場所を作成するには

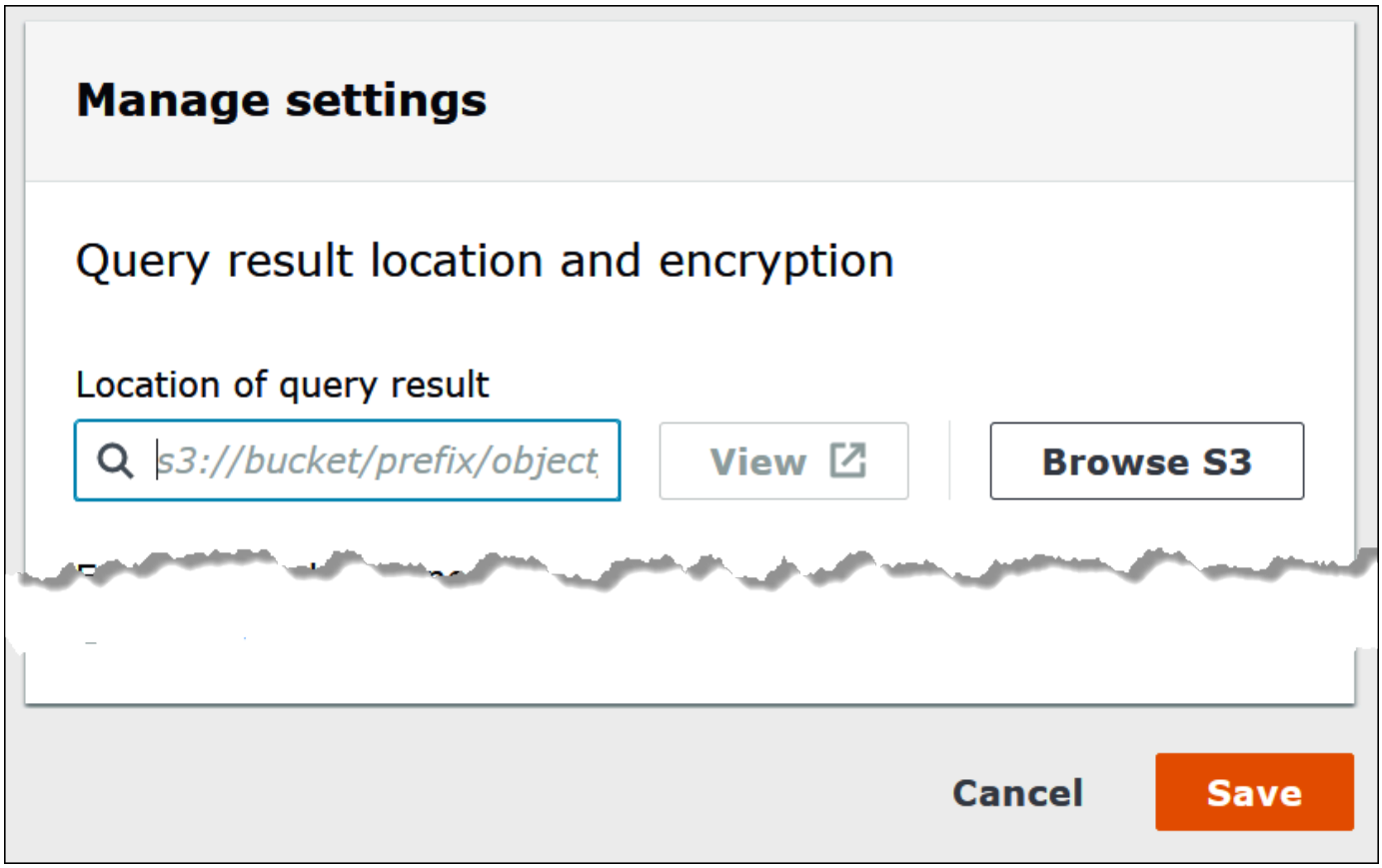
1. Athena で使用しているものと同じ AWS リージョンとアカウントを使用して、Athena からのクエリ結果を保存するバケットを Amazon S3 に作成するためのステップを実行します (Amazon S3 コンソールなどを使用して実行します)。このバケットをクエリ出力の場所として設定します。
2. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
3. この AWS リージョンで Athena コンソールを初めて使用する場合は、[クエリエディタを詳しく確認する] を選択してクエリエディタを開きます。初めてではない場合は、クエリエディタで Athena が開きます。
4. [Edit Settings] (設定を編集) をクリックし、Amazon S3 内のクエリ結果の保存場所をセットアップします。



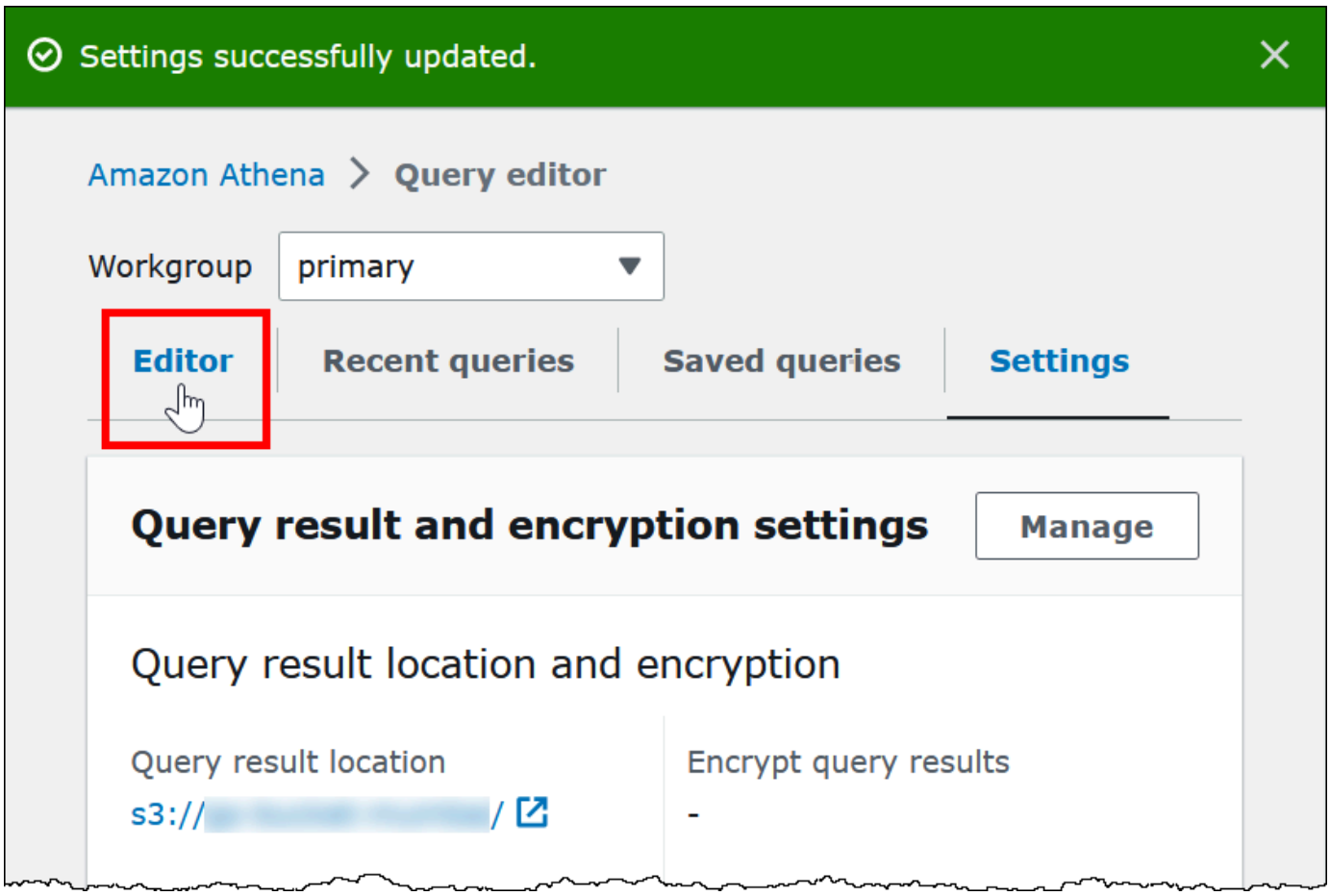
5. [Manage settings] (設定の管理) で、以下のいずれかを実行します。

   - [Location of query result] (クエリ結果の場所) ボックスで、クエリ結果のために Amazon S3 に作成したバケットへのパスを入力します。パスの先頭に `s3://` を付けます。

- [Browse S3] (S3 をブラウズ) をクリックし、現在のリージョンで作成した Amazon S3 バケットを選択した上で、[Choose] (選択) をクリックします。



6. [Save] (保存) を選択します。
7. [Editor] (エディタ) をクリックして、クエリエディタに切り替えます。



データベースを作成する

クエリ結果の場所を設定した後は、Athena コンソールのクエリエディタでデータベースを簡単に作成できます。

Athena クエリエディタを使用してデータベースを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. [エディタ] タブのクエリエディタで、Hive データ定義言語 (DDL) コマンド `CREATE DATABASE myDataBase` を入力します。*myDatabase* は、実際に使用する名前に置き換えます。データベース名の制限については、「テーブル、データベース、および列の名前」を参照してください。
3. [Run] (実行) をクリックするか、**Ctrl+ENTER** キーを押します。
4. 使用しているデータベースを現行のデータベースにするには、クエリエディタの左側にある [Database] (データベース) メニューから、対象のデータベースを選択します。

Athena データベースへのアクセス許可の制御については、「AWS Glue Data Catalog のデータベースとテーブルへのきめ細かなアクセス」を参照してください。

Athena でのテーブルの作成

Athena コンソールでの DDL ステートメントの実行は、JDBC もしくは ODBC ドライバー、または Athena の [Create table] (テーブルを作成) フォームを使用して行うことができます。

Athena で新しいテーブルスキーマを作成すると、Athena はこのスキーマをデータカタログに保存して、クエリの実行時に使用します。

Athena は schema-on-read として知られるアプローチを使用します。これは、クエリを実行するときにスキーマがデータに反映されることを意味し、データのロードや変換が不要になります。

Athena は、Amazon S3 内のデータを変更しません。

Athena は、Apache Hive を使用してテーブルを定義し、本質的にテーブルの論理名前空間であるデータベースを作成します。

Athena でデータベースとテーブルを作成するときは、read-time クエリのために、スキーマと、テーブルデータが格納されている Amazon S3 内の場所を記述することになります。このため、Athena のデータベースとテーブルには、従来のリレーショナルデータベースシステムのデータベースとテーブルとは少し異なる意味があります。データがデータベースとテーブルのスキーマ定義と共に保存されないからです。

クエリを実行する場合は、標準 SQL を使用してテーブルをクエリし、その時点でデータが読み取られます。データベースとテーブルの作成方法に関するガイダンスはApache Hive のドキュメントで参照できますが、以下は Athena に固有のガイダンスを提供します。

クエリ文字列の最大長は 256 KB です。

Hive では、シリアライザー/デシリアライザー (SerDe) ライブラリの使用を通じて複数のデータ形式をサポートします。正規表現を使用して複合型のスキーマを定義することもできます。サポートされている SerDe ライブラリのリストについては、「サポートされる SerDes とデータ形式」を参照してください。

考慮事項と制約事項

以下は、Athena のテーブルに関する重要な制限事項と考慮事項です。

Athena のテーブルと Amazon S3 のデータに関する要件

テーブルを作成するときは、LOCATION 句を使用して、基盤となるデータが格納される Amazon S3 バケットの場所を指定します。以下の点を考慮してください。

- Athena は、バージョン管理された Amazon S3 バケットの最新バージョンのデータしかクエリできず、以前のバージョンのデータはクエリできません。
- Amazon S3 内のデータを使用するには、適切なアクセス許可が必要です。詳細については、「Amazon S3 へのアクセス権」を参照してください。
- Athena は、LOCATION 句で指定されたものと同じバケット内にある、複数のストレージクラスを使って保存されたオブジェクトのクエリをサポートします。例えば、Amazon S3 内の異なるストレージクラス (スタンダード、標準、およびインテリジェントティアリング) で保存されたオブジェクトのデータをクエリできます。
- Athena では、リクエスタ支払いバケットがサポートされています。Athena でクエリする予定のソースデータが含まれるバケットに対してリクエスタ支払いを有効にする方法については、「ワークグループの作成」を参照してください。
- Athena は、S3 Glacier Flexible Retrieval または S3 Glacier Deep Archive ストレージクラス内にあるデータへのクエリをサポートしていません。S3 Glacier Flexible Retrieval、または S3 Glacier Deep Archive ストレージクラスにアーカイブされたオブジェクトは無視されます。代替方法として、Athena によってクエリ可能な Amazon S3 Glacier Instant Retrieval ストレージクラスを使用できます。詳細については、「Amazon S3 Glacier Instant Retrieval storage class」(Amazon S3 Glacier Instant Retrieval ストレージクラス) を参照してください。

  ストレージクラスの詳細については、「Amazon Simple Storage Service ユーザーガイド」の、「Storage Classes」(Amazon S3 ストレージクラスを使用する)、「Changing the Storage Class of an Object in Amazon S3」(Amazon S3 のオブジェクトのストレージクラスの変更)、「Transitioning to the GLACIER Storage Class (Object Archival)」(Glacier ストレージクラス (オブジェクトアーカイブ) へのオブジェクトの移行)、および「Requester Pays Buckets」(リクエスタ支払いバケット) を参照してください。
- 多数のオブジェクトがあり、データがパーティションされていない Amazon S3 バケットに対してクエリを発行する場合、このようなクエリは Amazon S3 の リクエストレート制限に影響を及ぼし、Amazon S3 例外を引き起こす可能性があります。エラーを防ぐには、データをパーティションします。また、Amazon S3 のリクエストレートをチューニングすることも検討してください。詳細については、「リクエスト率およびリクエストパフォーマンスに関する留意事項」を参照してください。

- AWS Glue CreateTable API 操作や AWS CloudFormation AWS::Glue::Table テンプレートを使用して、TableType プロパティを指定せずに Athena で使用するテーブルを作成し、SHOW CREATE TABLE または MSCK REPAIR TABLE などの DDL クエリを実行すると、「失敗: NullPointerException 名は null です」というエラーメッセージを受け取る場合があります。

  このエラーを解決するには、TableInput TableType 属性の値を AWS Glue CreateTable API コール、または AWS CloudFormation テンプレートの一部として指定します。TableType に使用できる値には、EXTERNAL_TABLE や VIRTUAL_VIEW が含まれます。

  この要件は、AWS Glue CreateTable API 操作や AWS::Glue::Table テンプレートを使用してテーブルを作成する場合にだけ適用されます。DDL ステートメントや AWS Glue クローラを使用して Athena のテーブルを作成すると、TableType プロパティが自動的に定義されます。

サポートされている関数

Athena クエリでサポートされる関数は、Trino と Presto の関数に対応しています。各関数の詳細については、「Trino ドキュメント」あるいは「Presto ドキュメント」の関数と演算子のセクションを参照してください。

トランザクションデータ変換はサポート対象外

Athena は、テーブルデータに対するトランザクションベースのオペレーション (Hive または Presto のものなど) をサポートしません。サポートされないキーワードの詳細なリストについては、「サポートされない DDL」を参照してください。

テーブルの状態を変更するオペレーションは ACID

テーブルを作成、更新、または削除する場合、これらのオペレーションは ACID 対応であることが保証されます。たとえば、複数のユーザーやクライアントが同時に既存のテーブルを作成または変更しようとした場合、成功するのは 1 つのユーザーまたはクライアントのみです。

テーブルは EXTERNAL

Iceberg テーブルの作成時を除いて、常に EXTERNAL キーワードが使用されます。Iceberg 以外のテーブルで、EXTERNAL キーワードを指定せずに CREATE TABLE を使用すると、Athena でエラーが発生します。Athena でテーブルをドロップすると、テーブルメタデータのみが削除され、データは Amazon S3 に残ります。

AWS Glue または Athena コンソールを使用したテーブルの作成

Athena でのテーブル作成には、AWS Glue やテーブルの追加フォームを使用するか、Athena クエリエディタで DDL ステートメントを実行します。

AWS Glue クローラを使用してテーブルを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. クエリエディタの [Tables and views] (テーブルとビュー) の横にある [Create] (作成) を選択し、その後 [AWS Glue crawler] を選択します。
3. AWS Glue コンソールの [Add crawler] (クローラの追加) ページで、次のステップに従いクローラを作成します。

   詳細については、「AWS Glue クローラの使用」を参照してください。

Athena のテーブル作成フォームを使用してテーブルを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. クエリエディタで、[Tables and views] (テーブルとビュー) の横にある [Create] (作成) をクリックし、次に [S3 bucket data] (S3 バケットデータ) をクリックします。
3. [Create Table From S3 bucket data] (S3 バケットデータからテーブルを作成) フォームで、テーブルを作成するための情報を入力し、[Create table] (テーブルを作成) を選択します。フォーム内の各フィールドの詳細については、「フォームを使用したテーブルの追加」を参照してください。

Hive DDL を使用してテーブルを作成する

1. [Database (データベース)] メニューから、テーブルを作成するデータベースを選択します。CREATE TABLE ステートメントでデータベースを指定しない場合、テーブルはクエリエディタで現在選択されているデータベースに作成されます。
2. クエリエディタで次のようなステートメントを入力した後に、[[Run] (実行) をクリックするか、**Ctrl+ENTER** を押し下げます。

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
    `Date` Date,
    Time STRING,
    Location STRING,
    Bytes INT,
    RequestIP STRING,
```

```
    Method STRING,
    Host STRING,
    Uri STRING,
    Status INT,
    Referrer STRING,
    OS String,
    Browser String,
    BrowserVersion String
) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
"input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s
+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^
\/]+)[\/](.*)$"
) LOCATION 's3://athena-examples-MyRegion/cloudfront/plaintext/';
```

テーブル情報の表示

Athena で作成が完了したテーブルは、その名前が左側の [Tables] (テーブル) リスト内に表示されます。テーブルの情報を表示し管理を行うには、Athena コンソール上で、テーブル名の横にある縦の 3 つのドットをクリックします。

- テーブルのプレビュー – Athena クエリエディタで `SELECT * FROM "database_name"."table_name" LIMIT 10` ステートメントを実行して、すべてのカラムの最初の 10 行を表示します。
- テーブル DDL の生成 – Athena クエリエディタで `SHOW CREATE TABLE table_name` ステートメントを実行して、テーブルを再作成するための DDL ステートメントを生成します。
- パーティションのロード – Athena クエリエディタで `MSCK REPAIR TABLE table_name` ステートメントを実行します。このオプションは、テーブルにパーティションがある場合にのみ使用できます。
- エディタへの挿入 – クエリエディタで現在編集を行っているか所に、テーブルの名前を挿入します。
- テーブルの削除 – テーブルを削除するかどうかを確認するダイアログボックスを表示します。同意すると、Athena クエリエディタで `DROP TABLE table_name` ステートメントが実行されます。
- テーブルのプロパティ – テーブル名、データベース名、作成時刻、テーブルに暗号化されたデータがあるかどうかが表示されます。

テーブル、データベース、および列の名前

Athena のデータベースオブジェクトに命名するには、以下のヒントを参考にしてください。

データベース、テーブル、および列の名前要件

- AWS Glue 内のデータベース名、テーブル名、および列名で使用できる文字は、UTF-8 文字列である必要があります。文字列は、1 バイト以上、255 バイト未満の長さにする必要があります。この制限を超えると、「Value at 'name' failed to satisfy constraint: Member must have length less than or equal to 255」などのエラーが生成されます。使用できる文字にはスペースが含まれ、以下にある単一行の文字列パターンで定義されます。

  ```
  [\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\t]*
  ```

- 現在、AWS Glue regex パターンでは、名前の先頭にスペースを追加することができます。このような先頭のスペースは検出が難しく、作成後にユーザビリティ問題を引き起こす可能性があるため、先頭にスペースがあるオブジェクト名は作成しないようにしてください。
- AWS Glue データベースを作成するために、AWS::Glue::Database の AWS CloudFormation テンプレートを、データベース名を指定せずに使用した場合は、AWS Glue が自動的にデータベース名を生成します。この形式は *resource_name-random_string* となり、Athena との互換性がなくなります。
- AWS Glue Catalog Manager では、列の名前を変更できますが、テーブル名やデータベース名は変更できません。この制限を回避するには、古いデータベースの定義を使用して、新しい名前のデータベースを作成する必要があります。次に、古いデータベースのテーブルの定義を使用して、新しいデータベースにテーブルを再度作成します。これには、AWS CLI または AWS Glue SDK を使用できます。この手順については、「AWS CLI を使用して AWS Glue データベースとそのテーブルを再度作成する」を参照してください。

Athena のテーブル名とテーブルの列名に小文字を使用する

Athena では大文字と小文字の DDL クエリと DML クエリを受け入れますが、クエリを実行するときの名前は小文字になります。このため、テーブル名や列名で大文字と小文字を混在させることは避け、そのような名前を区別するために、大文字と小文字の区別だけに依存しないでください。例えば、DDL ステートメントを使用して `Castle` という名前の列を作成する場合、作成される列は小文字で `castle` になります。また、DML クエリで列名を `Castle` または `CASTLE` に指定すると、Athena はクエリを実行する際の名前を小文字にし、クエリで選択した大文字と小文字を使用して列見出しを表示します。

データベース、テーブル、および列名は 255 文字以下である必要があります。

アンダースコアで始まる名前

テーブルを作成するときは、アンダースコアで始まるテーブル名、ビュー名、または列名はバックティックで囲みます。以下に例を示します。

```
CREATE EXTERNAL TABLE IF NOT EXISTS `_myunderscoretable`(
  `_id` string, `_index` string)
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

数字で始まるテーブル、ビュー、または列の名前

SELECT、CTAS、または VIEW クエリを実行するときは、数字で始まるテーブル名、ビュー名、または列名といった識別子を引用符で囲みます。以下に例を示します。

```
CREATE OR REPLACE VIEW "123view" AS
SELECT "123columnone", "123columntwo"
FROM "234table"
```

列名と複合型

複合型の場合、列名には、英数字、アンダースコア (_)、ピリオド (.) のみが使用可能です。カスタム DDL ステートメントを使用すると、テーブルとキーのマッピングの作成に使用する文字を制限できます。詳細については、AWS ビッグデータブログの「JSONSerDe によるマッピングを使って、入れ子の JSON から Amazon Athena のテーブルを作成する」を参照してください。

予約語

Athena の特定の予約語は、エスケープする必要があります。DDL ステートメントで予約キーワードをエスケープするには、バックティック (`) で囲みます。SQL の SELECT ステートメントや ビュー のクエリで予約キーワードをエスケープするには、二重引用符 (") で囲みます。

詳細については、「予約済みキーワード」を参照してください。

追加リソース

データベースおよびテーブル作成の構文の詳細については、次のページを参照してください。

- CREATE DATABASE

- CREATE TABLE

AWS Glue 内のデータベースとテーブルの詳細については、「AWS Glue デベロッパーガイド」の「データベース」と「テーブル」を参照してください。

予約済みキーワード

Athena で予約キーワードを含むクエリを実行する場合は、それらを特殊文字で囲んでエスケープする必要があります。このトピックのリストを使用して、どのキーワードが Athena で予約されているのかを確認してください。

DDL ステートメントで予約キーワードをエスケープするには、バックティック (`) で囲みます。SQL の SELECT ステートメントや ビュー のクエリで予約キーワードをエスケープするには、二重引用符 (") で囲みます。

- DDL ステートメントの予約キーワードのリスト
- SQL SELECT ステートメントの予約キーワードのリスト
- 予約ワードを含むクエリの例

DDL ステートメントの予約キーワードのリスト

Athena は、DDL ステートメントで以下にリストされている予約キーワードを使用します。Athena でこれらをエスケープせずに使用すると、エラーが発生します。予約キーワードをエスケープするには、バックティック (`) で囲みます。

DDL 予約キーワードをバックティック (`) で囲まずに DDL ステートメントで識別子名として使用することはできません。

```
ALL, ALTER, AND, ARRAY, AS, AUTHORIZATION, BETWEEN, BIGINT,
BINARY, BOOLEAN, BOTH, BY, CASE, CASHE, CAST, CHAR, COLUMN,
CONF, CONSTRAINT, COMMIT, CREATE, CROSS, CUBE, CURRENT,
CURRENT_DATE, CURRENT_TIMESTAMP, CURSOR, DATABASE, DATE,
DAYOFWEEK, DECIMAL, DELETE, DESCRIBE, DISTINCT, DOUBLE, DROP,
ELSE, END, EXCHANGE, EXISTS, EXTENDED, EXTERNAL, EXTRACT,
FALSE, FETCH, FLOAT, FLOOR, FOLLOWING, FOR, FOREIGN, FROM,
FULL, FUNCTION, GRANT, GROUP, GROUPING, HAVING, IF, IMPORT,
IN, INNER, INSERT, INT, INTEGER, INTERSECT, INTERVAL, INTO,
IS, JOIN, LATERAL, LEFT, LESS, LIKE, LOCAL, MACRO, MAP, MORE,
NONE, NOT, NULL, NUMERIC, OF, ON, ONLY, OR, ORDER, OUT,
```

```
OUTER, OVER, PARTIALSCAN, PARTITION, PERCENT, PRECEDING,
PRECISION, PRESERVE, PRIMARY, PROCEDURE, RANGE, READS,
REDUCE, REGEXP, REFERENCES, REVOKE, RIGHT, RLIKE, ROLLBACK,
ROLLUP, ROW, ROWS, SELECT, SET, SMALLINT, START,TABLE,
TABLESAMPLE, THEN, TIME, TIMESTAMP, TO, TRANSFORM, TRIGGER,
TRUE, TRUNCATE, UNBOUNDED,UNION, UNIQUEJOIN, UPDATE, USER,
USING, UTC_TIMESTAMP, VALUES, VARCHAR, VIEWS, WHEN, WHERE,
WINDOW, WITH
```

SQL SELECT ステートメントの予約キーワードのリスト

Athena は、SQL SELECT ステートメント、およびビューでのクエリで以下にリストされている予約キーワードを使用します。

これらのキーワードを識別子として使用する場合、クエリステートメントでは二重引用符 (") で囲む必要があります。

```
ALTER, AND, AS, BETWEEN, BY, CASE, CAST, CONSTRAINT, CREATE,
CROSS, CUBE, CURRENT_CATALOG, CURRENT_DATE, CURRENT_PATH,
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER,
DEALLOCATE, DELETE, DESCRIBE, DISTINCT, DROP, ELSE, END, ESCAPE,
EXCEPT, EXECUTE, EXISTS, EXTRACT, FALSE, FIRST, FOR, FROM,
FULL, GROUP, GROUPING, HAVING, IN, INNER, INSERT, INTERSECT,
INTO, IS, JOIN, JSON_ARRAY, JSON_EXISTS, JSON_OBJECT,
JSON_QUERY, JSON_TABLE, JSON_VALUE, LAST, LEFT, LIKE,
LISTAGG, LOCALTIME, LOCALTIMESTAMP, NATURAL, NORMALIZE,
NOT, NULL, OF, ON, OR, ORDER, OUTER, PREPARE, RECURSIVE, RIGHT,
ROLLUP, SELECT, SKIP, TABLE, THEN, TRIM, TRUE, UESCAPE, UNION,
UNNEST, USING, VALUES, WHEN, WHERE, WITH
```

予約ワードを含むクエリの例

次の例のクエリでは、バックティック (`) を使用して DDL に関連する予約キーワード partition および date をエスケープします。これらはテーブル名および列の名前の 1 つに使用されています。

```
CREATE EXTERNAL TABLE `partition` (
`date` INT,
col2 STRING
)
PARTITIONED BY (year STRING)
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/test_examples/';
```

次のクエリ例では、ALTER TABLE ADD PARTITION と ALTER TABLE DROP PARTITION ステートメントに、DDL に関連する予約キーワードを含む列名が含まれています。DDL 予約キーワードは、バックティック (') で囲まれます。

```
ALTER TABLE test_table
ADD PARTITION (`date` = '2018-05-14')
```

```
ALTER TABLE test_table
DROP PARTITION (`partition` = 'test_partition_value')
```

次のクエリ例では、予約キーワード (end) が SELECT ステートメントの識別子として含まれています。キーワードは二重引用符でエスケープされます。

```
SELECT *
FROM TestTable
WHERE "end" != nil;
```

次のクエリ例では、SELECT ステートメントに予約キーワード (first) が含まれています。キーワードは二重引用符でエスケープされます。

```
SELECT "itemId"."first"
FROM testTable
LIMIT 10;
```

Amazon S3 のテーブルの場所

Athena で CREATE TABLE クエリを実行するとき、Athena はテーブルを Athena がメタデータを保存する AWS Glue データカタログに登録します。

以下の例にあるように、Amazon S3 内のデータへのパスを指定するには、LOCATION プロパティを使用します。

```
CREATE EXTERNAL TABLE `test_table`(
...
)
ROW FORMAT ...
STORED AS INPUTFORMAT ...
OUTPUTFORMAT ...
LOCATION s3://DOC-EXAMPLE-BUCKET/folder/
```

- バケットの命名についての詳細は、「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。
- Amazon S3 でのフォルダの使用についての詳細は、「Amazon Simple Storage Service ユーザーガイド」の「Using Folders」(フォルダの使用) を参照してください。

Amazon S3 の LOCATION は、テーブルを表すすべてのファイルを指定します。

> **Important**
>
> Athena が、指定した Amazon S3 フォルダに保存されているすべてのデータを読み込みます。Athena に読み込ませたくないデータがある場合は、そのデータを Athena に読み込ませたいデータと同じ Amazon S3 フォルダに保存しないでください。パーティションを利用している場合に、Athena がパーティション内のデータをスキャンすることを確実にするには、WHERE フィルターにパーティションが含まれている必要があります。詳細については、「テーブルの場所とパーティション」を参照してください。

CREATE TABLE ステートメントで LOCATION を指定する場合は、次のガイドラインを使用します。

- 末尾にスラッシュを使用します。
- Amazon S3 フォルダまたは Amazon S3 アクセスポイントエイリアスへのパスを使用できます。Amazon S3 アクセスポイントエイリアスの詳細については、「Amazon S3 ユーザーガイド」の「Using a bucket-style alias for your access point」を参照してください。

を使用します。

```
s3://DOC-EXAMPLE-BUCKET/folder/
```

```
s3://DOC-EXAMPLE-BUCKET-metadata-s3alias/folder/
```

以下の項目をデータの LOCATION の指定に使用しないでください。

- ファイルの場所を指定するために、ファイル名、アンダースコア、ワイルドカード、または glob パターンを使用しないでください。
- s3.amazon.com などの完全な HTTP 表記を Amazon S3 バケットのパスに追加しないでください。

- パスに // のような空のフォルダを使用しないでください (例: S3://DOC-EXAMPLE-BUCKET/*folder*//*folder*/)。これは有効な Amazon S3 パスですが、Athena はそれを許可せず、余分な / を削除して、s3://DOC-EXAMPLE-BUCKET/*folder*/*folder*/ に変更します。

  使用しない:

```
s3://DOC-EXAMPLE-BUCKET
s3://DOC-EXAMPLE-BUCKET/*
s3://DOC-EXAMPLE-BUCKET/mySpecialFile.dat
s3://DOC-EXAMPLE-BUCKET/prefix/filename.csv
s3://DOC-EXAMPLE-BUCKET.s3.amazon.com
S3://DOC-EXAMPLE-BUCKET/prefix//prefix/
arn:aws:s3:::DOC-EXAMPLE-BUCKET/prefix
s3://arn:aws:s3:<region>:<account_id>:accesspoint/<accesspointname>
https://<accesspointname>-<number>.s3-accesspoint.<region>.amazonaws.com
```

テーブルの場所とパーティション

ソースデータは、列のセットに基づいたパーティションと呼ばれる Amazon S3 フォルダに分類できます。たとえば、これらの列が、特定のレコードが作成された年、月、日を表す場合があります。

テーブルを作成するときに、パーティション分割されるように選択できます。Athena がパーティション分割されていないテーブルに対して SQL クエリを実行するときは、テーブル定義からの LOCATION プロパティをリスト化するための基本的なパスとして使用してから、利用可能なすべてのファイルをスキャンします。ただし、パーティション分割されたテーブルをクエリする前に、パーティション情報で AWS Glue データカタログを更新する必要があります。この情報は、特定のパーティション内のファイルのスキーマと、そのパーティションに関する Amazon S3 内のファイルの LOCATION を表します。

- AWS Glue クローラがパーティションを追加する方法については、「AWS Glue デベロッパーガイド」で「クローラがパーティションを作成するタイミングを判断する方法」を参照してください。
- 既存のパーティションでデータのテーブルを作成するようにクローラを設定する方法については、「クローラでの複数のデータソースの使用」を参照してください。
- テーブルのパーティションは、Athena で直接作成することもできます。詳細については、「Athena でのデータのパーティション化」を参照してください。

Athena がパーティション分割されたテーブルをクエリするときは、クエリの WHERE 句にパーティション分割された列が使用されているかどうかを確認します。パーティション分割された列が使用さ

れている場合、Athena は AWS Glue データカタログに対して、指定されたパーティション列に一致するパーティション仕様を返すようにリクエストします。パーティション仕様には、データを読み取るときに使用する Amazon S3 プレフィックスを Athena に伝える LOCATION プロパティが含まれています。この場合、このプレフィックスに保存されたデータのみがスキャンされます。WHERE 句でパーティション分割された列を使用しない場合、Athena はテーブルのパーティションに属するすべてのファイルをスキャンします。

クエリパフォーマンスを向上させてクエリコストを削減するために Athena でパーティショニングを使用する例については、「Amazon Athena のパフォーマンスチューニング Tips トップ 10」を参照してください。

列指向ストレージ形式

Apache Parquet や ORC は、データを高速に取得できるように最適化された、AWS 分析アプリケーションで使用されている、列指向ストレージ形式です。

列指向ストレージ形式には以下の特性があるため、Athena での使用に適しています。

- 列のデータ型に合わせて選択された圧縮アルゴリズムによる列ごとの圧縮で、Amazon S3 のストレージ領域を節約し、ディスク容量とクエリの処理中における I/O を削減します。
- Parquet および ORC での述語プッシュダウンにより、Athena クエリが必要なブロックのみを取得できるようになり、クエリパフォーマンスが向上します。Athena クエリがデータから特定の列値を取得すると、データブロック述語からの統計 (最大値や最小値など) を使用して、そのブロックを読み取るかスキップするかを判断します。
- Parquet および ORC でのデータの分割により、Athena がデータの読み取りを複数のリーダーに分割して、クエリ処理時における並列化を向上させることが可能になります。

既存の raw データを他のストレージ形式から Parquet または ORC に変換するには、Athena で CREATE TABLE AS SELECT (CTAS) クエリを実行してデータストレージ形式を Parquet もしくは ORC として指定する、または AWS Glue クローラを使用することができます。

Parquet または ORC のどちらかを選択

ORC (最適化された列指向) または Parquet のどちらを選択するかは、特定の使用要件によって異なります。

Apache Parquet は効率的なデータ圧縮とエンコーディングスキームを提供しており、複雑なクエリの実行や大量データの処理に最適です。Parquet は Apache Arrow での使用に最適化されているため、Arrow 関連ツールを使用する場合に便利です。

ORC は Hive データを効率的に保存する方法を提供します。ORC ファイルは Parquet ファイルよりも小さいことが多く、ORC インデックスを使用するとクエリを高速化できます。さらに、ORC は構造体、マップ、リストなどの複雑な型をサポートしています。

Parquet または ORC のいずれかを選択するには、以下の点を考慮してください。

クエリのパフォーマンス — Parquet はより幅広い種類のクエリをサポートしているため、複雑なクエリを実行する場合は、Parquet の方が適している場合があります。

複雑なデータ型 — 複雑なデータ型を使用している場合は、ORC の方が幅広い複合データ型をサポートしているので、より適している場合があります。

ファイルサイズ — ディスク容量が懸念される場合、通常、ORC を使用するとファイルが小さくなり、ストレージコストを低減できます。

圧縮 — Parquet と ORC はどちらも優れた圧縮機能を備えていますが、最適な形式は特定のユースケースによって異なります。

進化 — Parquet と ORC はどちらもスキーマの進化をサポートしています。つまり、時間の経過とともに列を追加、削除、または変更できます。

Parquet と ORC はどちらもビッグデータアプリケーションに適していますが、選択する前にシナリオの要件を検討してください。データとクエリに対してベンチマークを実行して、どの形式がユースケースに適したパフォーマンスを発揮するかを確認することをお勧めします。

列指向形式への変換

データを Apache Parquet または ORC などのオープンソースの列指向形式に変換すると、Amazon Athena のクエリパフォーマンスが向上します。

JSON や CSV などのソースデータを列指向形式に簡単に変換するためのオプションには、CREATE TABLE AS クエリや、AWS Glue でのジョブの実行などがあります。

- `CREATE TABLE AS` (CTAS) クエリを使用すると、データを Parquet または ORC に 1 ステップで変換できます。例については、「CTAS クエリの例」ページの「例: クエリ結果を異なる形式で書き込む」を参照してください。
- CSV データを Parquet に変換する AWS Glue ジョブの実行については、AWS Big Data ブログの記事「AWS Glue と Amazon S3 を使用してデータレイクの基礎を構築する」の「データを CSV 形式から Parquet 形式に変換する」を参照してください。AWS Glue では CSV データの ORC への変換、あるいは JSON データの Parquet または ORC への変換にも同じ手法を使用できます。

Athena でのデータのパーティション化

データをパーティションすることで、各クエリによってスキャンされるデータの量を制限できるようになるため、パフォーマンスが向上し、コストが削減されます。任意のキーでデータをパーティションに分割することができます。一般的な方法では、時間に基づいてデータをパーティションします。これにより、通常、複数レベルのパーティション構成となります。たとえば、1 時間ごとに配信されるデータを年、月、日、時間でパーティションできます。別の例として、データが配信されるソースが多数に分かれているものの、それらのロードは 1 日 1 回だけ行われる場合には、データソースと日付によるパーティションを行います。

Athena では Apache Hive スタイルのパーティションを使用できます。このパーティションのデータパスには、等号で連結されたキーと値のペア (例えば `country=us/...` または `year=2021/month=01/day=26/...`) が含まれています。つまり、それぞれのパスにより、パーティションのキーと値、両方の名前が表されます。新しい Hive パーティションをパーティションされたテーブルにロードするには、(Hive スタイルのパーティションのみで機能する) MSCK REPAIR TABLE コマンドを使用します。

Athena では、Hive 以外のスタイルのパーティション化スキームを使用することも可能です。たとえば、CloudTrail ログと Firehose 配信ストリームは、`data/2021/01/26/us/6fc7845e.json` など、日付部分に個別のパスコンポーネントを使用します。これらの Hive スタイルではないパーティションの場合、ALTER TABLE ADD PARTITION を使用して手動でパーティションを追加します。

考慮事項と制約事項

パーティションを使用する場合は、次の点に注意してください。

- パーティションされたテーブルをクエリし、`WHERE` 句でパーティションを指定する場合、Athena はそのパーティションからのデータしかスキャンしません。詳細については、「テーブルの場所とパーティション」を参照してください。
- 多数のオブジェクトがあり、データがパーティションされていない Amazon S3 バケットに対してクエリを発行する場合、このようなクエリは Amazon S3 の `GET` リクエストレート制限に影響を及ぼし、Amazon S3 例外を引き起こす可能性があります。エラーを防ぐには、データをパーティションします。また、Amazon S3 のリクエストレートをチューニングすることも検討してください。詳細については、「設計パターンのベストプラクティス: Simple Storage Service (Amazon S3) のパフォーマンスの最適化」を参照してください。
- Athena で使用されるパーティションの場所は、`s3` プロトコル (`s3://DOC-EXAMPLE-BUCKET/folder/` など) を使用する必要があります。Athena では、他のプロトコル (`s3a://`

DOC-EXAMPLE-BUCKET/*folder*/ など) を使用する場所は、そこにあるテーブルに対して MSCK REPAIR TABLE クエリを実行する場合にクエリが失敗する原因になります。

- Simple Storage Service (Amazon S3) パスがキャメルケースではなく小文字になっていることを確認します (例えば、userId ではなく userid)。S3 パスがキャメルケースの場合、MSCK REPAIR TABLE は AWS Glue Data Catalog にパーティションを追加しません。詳細については、「MSCK REPAIR TABLE」を参照してください。
- MSCK REPAIR TABLE がフォルダとそのサブフォルダの両方をスキャンして一致するパーティションスキームを検索するため、別個のテーブルのデータは別個のフォルダ階層に保存するようにしてください。例えば、テーブル 1 のデータが s3://DOC-EXAMPLE-BUCKET1 にあり、テーブル 2 のデータが s3://DOC-EXAMPLE-BUCKET1/table-2-data にあるとします。両方のテーブルが文字列でパーティション分割されている場合、MSCK REPAIR TABLE はテーブル 2 のパーティションをテーブル 1 に追加します。これを回避するには、この代わりに s3://DOC-EXAMPLE-BUCKET2 や s3://DOC-EXAMPLE-BUCKET1 といった別個のフォルダ構造を使用します。この動作は、Amazon EMR および Apache Hive と同じであることに注意してください。
- Athena で AWS Glue Data Catalog を使用している場合は、「AWS Glue エンドポイントとクォータ」を参照し、アカウントごとおよびテーブルごとのパーティションに関するサービスクォータをご確認ください。
  - Athena では、1,000 万のパーティションを持つ AWS Glue テーブルへのクエリがサポートされていますが、1 回のスキャンで読み取れるのは、100 万のパーティションまでです。このようなシナリオでは、パーティションのインデックス作成が役立ちます。詳細については、AWS Big Data ブログの記事「AWS Glue Data Catalog パーティションインデックスを使用して Amazon Athena のクエリパフォーマンスを向上させる」を参照してください。
- AWS Glue Data Catalog を使用している場合にパーティションのクォータの引き上げをリクエストするには、AWS Glue の Service Quotas コンソール を参照してください。

パーティションされたデータを使用するテーブルの作成とロード

パーティションを使用するテーブルを作成するには、CREATE TABLE ステートメントの中で PARTITIONED BY 句を使用します。次の例のように、PARTITIONED BY はデータのパーティションに使用するキーを定義します。LOCATION 句では、パーティションされたデータのルートロケーションを指定します。

```
CREATE EXTERNAL TABLE users (
first string,
last string,
username string
```

```
)
PARTITIONED BY (id string)
STORED AS parquet
LOCATION 's3://DOC-EXAMPLE-BUCKET'
```

テーブルを作成した後、クエリ用にパーティションにデータをロードします。Hive スタイルのパーティションでは、MSCK REPAIR TABLE を実行します。Hive スタイルではないパーティションの場合、ALTER TABLE ADD PARTITION を使用して手動でパーティションを追加します。

クエリ用の Hive スタイルおよび非 Hive スタイルデータの準備

以下のセクションでは、Hive スタイルおよび非 Hive スタイルのデータを、Athena でのクエリ向けに準備する方法について説明します。

シナリオ 1: Amazon S3 に Hive 形式で保存されているデータの場合

このシナリオでは、パーティションは Amazon S3 内で別々のフォルダに保存されています。例として、`aws s3 ls` コマンドが出力した、サンプル広告のインプレッションに関するリスティングの一部を以下に示します。指定されたプレフィックスの下に、S3 オブジェクトがリストされています。

```
aws s3 ls s3://elasticmapreduce/samples/hive-ads/tables/impressions/

    PRE dt=2009-04-12-13-00/
    PRE dt=2009-04-12-13-05/
    PRE dt=2009-04-12-13-10/
    PRE dt=2009-04-12-13-15/
    PRE dt=2009-04-12-13-20/
    PRE dt=2009-04-12-14-00/
    PRE dt=2009-04-12-14-05/
    PRE dt=2009-04-12-14-10/
    PRE dt=2009-04-12-14-15/
    PRE dt=2009-04-12-14-20/
    PRE dt=2009-04-12-15-00/
    PRE dt=2009-04-12-15-05/
```

この例では、ログを保存する列名 (dt) が日付、時、分の増分と等しくなるように設定されています。親フォルダの場所、スキーマ、およびパーティション分割された列の名前を指定して DDL を提供すると、Athena はこれらのサブフォルダのデータをクエリできます。

テーブルを作成する

このデータからテーブルを作成するには、以下の Athena DDL ステートメントにあるように、「dt」を使用してパーティションを作成します。

```
CREATE EXTERNAL TABLE impressions (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string,
    number string,
    processId string,
    browserCookie string,
    requestEndTime string,
    timers struct<modelLookup:string, requestTime:string>,
    threadId string,
    hostname string,
    sessionId string)
PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION 's3://elasticmapreduce/samples/hive-ads/tables/impressions/' ;
```

このテーブルは、Hive のネイティブ JSON シリアライザー/デシリアライザーを使用して Amazon S3 に保存された JSON データを読み込みます。サポートされる形式の詳細については、「サポートされる SerDes とデータ形式」を参照してください。

MSCK REPAIR TABLE を実行する

`CREATE TABLE` クエリの実行後、以下の例のように Athena クエリエディタで `MSCK REPAIR TABLE` コマンドを実行し、パーティションをロードします。

```
MSCK REPAIR TABLE impressions
```

このコマンドを実行すると、データにクエリするための準備が整います。

データのクエリ

パーティション列を使用して、インプレッションテーブルのデータをクエリします。例を示します。

```
SELECT dt,impressionid FROM impressions WHERE dt<'2009-04-12-14-00' and
 dt>='2009-04-12-13-00' ORDER BY dt DESC LIMIT 100
```

このクエリにより、次のような結果が表示されます。

```
2009-04-12-13-20    ap3HcVKAWfXtgIPu6WpuUfAfL0DQEc
2009-04-12-13-20    17uchtodoS9kdeQP1x0XThKl5IuRsV
2009-04-12-13-20    JOUf1SCtRwviGw8sVcghqE5h0nkgtp
2009-04-12-13-20    NQ2XP0J0dvVbCXJ0pb4XvqJ5A4QxxH
2009-04-12-13-20    fFAItiBMsgqro9kRdIwbeX60SROaxr
2009-04-12-13-20    V4og4R9W6G3QjHHwF7gI1cSqig5D1G
2009-04-12-13-20    hPEPtBwk45msmwWTxPVVo1kVu4v11b
2009-04-12-13-20    v0SkfxegheD90gp31UCr6FplnKpx6i
2009-04-12-13-20    1iD9odVgOIi4QWkwHMcOhmwTkWDKfj
2009-04-12-13-20    b31tJiIA25CK8eDHQrHnbcknfSndUk
```

シナリオ 2: データが Hive 形式でパーティション化されない

次の例では aws s3 ls コマンドにより、Amazon S3 に保存されている ELB ログを表示します。このデータレイアウトでは key=value ペアを使用しておらず、つまり、Hive 形式ではないという点に注意してください。(aws s3 ls コマンドの --recursive オプションは、指定したディレクトリまたはプレフィックスに含まれる、すべてのファイルまたはオブジェクトを一覧表示するように指定します。)

```
aws s3 ls s3://athena-examples-myregion/elb/plaintext/ --recursive

2016-11-23 17:54:46   11789573 elb/plaintext/2015/01/01/part-r-00000-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    8776899 elb/plaintext/2015/01/01/part-r-00001-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    9309800 elb/plaintext/2015/01/01/part-r-00002-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9412570 elb/plaintext/2015/01/01/part-r-00003-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47   10725938 elb/plaintext/2015/01/01/part-r-00004-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    9439710 elb/plaintext/2015/01/01/part-r-00005-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47          0 elb/plaintext/2015/01/01_$folder$
2016-11-23 17:54:47    9012723 elb/plaintext/2015/01/02/part-r-00006-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:47    7571816 elb/plaintext/2015/01/02/part-r-00007-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9673393 elb/plaintext/2015/01/02/part-r-00008-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11979218 elb/plaintext/2015/01/02/part-r-00009-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    9546833 elb/plaintext/2015/01/02/part-r-00010-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   10960865 elb/plaintext/2015/01/02/part-r-00011-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48          0 elb/plaintext/2015/01/02_$folder$
2016-11-23 17:54:48   11360522 elb/plaintext/2015/01/03/part-r-00012-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11211291 elb/plaintext/2015/01/03/part-r-00013-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    8633768 elb/plaintext/2015/01/03/part-r-00014-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11891626 elb/plaintext/2015/01/03/part-r-00015-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49    9173813 elb/plaintext/2015/01/03/part-r-00016-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11899582 elb/plaintext/2015/01/03/part-r-00017-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49          0 elb/plaintext/2015/01/03_$folder$
2016-11-23 17:54:50    8612843 elb/plaintext/2015/01/04/part-r-00018-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50   10731284 elb/plaintext/2015/01/04/part-r-00019-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    9984735 elb/plaintext/2015/01/04/part-r-00020-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    9290089 elb/plaintext/2015/01/04/part-r-00021-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    7896339 elb/plaintext/2015/01/04/part-r-00022-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8321364 elb/plaintext/2015/01/04/part-r-00023-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/04_$folder$
2016-11-23 17:54:51    7641062 elb/plaintext/2015/01/05/part-r-00024-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51   10253377 elb/plaintext/2015/01/05/part-r-00025-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8502765 elb/plaintext/2015/01/05/part-r-00026-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:51   11518464 elb/plaintext/2015/01/05/part-r-00027-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7945189 elb/plaintext/2015/01/05/part-r-00028-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7864475 elb/plaintext/2015/01/05/part-r-00029-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/05_$folder$
2016-11-23 17:54:51   11342140 elb/plaintext/2015/01/06/part-r-00030-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8063755 elb/plaintext/2015/01/06/part-r-00031-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9387508 elb/plaintext/2015/01/06/part-r-00032-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9732343 elb/plaintext/2015/01/06/part-r-00033-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11510326 elb/plaintext/2015/01/06/part-r-00034-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9148117 elb/plaintext/2015/01/06/part-r-00035-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52          0 elb/plaintext/2015/01/06_$folder$
2016-11-23 17:54:52    8402024 elb/plaintext/2015/01/07/part-r-00036-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    8282860 elb/plaintext/2015/01/07/part-r-00037-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11575283 elb/plaintext/2015/01/07/part-r-00038-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53    8149059 elb/plaintext/2015/01/07/part-r-00039-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53   10037269 elb/plaintext/2015/01/07/part-r-00040-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53   10019678 elb/plaintext/2015/01/07/part-r-00041-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53          0 elb/plaintext/2015/01/07_$folder$
2016-11-23 17:54:53          0 elb/plaintext/2015/01_$folder$
2016-11-23 17:54:53          0 elb/plaintext/2015_$folder$
```

ALTER TABLE ADD PARTITION を実行する

ここでのデータは Hive 形式ではないため、パーティションの作成後に、テーブルにパーティションを追加するために `MSCK REPAIR TABLE` コマンドを使用することはできません。代わりに ALTER TABLE ADD PARTITION コマンドを使用することで、各パーティションを手動で追加できます。例えば、s3://athena-examples-*myregion*/elb/plaintext/2015/01/01/ のデータをロードするには、次の

クエリを実行します。Amazon S3 フォルダごとに個別のパーティション列は必要ないこと、およびパーティションキーの値が Amazon S3 キーと同じではない場合があることに注意してください。

```
ALTER TABLE elb_logs_raw_native_part ADD PARTITION (dt='2015-01-01') location 's3://
athena-examples-us-west-1/elb/plaintext/2015/01/01/'
```

パーティションが既に存在する場合は、Partition already exists というエラーが表示されます。このエラーを回避するには、`IF NOT EXISTS` 句を使用することができます。詳細については、「ALTER TABLE ADD PARTITION」を参照してください。パーティションを削除する場合は、ALTER TABLE DROP PARTITION を使用します。

パーティション射影

パーティション射影を使用すると、パーティション管理の必要性を回避できます。パーティション射影は、高度にパーティション化されていて、その構造が既知のテーブルで使用できるオプションです。パーティション射影では、パーティションの値と場所はメタデータリポジトリから読み取られるのではなく、テーブルのプロパティ設定を基に算出されます。メモリ内での計算処理は、リモートルックアップよりも高速であるため、パーティション射影を使用することでクエリランタイムが大幅に短縮されます。

詳細については、「Amazon Athena でのパーティション射影」を参照してください。

追加リソース

- Firehose データのパーティショニングオプションについては、「Amazon Data Firehose 例」を参照してください。
- JDBC ドライバーを使用して、パーティションの追加を自動化できます。
- CTAS と INSERT INTO を使用して、データセットをパーティションできます。詳細については、「ETL およびデータ分析での CTAS および INSERT INTO の使用」を参照してください。

Amazon Athena でのパーティション射影

Athena では、高度にパーティションされたテーブルのクエリ処理を高速化し、パーティション管理を自動化するためにパーティション射影を使用できます。

パーティション射影では、Athena は AWS Glue のテーブルに直接設定したテーブルプロパティを使用してパーティション値と場所を計算します。テーブルプロパティにより、Athena は AWS Glue

Data Catalogで時間をかけてメタデータを検索しなくても、必要なパーティション情報を「射影」または決定できます。多くの場合、インメモリオペレーションはリモートオペレーションよりも高速であるため、パーティション射影は高度にパーティションされたテーブルに対するクエリの実行時間を短縮できます。クエリおよび基盤となるデータの特定の特性によっては、パーティション射影によって、パーティションメタデータの取得時に制限されているクエリのクエリランタイムが大幅に短縮されます。

大きくパーティション化されたテーブルのプルーニングと射影

パーティションプルーニングは、メタデータを収集し、クエリに適用されるパーティションのみに「プルーニング」します。多くの場合、これによってクエリが高速化されます。Athena は、パーティション射影用に設定されたテーブルなどのパーティション列があるすべてのテーブルに対してパーティションプルーニングを使用します。

通常、クエリを処理する場合、Athena はパーティションプルーニングを実行する前に AWS Glue Data Catalog に対して `GetPartitions` 呼び出しを行います。テーブルに多数のパーティションがある場合、`GetPartitions` を使用すると、パフォーマンスに悪影響が及ぶ可能性があります。これを回避するには、パーティション射影を使用します。パーティション射影を設定すると、Athena が独自にパーティションを構築するために必要なすべての情報を得ることができるため、`GetPartitions` を呼び出す必要がなくなります。

パーティション射影の使用

パーティション射影を使用するには、AWS Glue Data Catalog または 外部 Hive メタストアのテーブルプロパティで、各パーティション列のパーティション値と射影型の範囲を指定します。テーブル上のこれらのカスタムプロパティは、Athena がテーブルでクエリを実行するときに、どのパーティションパターンを期待すべきかを把握できるようにします。クエリの実行中、Athena は、この情報を使用して、パーティション値を射影します。AWS Glue Data Catalog または外部 Hive メタストアから取得するのではありません。これにより、クエリの実行時間が短縮されるだけでなく、Athena、AWS Glue、または外部の Hive メタストアでパーティションを手動で作成する必要がなくなるため、パーティション管理も自動化されます。

> **Important**
>
> テーブルでパーティション射影を有効にすると、AWS Glue Data Catalog または Hive メタストア内のテーブルに登録されているパーティションメタデータが Athena によりすべて無視されます。

ユースケース

パーティション射影が役立つシナリオには、次のようなものがあります。

- 高度にパーティション化されたテーブルに対するクエリが、思ったほどすぐに完了しない。
- データに新しい日付または時刻パーティションが作成されたとき、定期的にパーティションをテーブルに追加する。パーティション射影で、新しいデータが到着したときに使用できる相対日付範囲を設定している。
- Amazon S3 に高度にパーティションされたデータがある。データは、AWS Glue Data Catalog または Hive メタストア内のモデルに対して実用的ではなく、クエリがそのごく一部のみを読み込む。

射影可能なパーティション構造

パーティション射影は、パーティションが次のような予測可能なパターン (ただし、これに限りません) に従う場合に最も簡単に設定できます。

- 整数 – `[1, 2, 3, 4, ..., 1000]` または `[0500, 0550, 0600, ..., 2500]` などの整数の連続的なシーケンス。
- 日付 – `[20200101, 20200102, ..., 20201231]` または `[1-1-2020 00:00:00, 1-1-2020 01:00:00, ..., 12-31-2020 23:00:00]` などの日付や日時の連続的なシーケンス。
- 列挙値 – 空港コードや AWS リージョン など、列挙値の有限集合。
- AWS のサービス ログ – AWS のサービス ログは通常、AWS Glue で指定できるので Athena がパーティション射影に使用できるパーティションスキームを持つ既知の構造になっています。

パーティションパステンプレートのカスタマイズ

デフォルトで、Athena はフォーム `s3://DOC-EXAMPLE-BUCKET/<table-root>/partition-col-1=<partition-col-1-val>/partition-col-2=<partition-col-2-val>/` を使用してパーティションの場所を構築しますが、データの編成が異なる場合、Athena はこのパステンプレートをカスタマイズするためのメカニズムを提供します。この手順については、「カスタム S3 ストレージの場所の指定」を参照してください。

考慮事項と制約事項

以下の考慮事項に注意してください。

- パーティション射影により、AWS Glue または外部 Hive メタストアでパーティションを手動で指定する必要がなくなります。
- テーブルでパーティション射影を有効にすると、そのテーブルについては、Athena は AWS Glue Data Catalog 内または外部 Hive メタストア内のパーティションメタデータを無視します。
- 射影されたパーティションが Amazon S3 に存在しない場合でも、Athena はそのパーティションを射影します。Athena はエラーをスローし、データを返しません。ただし、空のパーティションが多すぎる場合、従来の AWS Glue パーティションに比べてパフォーマンスが低下する可能性があります。射影パーティションの半分以上が空の場合は、従来のパーティションを使用することをお勧めします。
- パーティション射影に定義された範囲境界を超える値のクエリは、エラーを返しません。その代わりに、クエリが実行されても行は返されません。たとえば、2020 年に開始される時間関連データがあり、`'projection.timestamp.range'='2020/01/01,NOW'` として定義されている場合、`SELECT * FROM table-name WHERE timestamp = '2019/02/02'` のようなクエリは正常に完了しますが、行は返されません。
- パーティション射影は、テーブルが Athena を通じてクエリされる場合以外は使用できません。同じテーブルが Amazon Redshift Spectrum、Athena for Spark、Amazon EMR などの別のサービスを介して読み取られる場合、標準のパーティションメタデータが使用されます。
- パーティション射影は DML 限定の機能であるため、`SHOW PARTITIONS` は、Athena によって射影されていても、AWS Glue カタログまたは外部の Hive メタストアには登録されていないパーティションをリストしません。
- Athena は、パーティションの射影の設定としてビューのテーブルプロパティを使用しません。この制限を回避するには、ビューが参照するテーブルのテーブルプロパティでパーティション射影を構成して有効にします。
- Lake Formation データフィルターは、Athena でのパーティション射影では使用できません。

動画

以下の動画は、パーティション射影を使用して Athena でのクエリのパフォーマンスを向上させる方法を紹介するものです。

Amazon Athena でのパーティション射影

トピック

- パーティション射影のセットアップ
- パーティション射影用にサポートされている型

- ID の動的なパーティション化
- Amazon Data Firehose 例

パーティション射影のセットアップ

テーブルのプロパティでのパーティション射影のセットアップは、次の 2 つのステップで行います。

1. 各パーティション列のデータ範囲と関連するパターンを指定するか、カスタムテンプレートを使用します。
2. テーブルのパーティション射影を有効にします。

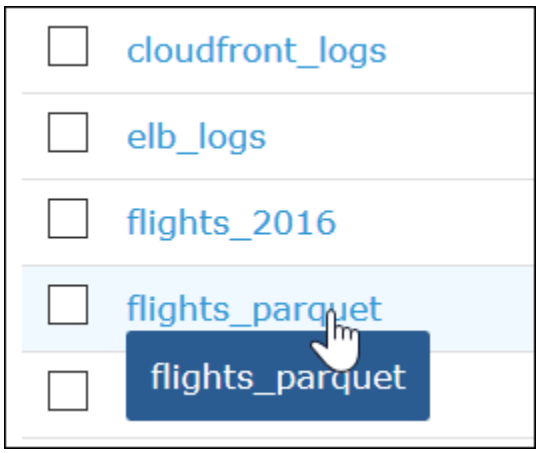
> **Note**
>
> パーティション射影プロパティを既存のテーブルに追加する前に、パーティション射影プロパティをセットアップするパーティションの列がテーブルスキーマにすでに存在している必要があります。パーティション列がまだ存在しない場合、既存のテーブルに手動でパーティション列を追加する必要があります。AWS Glue では、この手順を自動的に実行しません。

このセクションでは、AWS Glue にテーブルプロパティを設定する方法について説明します。これらを設定するには、AWS Glue コンソール、Athena CREATE TABLE クエリ、または AWS Glue API オペレーションを使用できます。次の手順は、AWS Glue コンソールでプロパティを設定する方法を示しています。

AWS Glue コンソールを使用してパーティション射影を設定して有効化する

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Tables] (テーブル) タブを選択します。

   [Tables] (テーブル) タブでは、既存のテーブルを編集する、または [Add tables] (テーブルの追加) を選択して新しいテーブルを作成できます。テーブルを手動またはクローラで追加する方法については、「AWS Glue デベロッパーガイド」の「AWS Glue コンソールでのテーブルの使用」を参照してください。
3. テーブルのリストで、編集するテーブルのリンクを選択します。

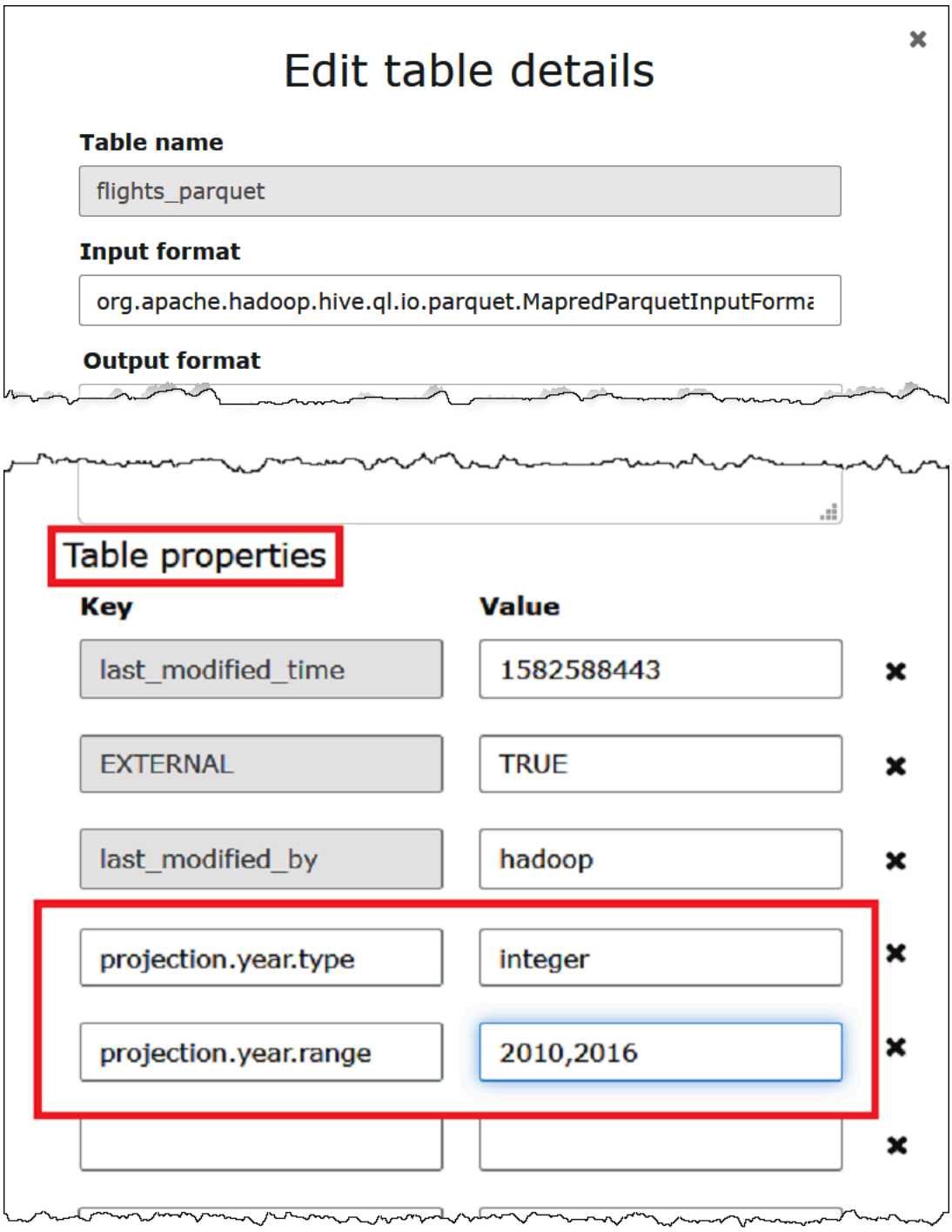


4. [Actions] (アクション)、[Edit table] (テーブルの編集) の順に選択します。
5. [Edit table] (テーブルの編集) ページの [Table properties] (テーブルプロパティ) セクションで、パーティション化された列ごとに以下のキーと値のペアを追加します。

   a. [キー] に、`projection.`*`columnName`*`.type` を追加します。

   b. [値] に、サポートされている型 (`enum`、`integer`、`date`、`injected`) のいずれかを追加します。詳細については、「パーティション射影用にサポートされている型」を参照してください。
6. 「パーティション射影用にサポートされている型」のガイダンスに従って、設定要件に従って追加のキーと値のペアを追加します。

   次のテーブル設定例では、パーティション射影用に `year` 列を設定し、返すことのできる値を 2010 ~ 2016 の範囲に制限します。

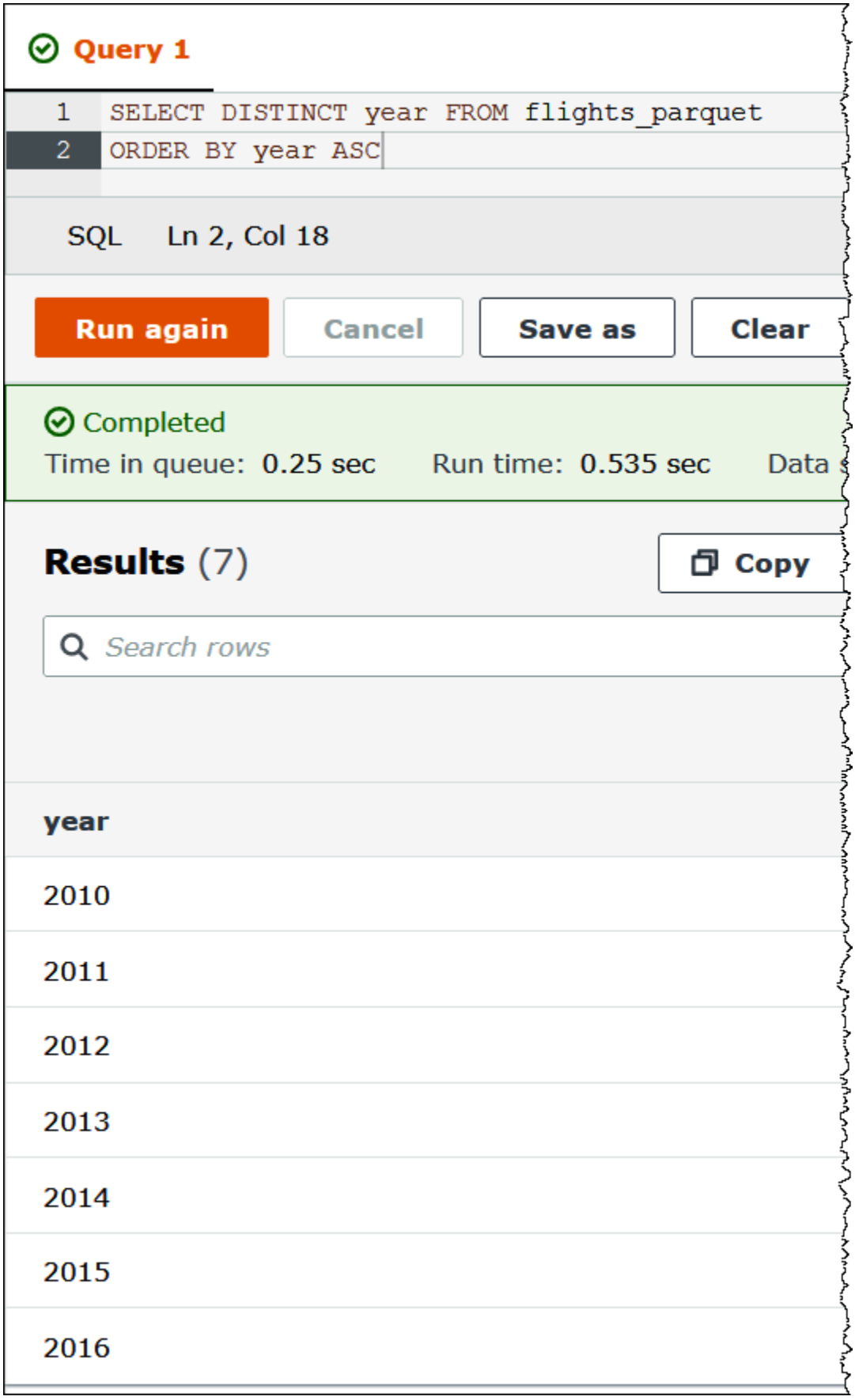


7. キーと値のペアを追加して、パーティション射影を有効にします。[Key] (キー) には `projection.enabled` と入力し、その [Value] (値) には `true` と入力します。

> **Note**
>
> `projection.enabled` を `false` に設定することで、このテーブルのパーティション射影をいつでも無効にできます。

8. 完了したら、[Save] を選択します。
9. Athena クエリエディタで、テーブル用に設定した列でのクエリをテストします。

   次のクエリ例では、`SELECT DISTINCT` を使用して、`year` 列から一意の値を返します。データベースには 1987 年から 2016 年のデータが含まれていますが、`projection.year.range` プロパティで返される値が 2010 年から 2016 年に制限されています。


Query 1
1 SELECT DISTINCT year FROM flights_parquet
2 ORDER BY year ASC
SQL Ln 2, Col 18
Run again
Cancel
Save as
Clear
Completed
Time in queue: 0.25 sec
Run time: 0.535 sec
Data
Results (7)
Copy
Search rows
year
2010
2011
2012
2013
2014
2015
2016

> **Note**
>
> `projection.enabled` を `true` に設定したが、1 つ以上のパーティション列を設定できない場合は、次のようなエラーメッセージが表示されます。`HIVE_METASTORE_ERROR: Table database_name.table_name is configured for partition projection, but the following partition columns are missing projection configuration: [column_name] (table database_name.table_name).`

カスタム S3 ストレージの場所の指定

AWS Glue でテーブルプロパティを編集するときは、射影されたパーティションのカスタム Amazon S3 パステンプレートを指定することもできます。カスタムテンプレートは、Athena が通常の `.../column=value/...` パターンに従わないカスタム Amazon S3 ファイルの場所に、パーティション値を適切にマッピングすることを可能にします。

カスタムテンプレートの使用はオプションです。ただし、カスタムテンプレートを使用する場合、各パーティション列のプレースホルダをテンプレートに含める必要があります。テンプレート化された場所は、パーティション化されたデータファイルがパーティションごとの「フォルダ」に格納されるように、スラッシュで終わる必要があります。

カスタムパーティションの場所のテンプレートを指定するには

1. 手順に従って、AWS Glue コンソールを使用して、パーティション射影の設定と有効化を行い、次のようにカスタムテンプレートを指定するキーと値のペアを追加します。
   a. [キー] に「`storage.location.template`」と入力します。
   b. [値] で、すべてのパーティション列のプレースホルダを含む場所を指定します。各プレースホルダ (および S3 パス自体) が 1 つのスラッシュで終了していることを確認します。

      次のテンプレート値の例では、パーティション列 a、b、c があるテーブルを想定しています。

      ```
      s3://DOC-EXAMPLE-BUCKET/table_root/a=${a}/${b}/some_static_subdirectory/${c}/
      ```

```
s3://DOC-EXAMPLE-BUCKET/table_root/c=${c}/${b}/some_static_subdirectory/${a}/
${b}/${c}/${c}/
```

同じテーブルの場合、次のテンプレート値の例は、列 c のプレースホルダが含まれていないため無効です。

```
s3://DOC-EXAMPLE-BUCKET/table_root/a=${a}/${b}/some_static_subdirectory/
```

2. [Apply] を選択します。

パーティション射影用にサポートされている型

テーブルには、enum、integer、date,、または injected パーティション列の型の任意の組み合わせを使用できます。

列挙型

値が列挙セット (空港コードや AWS リージョン など) のメンバーであるパーティション列には、enum 型を使用します。

テーブルでパーティションプロパティを次のように定義します。

プロパティ名	値の例	説明
projection. *columnName* .type	enum	必須。列 *columnName* に使用する射影型です。列挙型の使用を通知するには、値を enum (大文字と小文字を区別しない) にする必要があります。先頭と末尾に空白文字を使用できます。
projection. *columnName* .values	A,B,C,D,E,F,G,Unknown	必須。列 *columnName* に対する列挙されたパーティション値のカンマ区切りリスト。空白は列挙値の一部とみなされます。

> **Note**
>
> ベストプラクティスとして、enum ベースのパーティション射影の使用は数 10 個以下に制限することが推奨されます。enum 射影に固有の制限はありませんが、テーブルのメタデータを gzip に圧縮するときには、合計サイズが約 1MB という AWS Glue の制限を超えることはできません。この制限は、列名、場所、およびストレージ形式といったテーブルの主要部分全体で共有されることに注意してください。enum 射影で使用している一意の ID が数十個ある場合は、サロゲートフィールドにバケット化する一意の値を少なくするなど別のアプローチを検討してください。カーディナリティをトレードオフすることで、enum フィールドの一意の値の数を制御できます。

整数型

使用できる値が定義された範囲内の整数として解釈可能なパーティション列には、整数型を使用します。射影整数列は現在、Java 符号付き長整数値 (-2^{63} ~ $2^{63}-1$。両端の値を含みます) の範囲に制限されています。

プロパティ名	値の例	説明
`projection.`*`columnName`*`.type`	`integer`	必須。列 *`columnName`* に使用する射影型です。整数型の使用を通知するには、値を `integer` (大文字と小文字を区別しない) にする必要があります。先頭と末尾に空白文字を使用できます。

`projection.`*`columnName`*`.range`	`0,10`
	
	`-1,8675309`
	
	`0001,9999`
		必須。*`columnName`* 列のクエリによって返される最小値と最大値の範囲を示す 2 要素カンマ区切りリスト。値はハイフンではなくカンマで区切る必要があることに注意してください。両端の値が含まれ、負の値でもかまいません。また、先頭をゼロにすること

プロパティ名	値の例	説明
		もできます。先頭と末尾に空白文字を使用できます。
projection. *columnName* .interval	1 5	オプション。列 *columnName* の連続するパーティション値の間隔を指定する正の整数。たとえば、range 値が「1,3」で、interval 値が「1」の場合、値 1、2、3 が生成されます。range 値が同じで interval が「2」の場合、値 1 と 3 が生成され、2 はスキップされます。先頭と末尾に空白文字を使用できます。デフォルト は 1 です。
projection. *columnName* .digits	1 5	オプション。列 *columnName* のパーティション値の最終表現に含める桁数を指定する正の整数。たとえば、range 値が「1,3」で digits 値が「1」の場合、値 1、2、3 が生成されます。range 値が同じで digits 値が「2」の場合、値 01、02、03 が生成されます。先頭と末尾に空白文字を使用できます。デフォルトでは、静的な桁数も先頭のゼロもありません。

日付型

定義された範囲内の日付 (オプションで時刻を含む) として解釈可能な値を持つパーティション列には、日付型を使用します。

> **Important**
>
> 射影される日付列は、クエリの実行時に協定世界時 (UTC) で生成されます。

プロパティ名	値の例	説明
`projection.`*`columnName`*`.type`	`date`	必須。列 *`columnName`* に使用する射影型です。日付タイプの使用を通知するには、値を `date` (大文字と小文字を区別しない) にする必要があります。先頭と末尾に空白文字を使用できます。

`projection.`*`columnName`*`.range`	`201701,201812`
	
	`01-01-2010,12-31-2018`
	
	`NOW-3YEARS,NOW`
	
	`201801,NOW+1MONTH`
		必須。*`columnName`* 列の最小および最大 `range` 値を提供する、2 要素のカンマ区切りのリストです。両端の値が含まれ、Java の `java.time.*` 日付型と互換性のある任意の形式を使用できます。最小値と最大値のどちらも同じ形式を使用する必要があります。`.format` プロパティで指定する形式は、これらの値に使用される形式にする必要があります。
		
		この列には、次の正規表現パターンの形式で相対日付文字列を含めることもできます。
		
		`\s*NOW\s*(([\+\-])\s*([0-9]+)\s*(YEARS?|MONTHS?|WEEKS?|DAYS?|HOURS?|MINUTES?|SECONDS?)\s*)?`
		
		空白は使用できますが、日付リテラルは日付文字列自体の一部と見なされます。

`projection.`*`columnName`*`.format`	`yyyyMM`
	
	`dd-MM-yyyy`
		必須。Java の日付フォーマット DateTimeFormatter に基づく日付形式文字列。サポートされている任意の `Java.time.*` 型を使用できます。

プロパティ名	値の例	説明
	dd-MM-yyyy-HH-mm-ss	
projection. *columnName* .interval	1 5	列 *columnName* の連続するパーティション値の間隔を指定する正の整数。たとえば、range 値が 2017-01,2018-12 、interval 値が 1、interval.unit 値が MONTHS の場合、2017-01、2017-02、2017-03 というように値が生成されます。range 値が同じで interval 値が 2、interval.unit 値が MONTHS の場合、2017-01、2017-03、2017-05 というように値が生成されます。先頭と末尾に空白文字を使用できます。 指定された日付が単一日または単一月の精度である場合、interval はオプションであり、デフォルトではそれぞれ 1 日または 1 か月です。それ以外の場合、interval が必要です。
projection. *columnName* .interval.unit	YEARS MONTHS WEEKS DAYS HOURS MINUTES SECONDS MILLIS	ChronoUnit のシリアル化された形式を表す時間単位の単語。使用できる値は YEARS、MONTHS、WEEKS、DAYS、HOURS、MINUTES、です。これらの値の大文字と小文字は区別されません。 指定された日付が単一日または単一月の精度である場合、interval.unit はオプションであり、デフォルトではそれぞれ 1 日または 1 か月です。それ以外の場合は、interval.unit が必要です。

挿入型

使用できる値をいくつかの論理範囲内で手動で生成することができないが、クエリの WHERE 句で単一の値として指定されるパーティション列には、挿入型を使用します。

次の点に留意することが重要です。

- 挿入列ごとにフィルタ式が指定されていない場合、挿入列でのクエリは失敗します。
- 挿入された列のフィルター式に複数の値を持つクエリは、値が論理和である場合にのみ成功します。
- string 型の列のみがサポートされています。

プロパティ名	Value	説明
projection. *columnName* .type	inje	必須。列 *columnName* に使用する射影型です。string 型のみがサポートされています。値 injected (大文字と小文字は区別されません) を指定する必要があります。先頭と末尾に空白文字を使用できます。

詳細については、「injected 射影型を使用する」を参照してください。

ID の動的なパーティション化

カーディナリティの高いプロパティによってデータが分割されている場合や、値が事前にわからない場合は、injected 射影型を使用できます。このようなプロパティの例としては、ユーザー名、デバイスや製品の ID などがあります。injected 射影型を使用してパーティションキーを設定すると、Athena はクエリ自体の値を使用して、読み取られるパーティションのセットを計算します。

Athena が injected プロジェクションタイプで設定されたパーティションキーを持つテーブルでクエリを実行できるようにするには、以下が満たされている必要があります。

- クエリには、パーティションキーの値が少なくとも 1 つ含まれている必要があります。
- 値は、データを読み取らなくても評価できるリテラルまたは式でなければなりません。

これらの基準のいずれかが満たされない場合、クエリは次のエラーで失敗します。

CONSTRAINT_VIOLATION: injected 射影型のパーティション列 *column_name* では、WHERE 句に静的な等号条件のみが含まれており、このような条件が少なくとも 1 つ存在する必要があります。

injected 射影型を使用する

IoT デバイスからのイベントで構成され、デバイスの ID で分割されたデータセットがあるとします。このデータセットには次の特徴があります。

- デバイス ID はランダムに生成されています。
- 新しいデバイスは頻繁にプロビジョニングされます。
- 現在、数十万台のデバイスがあり、今後は数百万台になる見込みです。

このデータセットを従来のメタストアで管理するのは容易ではありません。データストレージとメタストア間でパーティションを同期させることは難しく、クエリ計画中はパーティションのフィルタリングに時間がかかることがあります。ただし、パーティションプロジェクションを使用するようにテーブルを設定し、injected 射影型を使用する場合には、メタストア内のパーティションを管理する必要がないことと、クエリでパーティションメタデータを検索する必要がないことの 2 つの利点があります。

次の CREATE TABLE の例では、先ほど説明したデバイスイベントデータセットのテーブルを作成します。このテーブルでは、injected 射影型を使用しています。

```
CREATE EXTERNAL TABLE device_events (
   event_time TIMESTAMP,
  data STRING,
  battery_level INT
)
PARTITIONED BY (
  device_id STRING
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
  "projection.enabled" = "true",
  "projection.device_id.type" = "injected",
  "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${device_id}"
)
```

次のクエリ例では、特定の 3 つのデバイスから 12 時間にわたって受信したイベントの数を調べます。

```
SELECT device_id, COUNT(*) AS events
FROM device_events
WHERE device_id IN (
  '4a770164-0392-4a41-8565-40ed8cec737e',
  'f71d12cf-f01f-4877-875d-128c23cbde17',
  '763421d8-b005-47c3-ba32-cc747ab32f9a'
)
AND event_time BETWEEN TIMESTAMP '2023-11-01 20:00' AND TIMESTAMP '2023-11-02 08:00'
GROUP BY device_id
```

このクエリを実行すると、Athena は device_id パーティションキーの 3 つの値を確認し、それらを使用してパーティションの場所を計算します。Athena は storage.location.template プロパティの値を使用して以下の場所を生成します。

- s3://DOC-EXAMPLE-BUCKET/*prefix*/4a770164-0392-4a41-8565-40ed8cec737e
- s3://DOC-EXAMPLE-BUCKET/*prefix*/f71d12cf-f01f-4877-875d-128c23cbde17
- s3://DOC-EXAMPLE-BUCKET/*prefix*/763421d8-b005-47c3-ba32-cc747ab32f9a

パーティションの射影設定から storage.location.template プロパティを除外した場合、Athena は Hive 形式のパーティション分割を使用して、LOCATION の値 (例: s3://DOC-EXAMPLE-BUCKET/*prefix*/device_id=4a770164-0392-4a41-8565-40ed8cec737e) に基づいてパーティションの場所を射影します。

Amazon Data Firehose 例

Firehose を使用して Amazon S3 にデータを配信する場合、デフォルト設定は、次の例にあるようなキーを持つオブジェクトを書き込みます。

```
s3://DOC-EXAMPLE-BUCKET/prefix/yyyy/MM/dd/HH/file.extension
```

新しいデータが到着したときにパーティションを AWS Glue Data Catalog に追加する必要なく、クエリ時にパーティションを自動的に検索する Athena テーブルを作成するには、パーティション射影を使用できます。

以下の CREATE TABLE 例は、デフォルトの Firehose 設定を使用しています。

```
CREATE EXTERNAL TABLE my_ingested_data (
```

```
 ...
)
...
PARTITIONED BY (
 datehour STRING
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.datehour.type" = "date",
 "projection.datehour.format" = "yyyy/MM/dd/HH",
 "projection.datehour.range" = "2021/01/01/00,NOW",
 "projection.datehour.interval" = "1",
 "projection.datehour.interval.unit" = "HOURS",
 "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${datehour}/"
)
```

CREATE TABLE ステートメントの TBLPROPERTIES 句は、Athena に次のように指定します。

- テーブルのクエリ時にパーティション射影を使用する
- パーティションキー datehour は date 型である (オプションで時刻を含む)
- 日付の書式設定方法
- 日付時刻の範囲。値はハイフンではなくカンマで区切る必要があることに注意してください。
- Amazon S3 でのデータの検索先。

テーブルをクエリすると、Athena は datehour の値を計算し、ストレージロケーションテンプレートを使用して、パーティションの場所のリストを生成します。

date 型の使用

射影パーティションキーに date 型を使用する場合、範囲を指定する必要があります。Firehose 配信ストリームが作成される前の日付に関するデータがないため、作成日を開始点として使用できます。また、将来の日付のデータがないため、最後として NOW という特別なトークンを使用できます。

CREATE TABLE の例では、開始日は 2021 年 1 月 1 日 UTC 午前 0 時と指定されています。

> **Note**
>
> Athena が既存のパーティションのみを検索できるように、可能な限りデータと一致する範囲を設定します。

サンプルテーブルでクエリを実行すると、Athena は、datehourパーティションキーの条件を、値を生成する範囲と組み合わせて使用します。次のクエリについて考えます。

```
SELECT *
FROM my_ingested_data
WHERE datehour >= '2020/12/15/00'
AND datehour < '2021/02/03/15'
```

SELECT クエリの最初の条件では、CREATE TABLE ステートメントで指定された日付範囲の開始前の日付を使用しています。パーティション射影の設定では 2021 年 1 月 1 日より前の日付にはパーティションが指定されていないため、Athena は次の場所でのみデータを検索し、クエリ内のそれ以前の日付を無視します。

```
s3://DOC-EXAMPLE-BUCKET/prefix/2021/01/01/00/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/01/01/01/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/01/01/02/
...
s3://DOC-EXAMPLE-BUCKET/prefix/2021/02/03/12/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/02/03/13/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/02/03/14/
```

同様に、クエリが 2021 年 2 月 3 日 15:00 より前の日付と時刻で実行された場合、最後のパーティションに反映されているのは、クエリ条件の日時ではなく、現在の日付と時刻です。

最新のデータをクエリする場合は、Athena が将来の日付を生成しないという事実を活用し、次の例のように、開始 datehour のみを指定することができます。

```
SELECT *
FROM my_ingested_data
WHERE datehour >= '2021/11/09/00'
```

パーティションキーの選択

パーティション射影でパーティションの場所をパーティションキーにマッピングする方法を指定できます。前のセクションの CREATE TABLE の例では、日付と時刻を datehour という 1 つのパーティションキーにまとめましたが、他のスキームも可能です。例えば、パーティションキーを year、month、day、hour に分けてテーブルを構成することもできます。

ただし、日付を年、月、日に分割すると、date パーティション投影タイプは使用できなくなります。別の方法として、日付と時間を分離して date パーティション投影タイプを引き続き活用する方法がありますが、時間の範囲を指定するクエリは読みやすくします。

それを考慮して、次の CREATE TABLE の例では、日付と hour を分けています。date は SQL の予約語であるため、この例では日付を表すパーティションキーの名前として day を使用します。

```
CREATE EXTERNAL TABLE my_ingested_data2 (
 ...
)
...
PARTITIONED BY (
 day STRING,
 hour INT
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.day.type" = "date",
 "projection.day.format" = "yyyy/MM/dd",
 "projection.day.range" = "2021/01/01,NOW",
 "projection.day.interval" = "1",
 "projection.day.interval.unit" = "DAYS",
 "projection.hour.type" = "integer",
 "projection.hour.range" = "0,23",
 "projection.hour.digits" = "2",
 "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${day}/${hour}/"
)
```

CREATE TABLE ステートメントの例では、hour は独立したパーティションキーで、整数に設定されています。hour パーティションキーの設定では、0 ~ 23 の範囲が指定され、Athena がパーティションの場所を生成するときに、hour を 2 桁でフォーマットする必要があります。

my_ingested_data2 テーブルのクエリは次の例のようになります。

```
SELECT *
FROM my_ingested_data2
WHERE day = '2021/11/09'
AND hour > 3
```

パーティションキーの型とパーティション射影の型

最初の CREATE TABLE の例の datehour キーは、パーティション射影の設定では date として設定されていますが、パーティションキーの型は string なので、注意してください。2 番目の例の day についても同様です。パーティション射影に設定されている型は、Athena がパーティションの場所を生成するときの値のフォーマット方法のみを指示します。指定する型によって、パーティションキーの型は変更されません。クエリでは、datehour と day は string 型です。

クエリに day = '2021/11/09' のような条件が含まれている場合、Athena はパーティション射影設定で指定された日付形式を使用して、式の右辺の文字列を解析します。Athena は、日付が設定された範囲内であることを確認した後、日付形式を再び使用して日付を文字列として保存場所テンプレートに挿入します。

同様に、day > '2021/11/09' のようなクエリ条件の場合、Athena は右辺を解析し、設定された範囲内で一致するすべての日付のリストを生成します。次に、日付形式を使用して各日付を保存場所テンプレートに挿入し、パーティションの場所のリストを作成します。

day > '2021-11-09' や day > DATE '2021-11-09' と同じ条件を書くと動作しません。最初のケースでは、日付形式が一致せず (スラッシュではなくハイフンになっていることに注意してください)、2 番目のケースでは、データ型が一致しません。

カスタムプレフィックスと動的パーティション化の使用

Firehose は、カスタムプレフィックスと動的パーティショニングを使用して設定できます。これらの機能を使用して、Amazon S3 キーを設定し、ユースケースをより適切にサポートするパーティション化スキームを設定できます。また、これらのパーティション化スキームでパーティション射影を使用し、それに応じて設定することもできます。

例えば、カスタムプレフィックス機能を使用して、デフォルトの yyyy/MM/dd/HH スキームではなく、ISO 形式の日付を持つ Amazon S3 キーを取得できます。

以下の例にあるように、カスタムプレフィックスと動的パーティショニングを組み合わせて、Firehose メッセージから customer_id のようなプロパティを抽出することもできます。

```
prefix/!{timestamp:yyyy}-!{timestamp:MM}-!{timestamp:dd}/!
{partitionKeyFromQuery:customer_id}/
```

Amazon S3 プレフィックスを使用すると、Firehose 配信ストリームは、s3://DOC-EXAMPLE-BUCKET/prefix/2021-11-01/customer-1234/file.extension などのキーにオブジェクトを書き込みます。customer_id のようなプロパティの場合、事前に値が分かっていない場合もあるので、パーティション射影型 injected を使用し、次のような CREATE TABLE ステートメントを使用できます。

```
CREATE EXTERNAL TABLE my_ingested_data3 (
 ...
)
...
PARTITIONED BY (
 day STRING,
 customer_id STRING
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.day.type" = "date",
 "projection.day.format" = "yyyy-MM-dd",
 "projection.day.range" = "2021-01-01,NOW",
 "projection.day.interval" = "1",
 "projection.day.interval.unit" = "DAYS",
 "projection.customer_id.type" = "injected",
 "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${day}/${customer_id}/"
)
```

injected 型のパーティションキーを持つテーブルに対してクエリを実行する場合、クエリにそのパーティションキーの値を含める必要があります。my_ingested_data3 テーブルのクエリは次の例のようになります。

```
SELECT *
FROM my_ingested_data3
WHERE day BETWEEN '2021-11-01' AND '2021-11-30'
AND customer_id = 'customer-1234'
```

ISO 形式の日付

day パーティションキーの値は ISO フォーマットされているので、次の例のように、day パーティションキーに、STRING ではなく、DATE 型を使用することもできます。

```
PARTITIONED BY (day DATE, customer_id STRING)
```

クエリを実行するとき、この戦略では、次の例のように、解析やキャストを行わずに、パーティションキーで日付関数を使用できます。

```
SELECT *
FROM my_ingested_data3
WHERE day > CURRENT_DATE - INTERVAL '7' DAY
AND customer_id = 'customer-1234'
```

> **Note**
>
> DATE 型のパーティションキーを指定するには、カスタムプレフィックス機能を使用して ISO 形式の日付を含む Amazon S3 キーを作成したことを前提としています。デフォルトの Firehose 形式、yyyy/MM/dd/HH を使用している場合は、以下の例にあるように、対応するテーブルプロパティが date 型であっても、パーティションキーを string 型として指定する必要があります。
>
> ```
> PARTITIONED BY (
>   `mydate` string)
> TBLPROPERTIES (
>   'projection.enabled'='true',
>    ...
>   'projection.mydate.type'='date',
>   'storage.location.template'='s3://DOC-EXAMPLE-BUCKET/prefix/${mydate}')
> ```

クエリ結果からのテーブルの作成 (CTAS)

CREATE TABLE AS SELECT (CTAS) クエリは、別のクエリからの SELECT ステートメントの結果から、Athena で新しいテーブルを作成します。Athena は、CTAS ステートメントによって作成されたデータファイルを Amazon S3 の指定された場所に保存します。構文については、「CREATE TABLE AS」を参照してください。

CREATE TABLE AS は CREATE TABLE DDL ステートメントと SELECT DML ステートメントを組み合わせるため、技術的には DDL と DML の両方を含んでいます。ただし、Service Quotas の観点から、Athena の CTAS クエリは DML として扱われることに注意してください。Athena の Service Quotasの詳細については、「Service Quotas」を参照してください。

以下の目的で CTAS クエリを使用します。

- raw データセットのクエリを繰り返さずに、1 回のステップでクエリの結果からテーブルを作成する。これにより raw データセットの操作が簡単になります。
- クエリ結果を変換し、テーブルを Apache Iceberg などの他のテーブル形式に移行します。これにより、Athena でのクエリパフォーマンスが向上し、クエリコストが削減されます。詳細については、「Iceberg テーブルの作成」を参照してください。
- クエリ結果を Parquet や ORC などのストレージ形式に変換する。これにより、Athena でのクエリパフォーマンスが向上し、クエリコストが削減されます。詳細については、「列指向ストレージ形式」を参照してください。
- 必要なデータのみが含まれる既存のテーブルのコピーを作成する。

トピック

- CTAS クエリに関する考慮事項と制約事項
- コンソールでの CTAS クエリの実行
- Athena におけるパーティション化とバケット化
- CTAS クエリの例
- ETL およびデータ分析での CTAS および INSERT INTO の使用
- CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する

CTAS クエリに関する考慮事項と制約事項

以下のセクションでは、Athena で CREATE TABLE AS SELECT (CTAS) クエリを使用する際に留意すべき考慮事項と制限事項について説明します。

CTAS クエリ構文

CTAS クエリ構文は、テーブル作成用の CREATE [EXTERNAL] TABLE 構文とは異なります。「CREATE TABLE AS」を参照してください。

CTAS クエリとビューの違い

CTAS クエリは Amazon S3 の指定された場所に新しいデータ書き込みますが、ビューはデータを書き込みません。

CTAS クエリ結果の場所

ワークグループがクエリ結果の場所に関するクライアント側の設定を上書きする場合、Athena は `s3://DOC-EXAMPLE-BUCKET/tables/<query-id>/` の場所にテーブルを作成します。ワークグループに指定されたクエリ結果の場所を表示するには、ワークグループの詳細を表示します。

ワークグループがクエリ結果の場所を上書きしない場合は、CTAS クエリの構文 `WITH (external_location ='s3://DOC-EXAMPLE-BUCKET/')` を使用して、CTAS クエリ結果の保存場所を指定できます。

> **Note**
>
> `external_location` プロパティは、空の場所を指定する必要があります。CTAS クエリはバケットのパス位置 (プレフィックス) が空であるかどうかをチェックして、その場所に既にデータが含まれる場合は、データを上書きしません。同じ場所を再度使用するには、バケット内のキープレフィックスの場所のデータを削除します。

`external_location` 構文を省略しており、ワークグループ設定を使用していない場合、Athena はクエリ結果の場所にクライアント側の設定を使用し、`s3://DOC-EXAMPLE-BUCKET/<Unsaved-or-query-name>/<year>/<month/<date>/tables/<query-id>/` の場所にテーブルを作成します。

孤立したファイルの検索

`CTAS` または `INSERT INTO` ステートメントが失敗した場合、孤立したデータがデータの場所に残っている可能性があります。Athena は場合によってはバケットからデータまたは部分データを削除しないため、後続のクエリでこの部分データが読み込み可能になる可能性があります。検査または削除する孤立したファイルを見つけるには、Athena に用意されているデータマニフェストファイルを使用して、書き込まれるファイルのリストを追跡できます。詳細については、「クエリ出力ファイルの識別」および「DataManifestLocation」を参照してください。

ORDER BY 句は無視されます

CTAS クエリでは、Athena はクエリの `SELECT` 部分にある `ORDER BY` 句を無視します。

SQL 仕様 (ISO 9075 Part 2) では、クエリ式で指定されたテーブル行の順序が保証されるのは、ORDER BY 句が直後に含まれるクエリ式だけです。SQL のテーブルはどのような場合でも本来は順序付けられておらず、ORDER BY をサブクエリ句に追加してもクエリのパフォーマンスが低下するだけで、出力は順序付けられません。したがって、Athena CTAS クエリでは、データが書き込まされた際に ORDER BY 句によって指定された順序が保持される保証はありません。

クエリの結果を保管する形式

CTAS クエリの結果は、デフォルトで Parquet 形式で保存されます (データストレージ形式を指定しない場合)。CTAS の結果は、PARQUET、ORC、AVRO、JSON、および TEXTFILE で保存できます。CTAS TEXTFILE 形式では複数文字の区切り文字がサポートされません。CTAS クエリでは形式の変換を解釈するために SerDe を指定する必要はありません。「Example: Writing query results to a different format」を参照してください。

圧縮形式

JSON 形式と TEXTFILE 形式の CTAS クエリ結果には、GZIP 圧縮が使用されます。Parquet の場合、デフォルトは GZIP で、GZIP または SNAPPY も使用可能です。ORC の場合、デフォルトは ZLIB で、LZ4、SNAPPY、ZLIB、または ZSTD も使用できます。圧縮を指定する CTAS の例については、Example: Specifying data storage and compression formats を参照してください。Athena での圧縮の詳細については、「Athena での圧縮のサポート」を参照してください。

パーティションとバケットの制限

CTAS クエリの結果データをパーティションおよびバケット化することができます。詳細については、「Athena におけるパーティション化とバケット化」を参照してください。CTAS を使用してパーティションテーブルを作成する場合、Athena はパーティションの書き込みを 100 個に制限します。

宛先テーブルのプロパティを指定する WITH 句の最後に、パーティションとバケット化の述語を含めます。詳細については、Example: Creating bucketed and partitioned tablesおよびAthena におけるパーティション化とバケット化を参照してください。

100 パーティションの制限を回避する方法については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。

Encryption

CTAS クエリの結果は Amazon S3 で暗号化することができます。これは Athena で他のクエリ結果を暗号化する方法と似ています。詳細については、「Amazon S3 に保存された Athena のクエリ結果の暗号化」を参照してください。

予想されるバケット所有者

CTAS ステートメントの場合、予想されるバケット所有者の設定は、Amazon S3 内の送信先テーブルの場所には適用されません。予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。詳細については、「Athena コンソールを使用したクエリ結果の場所の指定」を参照してください。

データ型

CTAS クエリの列のデータ型は元のクエリに指定されているものと同じです。

コンソールでの CTAS クエリの実行

Athena コンソールでは、別のクエリから CTAS クエリを作成できます。

別のクエリから CTAS クエリを作成する

1. Athena コンソールのクエリエディターでクエリを実行します。
2. クエリエディタの最下部で [Create] (作成) オプションを選択し、次に [Table from query] (クエリからのテーブル) を選択します。
3. 次のように、[Create table as select] (選択したようにテーブルを作成) フォームの各フィールドに入力します。

    a. [Table name] (テーブル名) に新しいテーブルの名前を入力します。小文字とアンダースコアのみを使用します (例: `my_select_query_parquet`)。

    b. [Database configuration] (データベース設定) のオプションで、既存のデータベースを選択するか、データベースを作成するかを選択します。

    c. (オプション) [Result configuration] (結果設定) の [Location of CTAS query results] (CTAS クエリ結果の場所) で、ワークグループのクエリ結果の場所の設定がこのオプションを上書きしない場合は、次のいずれかを実行します。

    - 検索ボックスに既存の S3 の場所へのパスを入力するか、[Browse S3] (S3 の参照) を選択してリストから場所を選択します。

- [View] (表示) を選択して Amazon S3 コンソールの [Buckets] (バケット) ページを開き、既存のバケットに関する詳細情報を表示したり、独自の設定でバケットを選択または作成したりできます。

データの出力先となる Amazon S3 内の空の場所を指定します。データが既に存在する場所を指定すると、クエリがエラーを起こし失敗します。

ワークグループのクエリ結果の場所の設定によってこの場所の設定が上書きされる場合、Athena で `s3://DOC-EXAMPLE-BUCKET/tables/query_id/` の場所にテーブルが作成されます。

d. [Data format] (データ形式) で、データの形式を指定します。

- テーブルタイプ – Athena のデフォルトのテーブルタイプは Apache Hive です。
- ファイル形式 – CSV、TSV、JSON、Parquet、ORC などのオプションから選択します。Parquet 形式と ORC 形式の詳細については、「列指向ストレージ形式」を参照してください。
- 書き込み圧縮 – (オプション) 圧縮形式を選択します。Athena は、複数の圧縮形式を使用するテーブルからの読み込みなど、データの読み書きのためのさまざまな圧縮形式をサポートしています。例えば、一部の Parquet ファイルが Snappy で圧縮されており、他の Parquet ファイルは GZIP で圧縮されているといった、Parquet ファイル形式を使用するテーブル内のデータも、Athena は正常に読み込むことができます。同様なことが ORC、テキストファイル、および JSON のストレージ形式に対しても当てはまります。詳細については、「Athena での圧縮のサポート」を参照してください。
- パーティション – (オプション) パーティションする列を選択します。データをパーティションすると、各クエリによってスキャンされるデータの量が制限されるため、パフォーマンスが向上し、コストが削減されます。任意のキーでデータをパーティションに分割することができます。詳細については、「Athena でのデータのパーティション化」を参照してください。
- バケット – (オプション) バケット化する列を選択します。バケットとは、特定の列に基づいてデータを 1 つのパーティションにまとめる手法です。これらの列はバケットキーと呼ばれます。関連データを 1 つのバケット (パーティション内のファイル) にグループ化することで、Athena でスキャンされるデータ量を大幅に削減できるため、クエリのパフォーマンスが向上し、コストが削減されます。詳細については、「Athena におけるパーティション化とバケット化」を参照してください。

e. [Preview table query] (テーブルクエリのプレビュー) でクエリを確認します。クエリ構文については、「CREATE TABLE AS」を参照してください。

f. [Create table (テーブルの作成)] を選択します。

SQL テンプレートを使用して CTAS クエリを作成するには

クエリエディタで CTAS クエリを作成するには、`CREATE TABLE AS SELECT` テンプレートを使用します。

1. Athena コンソールで、[Tables and views] (テーブルとビュー) の横にある [Create table] (テーブルの作成) をクリックし、次に [CREATE TABLE AS SELECT] をクリックします。これにより、プレースホルダに値が設定された CTAS クエリが、クエリエディタに入力されます。
2. 必要に応じて、クエリエディタでクエリを編集します。クエリ構文については、「CREATE TABLE AS」を参照してください。
3. [Run] (実行) を選択します。

例については、「CTAS クエリの例」を参照してください。

Athena におけるパーティション化とバケット化

クエリを実行する際に Athena がスキャンする必要があるデータ量を減らすための 2 つの方法として、パーティション化とバケット化があります。パーティション化とバケット化は補完的な機能で、併用できます。スキャンするデータ量を減らすことは、パフォーマンスの向上とコストの削減につながります。Athena クエリのパフォーマンスに関する一般的なガイドラインについては、「Amazon Athena のパフォーマンスチューニング Tips トップ 10」を参照してください。

パーティション化とは

パーティション化とは、データの特定のプロパティに基づいて Amazon S3 上にあるディレクトリ (または「プレフィックス」) にデータを整理することを意味します。このようなプロパティはパーティションキーと呼ばれます。一般的なパーティションキーは、日付またはその他の時間単位 (年や月など) です。ただし、データセットは複数のキーでパーティション化できます。たとえば、製品の売上に関するデータは、日付、製品カテゴリ、市場ごとにパーティション化できます。

パーティション化する方法の決定

パーティションキーに適しているのは、クエリで常にまたは頻繁に使用され、カーディナリティが低いプロパティです。パーティションの数が多すぎる/少なすぎることについては、それぞれの欠点

があります。パーティションが多すぎると、ファイル数が増えるとオーバーヘッドが発生します。また、パーティション自体をフィルタリングすることによるオーバーヘッドもあります。パーティションが少なすぎると、クエリがより多くのデータをスキャンする必要性が多発します。

パーティションテーブルの作成

データセットをパーティション化すると、Athena でパーティションテーブルを作成できます。パーティションテーブルは、パーティションキーを含むテーブルです。CREATE TABLE を使用すると、テーブルにパーティションを追加します。CREATE TABLE AS を使用すると、Amazon S3 上で作成されたパーティションがテーブルに自動で追加されます。

CREATE TABLE ステートメントでは、PARTITIONED BY (*column_name data_type*) 句にパーティションキーを指定します。CREATE TABLE AS ステートメントでは、WITH (partitioned_by = ARRAY['*partition_key*']) 句に、または Iceberg テーブルの WITH (partitioning = ARRAY['*partition_key*']) にパーティションキーを指定します。パフォーマンス上の理由から、パーティションキーは常に STRING タイプにする必要があります。詳細については、「パーティションキーのデータ型として文字列を使用」を参照してください。

その他の CREATE TABLE および CREATE TABLE AS 構文の詳細については、「CREATE TABLE」と「CTAS テーブルのプロパティ」を参照してください。

パーティションテーブルのクエリ

パーティションテーブルをクエリすると、Athena はクエリ内の述語を使用してパーティションのリストをフィルタリングします。次に、一致するパーティションの場所を使用して、見つかったファイルを処理します。単純に、クエリ述語と一致しないパーティションのデータを読み取らないようにすることで、Athena がスキャンするデータ量を効率的に減らせます。

例

テーブルを sales_date と product_category でパーティション化していて、特定のカテゴリにおける 1 週間の総収益を知りたいとします。次の例のように、Athena がスキャンするデータ量が最小限になるように、sales_date 列と product_category 列に述語を含めます。

```
SELECT SUM(amount) AS total_revenue
FROM sales
WHERE sales_date BETWEEN '2023-02-27' AND '2023-03-05'
AND product_category = 'Toys'
```

日付別にパーティション化されているが詳細なタイムスタンプも含むデータセットがあるとします。

Iceberg テーブルではパーティションキーを宣言して列に関係を構築できますが、Hive テーブルではクエリエンジンは列とパーティションキーの関係に関するデータを持っていません。。このため、クエリが必要以上に多くのデータをスキャンしないように、クエリにある列とパーティションキーの両方に述語を含める必要があります。

たとえば、前の例における `sales` テーブルにも `TIMESTAMP` データ型の `sold_at` 列があるとします。特定の時間範囲の収益のみを求める場合は、クエリを次のように記述します:

```
SELECT SUM(amount) AS total_revenue
FROM sales
WHERE sales_date = '2023-02-28'
AND sold_at BETWEEN TIMESTAMP '2023-02-28 10:00:00' AND TIMESTAMP '2023-02-28
 12:00:00'
AND product_category = 'Toys'
```

Hive テーブルと Iceberg テーブルのクエリの違いに関する詳細については、「同じく時間分割されているタイムスタンプフィールドのクエリを作成する方法」を参照してください。

バケット化とは

バケット化は、データセットのレコードをバケットと呼ばれるカテゴリに整理する方法です。

この意味におけるバケットとバケット化は Amazon S3 バケットとは異なるため、混同しないでください。データバケットでは、プロパティと同じ値を含むレコードが同じバケットに入ります。レコードはバケット間で可能な限り均等に分散されるため、各バケットにはほぼ同じ量のデータが含まれます。

実際には、バケットはファイルであり、ハッシュ関数がレコードが入るバケットを決定します。バケット化されたデータセットには、パーティション/バケットごとに 1 つ以上のファイルがあります。ファイルが属するバケットはファイル名でエンコードされます。

バケット化のメリット

バケット化は、データセットが特定のプロパティによってバケット化されており、そのプロパティに特定の値があるレコードを取得する際に役立ちます。データはバケット化されているので、Athena は検索するファイルを決定する値を使用できます。たとえば、データセットが `customer_id` 別にバケット化されていて、特定の顧客に関するすべてのレコードを検索したいとします。Athena はそれらのレコードを含むバケットを特定し、そのバケット内にあるファイルのみを読み取ります。

高基数 (つまり異なる値が多い) 列が均一に分散していて特定の値を頻繁にクエリする列がある際に、バケット化に適している対象が見つかります。

> **Note**
>
> Athena は、`INSERT INTO` を使用してバケットテーブルに新しいレコードを追加する機能はサポートしていません。

バケット化された列でのフィルタリングでサポートされるデータ型

特定のデータ型を持つバケット列にフィルターを追加できます。Athena は、次のデータ型を持つバケット化された列に対してフィルタリングをサポートしています:

- BOOLEAN
- BYTE
- DATE
- DOUBLE
- FLOAT
- INT
- LONG
- SHORT
- STRING
- VARCHAR

Hive と Spark のサポート

Athena エンジンバージョン 2 は Hive バケットアルゴリズムを使用するバケット化されたデータセットをサポートし、Athena エンジンバージョン 3 は Apache Spark バケットアルゴリズムもサポートしています。Hive バケット化はデフォルトです。データセットが Spark アルゴリズムを使用してバケット化されている場合は、`TBLPROPERTIES` 句を使用して `bucketing_format` プロパティの値を `spark` に設定します。

> **Note**
>
> Athena では、`CREATE TABLE AS SELECT` (CTAS) クエリ内のパーティション数は 100 個に制限されています。同様に、INSERT INTO ステートメントを使用すると、宛先テーブルに最大 100 個のパーティションのみを追加できます。この 100 個の制限は、テーブルのバケット化とパーティション化の両方が行われている場合にのみ適用されます。

この制限を超えると、HIVE_TOO_MANY_OPEN_PARTITIONS: Exceeded limit of 100 open writers for partitions/buckets (HIVE_TOO_MANY_OPEN_PARTITIONS: パーティション/バケットのオープンライターの制限である 100 を超えました) エラーメッセージが表示されることがあります。これらの制限を回避するには、CTAS ステートメントと、それぞれ最大 100 個のパーティションを作成または挿入する一連の INSERT INTO ステートメントを使用できます。詳細については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。

CREATE TABLE のバケット化に関する例

既存のバケット化されたデータセット用のテーブルを作成するには、CLUSTERED BY (*column*) 句の後に INTO *N* BUCKETS 句を続けて使用します。INTO *N* BUCKETS 句は、データをバケット化する先のバケットの数を指定します。

次の CREATE TABLE の例では、sales データセットは customer_id 別に、Spark アルゴリズムを使用して 8 つのバケットにバケット化されています。この CREATE TABLE ステートメントでは、CLUSTERED BY 句と TBLPROPERTIES 句を使用してプロパティをそれぞれ設定します。

```
CREATE EXTERNAL TABLE sales (...)
...
CLUSTERED BY (`customer_id`) INTO 8 BUCKETS
...
TBLPROPERTIES (
  'bucketing_format' = 'spark'
)
```

CREATE TABLE AS (CTAS) のバケット化に関する例

CREATE TABLE AS のバケット化を指定するには、bucketed_by パラメータと bucket_count パラメータを次の例のように使用します。

```
CREATE TABLE sales
WITH (
  ...
  bucketed_by = ARRAY['customer_id'],
  bucket_count = 8
)
AS SELECT ...
```

クエリのバケット化に関する例

次のクエリ例では、特定の顧客が 1 週間を通じて購入した製品の名前を検索します。

```
SELECT DISTINCT product_name
FROM sales
WHERE sales_date BETWEEN '2023-02-27' AND '2023-03-05'
AND customer_id = 'c123'
```

このテーブルを sales_date 別にパーティション化して customer_id 別にバケット化すると、Athena は顧客レコードが入っているバケットを計算できます。Athena が読み込めるのは 1 パーティションにつき 1 ファイルまでです。

追加リソース

バケット化されたテーブルとパーティション化されたテーブルの両方を作成する CREATE TABLE AS の例については、「例: バケット化およびパーティションされたテーブルを作成する」を参照してください。

CTAS クエリの例

以下の例を使用して CTAS クエリを作成します。CTAS 構文の詳細については、「CREATE TABLE AS」を参照してください。

このセクションの内容:

- Example: Duplicating a table by selecting all columns
- Example: Selecting specific columns from one or more tables
- Example: Creating an empty copy of an existing table
- Example: Specifying data storage and compression formats
- Example: Writing query results to a different format
- Example: Creating unpartitioned tables
- Example: Creating partitioned tables
- Example: Creating bucketed and partitioned tables
- Example: Creating an Iceberg table with Parquet data
- Example: Creating an Iceberg table with Avro data

Example 例: すべての列を選択してテーブルを複製する

次の例では、テーブルからすべての列をコピーしてテーブルを作成します。

```
CREATE TABLE new_table AS
SELECT *
FROM old_table;
```

同じ例の次のバリエーションで、SELECT ステートメントには WHERE 句も含まれています。この場合、クエリはテーブルから、WHERE 句を満たす行のみを選択します。

```
CREATE TABLE new_table AS
SELECT *
FROM old_table
WHERE condition;
```

Example 例: 1 つ、または複数のテーブルから特定の列を選択する

次の例では、別のテーブルからの列のセットで実行される新しいクエリが作成されます。

```
CREATE TABLE new_table AS
SELECT column_1, column_2, ... column_n
FROM old_table;
```

同じ例のこのバリエーションで、複数のテーブルの特定の列から新しいテーブルを作成します。

```
CREATE TABLE new_table AS
SELECT column_1, column_2, ... column_n
FROM old_table_1, old_table_2, ... old_table_n;
```

Example 例: 既存のテーブルの空のコピーを作成する

以下の例では、WITH NO DATA を使用して、元のテーブルと同じスキーマである、空の新しいテーブルを作成します。

```
CREATE TABLE new_table
AS SELECT *
FROM old_table
WITH NO DATA;
```

Example 例: データストレージと圧縮形式を指定する

CTAS を使用すると、ソーステーブルをあるストレージ形式で使用して、別のテーブルを異なるストレージ形式で作成することができます。

`format` プロパティを使用して、新しいテーブルのストレージ形式として ORC、PARQUET、AVRO、JSON、または TEXTFILE を指定します。

PARQUET、ORC、TEXTFILE、および JSON のストレージ形式については、`write_compression` プロパティを使用して、新しいテーブルのデータに対する圧縮形式を指定します。各ファイル形式がサポートする圧縮形式については、「Athena での圧縮のサポート」を参照してください。

次の例では、テーブル new_table 内のデータは　Snappy 圧縮を使用した Parquet 形式で保存されることを指定しています。Parquet のデフォルトの圧縮は GZIP です。

```
CREATE TABLE new_table
WITH (
      format = 'Parquet',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table;
```

次の例では、テーブル new_table 内のデータは、Snappy 圧縮を使用した ORC 形式で保存されることを指定しています。ORC のデフォルトの圧縮は ZLIB です。

```
CREATE TABLE new_table
WITH (format = 'ORC',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table ;
```

次の例では、テーブル new_table 内のデータは、Snappy 圧縮を使用したテキストファイル形式で保存されることを指定しています。テキストファイルと JSON 形式のデフォルトの圧縮は GZIP です。

```
CREATE TABLE new_table
WITH (format = 'TEXTFILE',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table ;
```

Example 例: クエリ結果を異なる形式で書き込む

以下の CTAS クエリは、CSV 形式、または別の形式で保存されている可能性がある old_table からすべてのレコードを選択し、ORC 形式で Amazon S3 に保存された基盤となるデータを使って新しいテーブルを作成します。

```
CREATE TABLE my_orc_ctas_table
WITH (
      external_location = 's3://DOC-EXAMPLE-BUCKET/my_orc_stas_table/',
      format = 'ORC')
AS SELECT *
FROM old_table;
```

Example 例: パーティションされていないテーブルを作成する

次の例では、パーティションされていないテーブルを作成します。このテーブル データは異なる形式で格納されます。これらの例の一部では、外部の場所を指定します。

次の例では、テキストファイルとして結果を保存する CTAS クエリが作成されます。

```
CREATE TABLE ctas_csv_unpartitioned
WITH (
     format = 'TEXTFILE',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_csv_unpartitioned/')
AS SELECT key1, name1, address1, comment1
FROM table1;
```

次の例では、結果は Parquet で保存され、デフォルトの結果の場所が使用されます。

```
CREATE TABLE ctas_parquet_unpartitioned
WITH (format = 'PARQUET')
AS SELECT key1, name1, comment1
FROM table1;
```

次のクエリでは、テーブルは JSON で格納され、特定の列が元のテーブルの結果から選択されます。

```
CREATE TABLE ctas_json_unpartitioned
WITH (
     format = 'JSON',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_json_unpartitioned/')
```

```
AS SELECT key1, name1, address1, comment1
FROM table1;
```

次の例では、形式は ORC です。

```
CREATE TABLE ctas_orc_unpartitioned
WITH (
     format = 'ORC')
AS SELECT key1, name1, comment1
FROM table1;
```

次の例では、形式は Avro です。

```
CREATE TABLE ctas_avro_unpartitioned
WITH (
     format = 'AVRO',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_avro_unpartitioned/')
AS SELECT key1, name1, comment1
FROM table1;
```

Example 例: パーティションされたテーブルを作成する

次の例では、WITH 句で partitioned_by と他のプロパティを使用した、異なるストレージ形式のパーティション テーブルに対する CREATE TABLE AS SELECT クエリを示します。構文については、「CTAS テーブルのプロパティ」を参照してください。パーティションの列の選択の詳細については、「Athena におけるパーティション化とバケット化」を参照してください。

> **Note**
>
> SELECT ステートメントの列リストの末尾にパーティション列をリストします。複数の列でパーティションでき、最大 100 個の一意のパーティションとバケットの組み合わせを持つことができます。たとえば、バケットが指定されていない場合、100 個のパーティションを持つことができます。

```
CREATE TABLE ctas_csv_partitioned
WITH (
     format = 'TEXTFILE',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_csv_partitioned/',
     partitioned_by = ARRAY['key1'])
```

```
AS SELECT name1, address1, comment1, key1
FROM tables1;
```

```
CREATE TABLE ctas_json_partitioned
WITH (
     format = 'JSON',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_json_partitioned/',
     partitioned_by = ARRAY['key1'])
AS select name1, address1, comment1, key1
FROM table1;
```

Example 例: バケット化およびパーティションされたテーブルを作成する

以下の例は、Amazon S3 へのクエリ結果の保存にパーティションとバケット化の両方を使用する CREATE TABLE AS SELECT クエリを示しています。テーブルの結果は、異なる列でパーティションおよびバケット化されます。Athena は、最大 100 個の一意のバケットとパーティションの組み合わせをサポートします。たとえば、5 個のバケットを持つテーブルを作成する場合、それぞれ 5 個のバケットを持つ 20 個のパーティションがサポートされます。構文については、「CTAS テーブルのプロパティ」を参照してください。

バケット化の列の選択の詳細については、「Athena におけるパーティション化とバケット化」を参照してください。

```
CREATE TABLE ctas_avro_bucketed
WITH (
      format = 'AVRO',
      external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_avro_bucketed/',
      partitioned_by = ARRAY['nationkey'],
      bucketed_by = ARRAY['mktsegment'],
      bucket_count = 3)
AS SELECT key1, name1, address1, phone1, acctbal, mktsegment, comment1, nationkey
FROM table1;
```

Example 例: Parquet データを使った Iceberg テーブルの作成

次の例では、Parquet データファイルを持つ Iceberg テーブルを作成します。ファイルは、table1 の dt 列目を使用して月ごとに分割されます。この例では、テーブルの保存プロパティを更新して、テーブル内のすべてのブランチにデフォルトで 10 個のスナップショットを保持するようにしました。過去 7 日以内のスナップショットも保持されます。Athena の Iceberg テーブル プロパティの詳細については、「テーブルプロパティ」を参照してください。

```
CREATE TABLE ctas_iceberg_parquet
WITH (table_type = 'ICEBERG',
      format = 'PARQUET',
      location = 's3://DOC-EXAMPLE-BUCKET/ctas_iceberg_parquet/',
      is_external = false,
      partitioning = ARRAY['month(dt)'],
      vacuum_min_snapshots_to_keep = 10,
      vacuum_max_snapshot_age_seconds = 604800
   )
AS SELECT key1, name1, dt FROM table1;
```

Example 例: Avro データを含む Iceberg テーブルの作成

次の例では、key1 でパーティションされた Avro データファイルを含む Iceberg テーブルを作成しています。

```
CREATE TABLE ctas_iceberg_avro
WITH ( format = 'AVRO',
       location = 's3://DOC-EXAMPLE-BUCKET/ctas_iceberg_avro/',
       is_external = false,
       table_type = 'ICEBERG',
       partitioning = ARRAY['key1'])
AS SELECT key1, name1, date FROM table1;
```

ETL およびデータ分析での CTAS および INSERT INTO の使用

Athena で Create Table as Select (CTAS) と INSERT INTO ステートメントを使用して、データ処理のための Amazon S3 へのデータの抽出、変換、ロード (ETL) を行うことができます。このトピックでは、これらのステートメントを使用することで、データセットをパーティション分割して列指向データ形式に変換し、データ分析用に最適化する方法について説明します。

CTAS ステートメントでは、標準の SELECT クエリを使用して新しいテーブルを作成します。CTAS ステートメントを使用して、分析用のデータのサブセットを作成できます。1 つの CTAS ステートメントで、データのパーティション分割、圧縮の指定、Apache Parquet や Apache ORC などの列指向形式へのデータ変換を行うことができます。CTAS クエリを実行すると、それによって作成されるテーブルとパーティションが自動的に AWS Glue Data Catalog に追加されます。これにより、作成した新しいテーブルとパーティションは、その後のクエリですぐに使用できます。

INSERT INTO ステートメントは、ソーステーブルで実行される SELECT クエリステートメントに基づいて、ターゲットテーブルに新しい行を挿入します。INSERT INTO ステートメントを使用する

と、CTAS がサポートするすべての変換を使用して、CSV 形式のソーステーブルデータをターゲットテーブルデータに変換およびロードできます。

概要

Athena では、CTAS ステートメントを使用してデータの初期バッチ変換を実行します。次に、複数の INSERT INTO ステートメントを使用して、CTAS ステートメントによって作成されたテーブルに対して増分更新を行います。

ステップ

- ステップ 1: 元のデータセットに基づいてテーブルを作成する
- ステップ 2: CTAS を使用してデータをパーティション分割、変換、および圧縮する
- ステップ 3: INSERT INTO を使用してデータを追加する
- ステップ 4: パフォーマンスとコストの差を測定する

ステップ 1: 元のデータセットに基づいてテーブルを作成する

このトピックの例では、一般公開されている NOAA Global Historical Climatology Network Daily (GHCN-D) データセットを使用した、Amazon S3 の読み取り可能なサブセットを使用します。Amazon S3 上のデータには、以下の特徴があります。

```
Location: s3://aws-bigdata-blog/artifacts/athena-ctas-insert-into-blog/
Total objects: 41727
Size of CSV dataset: 11.3 GB
Region: us-east-1
```

元のデータは、パーティションなしで Amazon S3 に保存されます。データは CSV 形式で、以下のようなファイル内にあります。

```
2019-10-31 13:06:57  413.1 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0000
2019-10-31 13:06:57  412.0 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0001
2019-10-31 13:06:57   34.4 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0002
2019-10-31 13:06:57  412.2 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0100
2019-10-31 13:06:57  412.7 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0101
```

このサンプルのファイルサイズは比較的小さくなっています。それらを大きなファイルにマージすることによってファイルの合計数を減らすことができるため、より優れたクエリパフォーマンスが実現

されます。CTAS ステートメントと INSERT INTO ステートメントを使用して、クエリのパフォーマンスを向上させることができます。

サンプルデータセットに基づいてデータベースとテーブルを作成する

1. Athena コンソールで、AWS リージョン リージョンに [US East (N. Virginia)] (米国東部 (バージニア北部)) を選択します。このチュートリアルのクエリは、すべて `us-east-1` で実行するようにしてください。
2. Athena のクエリエディタで、CREATE DATABASE コマンドを実行してデータベースを作成します。

```
CREATE DATABASE blogdb
```

3. 次のステートメントを実行して、テーブルを作成します。

```
CREATE EXTERNAL TABLE `blogdb`.`original_csv` (
  `id` string,
  `date` string,
  `element` string,
  `datavalue` bigint,
  `mflag` string,
  `qflag` string,
  `sflag` string,
  `obstime` bigint)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://aws-bigdata-blog/artifacts/athena-ctas-insert-into-blog/'
```

ステップ 2: CTAS を使用してデータをパーティション分割、変換、および圧縮する

テーブルを作成したら、1 つの CTAS ステートメントを使用して、データを Snappy 圧縮で Parquet 形式に変換し、データを年ごとにパーティション化できます。

ステップ 1 で作成したテーブルには、日付が YYYYMMDD としてフォーマットされた `date` フィールド (例: `20100104`) があります。新しいテーブルが `year` でパーティション化されるため、次の手順

のサンプルステートメントでは、Presto 関数 `substr("date",1,4)` を使用して `date` フィールドから値 `year` を抽出します。

Snappy 圧縮を使用してデータを Parquet 形式に変換し、年ごとにパーティション分割する

- *`your-bucket`* をお使いの Amazon S3 バケットの場所に置き換えてから、以下の CTAS ステートメントを実行します。

```
CREATE table new_parquet
WITH (format='PARQUET',
parquet_compression='SNAPPY',
partitioned_by=array['year'],
external_location = 's3://DOC-EXAMPLE-BUCKET/optimized-data/')
AS
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) >= 2015
        AND cast(substr("date",1,4) AS bigint) <= 2019
```

> **Note**
>
> この例では、作成するテーブルには 2015 年から 2019 年のデータのみが含まれます。ステップ 3 では、INSERT INTO コマンドを使用して、このテーブルに新しいデータを追加します。

クエリが完了したら、以下の手順を使用して、CTAS ステートメントで指定した Amazon S3 の場所にある出力を検証します。

CTAS ステートメントによって作成されたパーティションと parquet ファイルを表示する

1. 作成されたパーティションを表示するには、次の AWS CLI コマンドを実行します。必ず、最後にスラッシュ (/) を含めてください。

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/
```

出力はパーティションを示しています。

```
      PRE year=2015/
      PRE year=2016/
      PRE year=2017/
      PRE year=2018/
      PRE year=2019/
```

2. Parquet ファイルを表示するには、以下のコマンドを実行します。出力を最初の 5 つの結果に制限する | head-5 オプションは、Windows では使用できません。

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/ --recursive --human-readable |
 head -5
```

出力は以下のようになります。

```
2019-10-31 14:51:05    7.3 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_1be48df2-3154-438b-b61d-8fb23809679d
2019-10-31 14:51:05    7.0 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_2a57f4e2-ffa0-4be3-9c3f-28b16d86ed5a
2019-10-31 14:51:05    9.9 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_34381db1-00ca-4092-bd65-ab04e06dc799
2019-10-31 14:51:05    7.5 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_354a2bc1-345f-4996-9073-096cb863308d
2019-10-31 14:51:05    6.9 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_42da4cfd-6e21-40a1-8152-0b902da385a1
```

ステップ 3: INSERT INTO を使用してデータを追加する

ステップ 2 では、CTAS を使用して 2015 年から 2019 年までのデータのパーティションを持つテーブルを作成しました。しかし、元のデータセットには、2010 年から 2014 年までのデータも含まれています。次に、INSERT INTO ステートメントを使用してそのデータを追加します。

1 つ以上の INSERT INTO ステートメントを使用してテーブルにデータを追加する

1. WHERE 句で 2015 年より前の年を指定して、以下の INSERT INTO コマンドを実行します。

```
INSERT INTO new_parquet
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) < 2015
```

2. 次の構文を使用して aws s3 ls コマンドを再度実行します。

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/
```

出力には、新しいパーティションが表示されます。

```
      PRE year=2010/
      PRE year=2011/
      PRE year=2012/
      PRE year=2013/
      PRE year=2014/
      PRE year=2015/
      PRE year=2016/
      PRE year=2017/
      PRE year=2018/
      PRE year=2019/
```

3. Parquet 形式の圧縮と列指向ストレージを使用して取得したデータセットのサイズが小さくなっていることを確認するには、以下のコマンドを実行します。

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/ --recursive --human-readable --
summarize
```

以下の結果は、Snappy 圧縮を使用した parquet 後のデータセットのサイズが 1.2 GB であることを示しています。

```
...
```

```
2020-01-22 18:12:02 2.8 MiB optimized-data/
year=2019/20200122_181132_00003_nja5r_f0182e6c-38f4-4245-afa2-9f5bfa8d6d8f
2020-01-22 18:11:59 3.7 MiB optimized-data/
year=2019/20200122_181132_00003_nja5r_fd9906b7-06cf-4055-a05b-f050e139946e
Total Objects: 300
     Total Size: 1.2 GiB
```

4. さらに多くの CSV データが元のテーブルに追加されている場合は、INSERT INTO ステートメントを使用して、そのデータを parquet テーブルに追加できます。たとえば、2020 年の新しいデータがある場合は、以下の INSERT INTO ステートメントを実行できます。このステートメントは、データおよび関連するパーティションを new_parquet テーブルに追加します。

```
INSERT INTO new_parquet
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) = 2020
```

> **Note**
>
> この INSERT INTO ステートメントは、最大 100 個のパーティションの宛先テーブルへの書き込みをサポートします。ただし、100 個を超えるパーティションを追加するには、複数の INSERT INTO ステートメントを実行できます。詳細については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。

ステップ 4: パフォーマンスとコストの差を測定する

データを変換した後、新しいテーブルと古いテーブルで同じクエリを実行し、結果を比較することで、パフォーマンスの向上とコスト削減を測定できます。

> **Note**
>
> Athena の クエリごとのコストについては、Amazon Athena の料金表を参照してください。

パフォーマンスの向上とコストの差を測定する

1. 元のテーブルで以下のクエリを実行します。このクエリにより、年のすべての値の個別の ID の数が検出されます。

```
SELECT substr("date",1,4) as year,
       COUNT(DISTINCT id)
FROM original_csv
GROUP BY 1 ORDER BY 1 DESC
```

2. クエリが実行された時刻とスキャンされたデータの量を確認します。
3. 新しいテーブルで同じクエリを実行し、クエリの実行時間とスキャンされたデータの量をメモします。

```
SELECT year,
  COUNT(DISTINCT id)
FROM new_parquet
GROUP BY 1 ORDER BY 1 DESC
```

4. 結果を比較し、パフォーマンスとコストの差を計算します。以下のサンプル結果は、新しいテーブルのテストクエリが古いテーブルのクエリよりも高速で安価であることを示しています。

表	実行時間	スキャンされたデータ
元	16.88 秒	11.35 GB
新規	3.79 秒	428.05 MB

5. 元のテーブルで次のサンプルクエリを実行します。このクエリでは、2018 年の地球の平均最高温度 (摂氏)、平均最低気温 (摂氏)、平均降水量 (mm) を計算します。

```
SELECT element, round(avg(CAST(datavalue AS real)/10),2) AS value
FROM original_csv
WHERE element IN ('TMIN', 'TMAX', 'PRCP') AND substr("date",1,4) = '2018'
GROUP BY 1
```

6. クエリが実行された時刻とスキャンされたデータの量を確認します。
7. 新しいテーブルで同じクエリを実行し、クエリの実行時間とスキャンされたデータの量をメモします。

```
SELECT element, round(avg(CAST(datavalue AS real)/10),2) AS value
FROM new_parquet
WHERE element IN ('TMIN', 'TMAX', 'PRCP') and year = '2018'
GROUP BY 1
```

8. 結果を比較し、パフォーマンスとコストの差を計算します。以下のサンプル結果は、新しいテーブルのテストクエリが古いテーブルのクエリよりも高速で安価であることを示しています。

表	実行時間	スキャンされたデータ
元	18.65 秒	11.35 GB
新規	1.92 秒	68 MB

[概要]

このトピックでは、Athena で CTAS と INSERT INTO ステートメントを使用して ETL オペレーションを実行する方法を説明しました。最初の変換セットは、データを Snappy 圧縮で Parquet 形式に変換する CTAS ステートメントを使用して実行しました。CTAS ステートメントは、データセットのパーティション分割されていないものからパーティション分割されたものへの変換も実行しました。これにより、サイズが小さくなり、クエリの実行コストが削減されました。新しいデータが使用可能になったら、INSERT INTO ステートメントを使用して、CTAS ステートメントで作成したテーブルにデータを変換およびロードできます。

CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する

Athena では、`CREATE TABLE AS SELECT` (CTAS) クエリごとのパーティション数は 100 個に制限されています。同様に、INSERT INTO ステートメントを使用すると、宛先テーブルに最大 100 個のパーティションを追加できます。

この制限を超えると、HIVE_TOO_MANY_OPEN_PARTITIONS: Exceeded limit of 100 open writers for partitions/buckets (HIVE_TOO_MANY_OPEN_PARTITIONS: パーティション/バケットのオープンライターの制限である 100 を超えました) エラーメッセージが表示されることがあります。これらの

制限を回避するには、CTAS ステートメントと、それぞれ最大 100 個のパーティションを作成または挿入する一連の INSERT INTO ステートメントを使用できます。

このトピックの例では、Amazon S3 バケットロケーション s3://DOC-EXAMPLE-BUCKET/ にデータが格納されている、tpch100 という名前のデータベースを使用します。

CTAS および INSERT INTO を使用して 100 個を超えるパーティションのテーブルを作成するには

1. CREATE EXTERNAL TABLE ステートメントを使用して、目的のフィールドでパーティション化されたテーブルを作成します。

   次の例のステートメントは、列 l_shipdate でデータを分割します。テーブルには 2525 個のパーティションがあります。

```
CREATE EXTERNAL TABLE `tpch100.lineitem_parq_partitioned`(
  `l_orderkey` int,
  `l_partkey` int,
  `l_suppkey` int,
  `l_linenumber` int,
  `l_quantity` double,
  `l_extendedprice` double,
  `l_discount` double,
  `l_tax` double,
  `l_returnflag` string,
  `l_linestatus` string,
  `l_commitdate` string,
  `l_receiptdate` string,
  `l_shipinstruct` string,
  `l_comment` string)
PARTITIONED BY (
  `l_shipdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe' STORED AS
 INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat' OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat' LOCATION
 's3://DOC-EXAMPLE-BUCKET/lineitem/'
```

2. 次のような SHOW PARTITIONS *<table_name>* コマンドを実行して、パーティションをリストします。

```
SHOW PARTITIONS lineitem_parq_partitioned
```

次に、結果の一部を示します。

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06

...

l_shipdate=1998-11-24
l_shipdate=1998-11-25
l_shipdate=1998-11-26
l_shipdate=1998-11-27
l_shipdate=1998-11-28
l_shipdate=1998-11-29
l_shipdate=1998-11-30
l_shipdate=1998-12-01
*/
```

3. CTAS クエリを実行して、パーティション分割されたテーブルを作成します。

   次の例では、my_lineitem_parq_partitioned というテーブルを作成し、WHERE 句を使用して、DATE を 1992-02-01 より前の日付に制限します。サンプルデータセットは 1992 年 1 月から始まるため、1992 年 1 月のパーティションのみが作成されます。

```
CREATE table my_lineitem_parq_partitioned
WITH (partitioned_by = ARRAY['l_shipdate']) AS
SELECT l_orderkey,
         l_partkey,
         l_suppkey,
         l_linenumber,
         l_quantity,
         l_extendedprice,
         l_discount,
         l_tax,
         l_returnflag,
         l_linestatus,
         l_commitdate,
         l_receiptdate,
         l_shipinstruct,
```

```
        l_comment,
        l_shipdate
FROM tpch100.lineitem_parq_partitioned
WHERE cast(l_shipdate as timestamp) < DATE ('1992-02-01');
```

4. SHOW PARTITIONS コマンドを実行して、必要なパーティションがテーブルに含まれていることを確認します。

```
SHOW PARTITIONS my_lineitem_parq_partitioned;
```

この例のパーティションは 1992 年 1 月のものです。

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06
l_shipdate=1992-01-07
l_shipdate=1992-01-08
l_shipdate=1992-01-09
l_shipdate=1992-01-10
l_shipdate=1992-01-11
l_shipdate=1992-01-12
l_shipdate=1992-01-13
l_shipdate=1992-01-14
l_shipdate=1992-01-15
l_shipdate=1992-01-16
l_shipdate=1992-01-17
l_shipdate=1992-01-18
l_shipdate=1992-01-19
l_shipdate=1992-01-20
l_shipdate=1992-01-21
l_shipdate=1992-01-22
l_shipdate=1992-01-23
l_shipdate=1992-01-24
l_shipdate=1992-01-25
l_shipdate=1992-01-26
l_shipdate=1992-01-27
l_shipdate=1992-01-28
l_shipdate=1992-01-29
l_shipdate=1992-01-30
l_shipdate=1992-01-31
```

```
*/
```

5. INSERT INTO ステートメントを使用して、テーブルにパーティションを追加します。

   次の例では、1992 年 2 月の日付のパーティションを追加します。

```
INSERT INTO my_lineitem_parq_partitioned
SELECT l_orderkey,
         l_partkey,
         l_suppkey,
         l_linenumber,
         l_quantity,
         l_extendedprice,
         l_discount,
         l_tax,
         l_returnflag,
         l_linestatus,
         l_commitdate,
         l_receiptdate,
         l_shipinstruct,
         l_comment,
         l_shipdate
FROM tpch100.lineitem_parq_partitioned
WHERE cast(l_shipdate as timestamp) >= DATE ('1992-02-01')
AND cast(l_shipdate as timestamp) < DATE ('1992-03-01');
```

6. SHOW PARTITIONS をもう一度実行します。

```
SHOW PARTITIONS my_lineitem_parq_partitioned;
```

   サンプルテーブルには、1992 年 1 月と 2 月のパーティションがあります。

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06

...

l_shipdate=1992-02-20
l_shipdate=1992-02-21
```

```
l_shipdate=1992-02-22
l_shipdate=1992-02-23
l_shipdate=1992-02-24
l_shipdate=1992-02-25
l_shipdate=1992-02-26
l_shipdate=1992-02-27
l_shipdate=1992-02-28
l_shipdate=1992-02-29
*/
```

7. 各々が読み取りと追加を 100 パーティション以上実行しない INSERT INTO ステートメントを引き続き使用します。必要なパーティション数に達するまで続行します。

> **Important**
>
> WHERE 条件を設定するときは、クエリが重複しないようにしてください。そうしなければ、一部のパーティションに重複するデータが発生する可能性があります。

SerDe リファレンス

Athena は、CSV、JSON、Parquet、および ORC などのさまざまなデータ形式からのデータを解析するために、複数の SerDe ライブラリをサポートしています。Athena はカスタム SerDe をサポートしません。

トピック

- SerDe の使用
- サポートされる SerDes とデータ形式

SerDe の使用

SerDe (シリアライザー/デシリアライザー) は、Athena がさまざまな形式のデータとやり取りする方法です。

SerDe は、ユーザーが指定するものであり、テーブルスキーマを定義する DDL ではありません。言い換えると、SerDe は、テーブルの作成時に Athena で指定する DDL 設定を上書きできます。

クエリで SerDe を使用するには

Athena でテーブルを作成するときに SerDe を使用するには、以下の方法のいずれかを使用します。

- 以下の例にあるように、ROW FORMAT DELIMITED を指定してから、DDL ステートメントを使用してフィールド区切り文字を指定します。ROW FORMAT DELIMITED を指定すると、Athena がデフォルトで LazySimpleSerDe を使用します。

```
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
ESCAPED BY '\\'
COLLECTION ITEMS TERMINATED BY '|'
MAP KEYS TERMINATED BY ':'
```

  ROW FORMAT DELIMITED の例については、次のトピックを参照してください。

  CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe

  Amazon CloudFront ログのクエリ

  Amazon EMR ログのクエリ

  Amazon VPC フローログのクエリ

  ETL およびデータ分析での CTAS および INSERT INTO の使用

- ROW FORMAT SERDE は、テーブルへのデータの読み書きを行うときに Athena が使用する必要がある SerDe のタイプを明示的に指定します。以下の例は、LazySimpleSerDe を指定しています。区切り文字を指定するには、WITH SERDEPROPERTIES を使用します。WITH SERDEPROPERTIES によって指定されたプロパティは、ROW FORMAT DELIMITED 例の個別のステートメント (FIELDS TERMINATED BY など) に対応します。

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'serialization.format' = ',',
'field.delim' = ',',
'collection.delim' = '|',
'mapkey.delim' = ':',
'escape.delim' = '\\'
)
```

  ROW FORMAT SERDE の例については、次のトピックを参照してください。

  Avro SerDe

Grok SerDe

JSON SerDe ライブラリ

CSV を処理するための OpenCSVSerDe

Regex SerDe

サポートされる SerDes とデータ形式

Athena がテーブルの作成とデータのクエリでサポートしているのは、CSV、TSV、カスタム区切り、JSON の各形式のデータ、Hadoop 関連形式 (ORC、Apache Avro、Parquet) のデータ、および Logstash からのログ、AWS CloudTrail ログ、Apache WebServer ログです。

> **Note**
>
> このセクションにリストされている形式は、Athena がデータの読み込みに使用します。Athena が CTAS クエリの実行時にデータの書き込みに使用する形式については、「クエリ結果からのテーブルの作成 (CTAS)」を参照してください。

Athena でこれらの形式のテーブルを作成し、クエリを実行するには、シリアライザー/デシリアライザークラス (SerDe) を指定して、使用する形式とデータの解析方法を Athena に指示します。

この表には、Athena でサポートされているデータ形式と、それらに対応する SerDe ライブラリがリストされています。

SerDe は、Athena で使用されているデータカタログにデータの処理方法を指示するカスタムライブラリです。SerDe タイプは、Athena で `CREATE TABLE` ステートメントの `ROW FORMAT` 部分に SerDe タイプを明示的にリストすることによって指定します。Athena は、特定のタイプのデータ形式にデフォルトでいくつかの SerDe タイプを使用するため、SerDe 名を省略できる場合もあります。

サポートされるデータ形式と SerDes

データ形式	説明	Athena でサポートされる SerDe タイプ
Amazon Ion	Amazon Ion は JSON のスーパーセットであるリッチタイプの自己記述データ形式で、Amazon によって開発およびオープンソース化されています。	Amazon Ion Hive SerDe を使用します。
Apache Avro	Hadoop にデータを保存する形式であり、JSON ベースのスキーマをレコード値として使用します。	Avro SerDe を使用します。
Apache Parquet	Hadoop のデータの列指向ストレージ形式。	Parquet SerDe および SNAPPY 圧縮を使用します。
Apache WebServer ログ	Apache WebServer にログを保存する形式。	Grok SerDe または Regex SerDe を使用します。
CloudTrail ログ	CloudTrail にログを保存するための形式。	• Hive JSON SerDe を使用します。詳細については、「AWS CloudTrail ログのクエリ」を参照してください。
CSV (カンマ区切り値)	CSV のデータでは、各行がデータレコードを表し、各レコードはカンマで区切られた 1 つ以上のフィールドで構成されます。	• 引用符で囲まれた値がデータに含まれていない場合や、`java.sql.Timestamp` 形式を使用している場合は、CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe を使用します。

データ形式	説明	Athena でサポートされる SerDe タイプ
		• 引用符で囲まれた値がデータに含まれている場合や、TIMESTAMP に UNIX の数値形式 (例えば、1564610311) を使用している場合は、CSV を処理するための OpenCSVSerDe を使用します。
カスタム区切り	この形式のデータでは、各行がデータレコードを表し、レコード間は 1 文字のカスタム区切り文字で区切られます。	CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe を使用し、1 文字のカスタム区切り文字を指定します。
JSON (JavaScript Object Notation)	JSON データでは、各行がデータレコードを表します。各レコードは属性と値のペアと配列で構成され、それぞれがカンマで区切られます。	• Hive JSON SerDe を使用します。 • OpenX JSON SerDe を使用します。
Logstash ログ	Logstash にログを保存する形式。	Grok SerDe を使用します。
ORC (Optimized Row Columnar)	Hive データの最適化された列指向ストレージの形式。	ORC SerDe および ZLIB 圧縮を使用します。
TSV (タブ区切り値)	TSV のデータでは、各行がデータレコードを表し、各レコードはタブで区切られた 1 つ以上のフィールドで構成されます。	CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe を使用し、区切り文字を FIELDS TERMINATED BY '\t' に指定します。

トピック

- Amazon Ion Hive SerDe
- Avro SerDe
- Grok SerDe
- JSON SerDe ライブラリ
- CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe
- CSV を処理するための OpenCSVSerDe
- ORC SerDe
- Parquet SerDe
- Regex SerDe

Amazon Ion Hive SerDe

Amazon Ion Hive SerDe を使用すると、Amazon Ion 形式で保存されているデータをクエリできます。Amazon Ion は、リッチタイプで自己記述型のオープンソースのデータ形式です。Amazon Ion 形式は、Amazon Quantum Ledger Database (Amazon QLDB) やオープンソースの SQL クエリ言語である PartiQL などのサービスで使用されています。

Amazon Ion には、交換可能なバイナリ形式とテキスト形式があります。この機能は、テキストの使いやすさとバイナリエンコードの効率性を兼ね備えています。

Athena から Amazon Ion データをクエリするには、Amazon Ion Hive SerDe を使用できます。これにより、Amazon Ion データがシリアル化および逆シリアル化されます。逆シリアル化では、Amazon Ion データをクエリしたり、データを読み取って Parquet や ORC など別の形式に書き出したりできます。シリアル化では、`CREATE TABLE AS SELECT` (CTAS) や `INSERT INTO` クエリを使用して既存のテーブルからデータをコピーすることで、Amazon Ion 形式のデータを生成できます。

> **Note**
>
> Amazon Ion は JSON のスーパーセットであるため、Amazon Ion Hive SerDe を使用して Amazon Ion 以外の JSON データセットをクエリできます。他の JSON SerDe ライブラリとは異なり、Amazon Ion SerDe ではデータの各行が 1 行にあるとは想定していません。この機能は、「プリティ印刷」形式の JSON データセットをクエリする場合や、改行文字で行のフィールドを分割する場合に便利です。

Athena で Amazon Ion をクエリする方法の追加情報および例については、「Analyze Amazon Ion datasets using Amazon Athena」を参照してください。

SerDe 名

- com.amazon.ionhiveserde.IonHiveSerDe

考慮事項と制約事項

- 重複フィールド – Amazon Ion 構造体は順序付けされ、重複フィールドに対応していますが、Hive の STRUCT<> と MAP<> はそうではありません。そのため、Amazon Ion 構造体から重複フィールドを逆シリアル化すると、ある 1 つの値が非決定的に選択され、それ以外の値は無視されます。
- 外部シンボルテーブルはサポート対象外 – 現時点で、Athena は外部シンボルテーブルや以下の Amazon Ion Hive SerDe プロパティをサポートしていません。
  - `ion.catalog.class`
  - `ion.catalog.file`
  - `ion.catalog.url`
  - `ion.symbol_table_imports`
- ファイル拡張子 – Amazon Ion は、ファイル拡張子に基づいて、Amazon Ion ファイルの逆シリアル化に使用する圧縮コーデックを決定します。そのため、圧縮ファイルのファイル拡張子は、使用する圧縮アルゴリズムに対応したものである必要があります。例えば、ZSTD を使用する場合、対応するファイルの拡張子は `.zst` である必要があります。
- 同種データ – Amazon Ion では、特定のフィールドの値に使用できるデータ型に制限がありません。例えば、2 つの異なる Amazon Ion ドキュメントに同じ名前のフィールドがある場合、それぞれのフィールドに異なるデータ型を使用できます。ただし、Hive はスキーマを使用するため、1 つの Hive 列に抽出する値はすべて同じデータ型にする必要があります。
- マップキーのタイプの制限 – 別の形式のデータを Amazon Ion にシリアル化する場合は、マップキーのタイプを STRING、VARCHAR、CHAR のいずれかにします。Hive では任意のプリミティブデータ型をマップキーとして使用できますが、Amazon Ion のシンボルは文字列型にする必要があります。
- 共用体型 – Athena は、現時点では Hive 共用体型をサポートしていません。
- 倍精度浮動小数点データ型 — Amazon Ion は現在、`double` データ型をサポートしていません。

トピック

- CREATE TABLE を使用して Amazon Ion テーブルを作成する

- CTAS と INSERT INTO を使用した Amazon Ion テーブルの作成
- Amazon Ion SerDe プロパティの使用
- パスエクストラクタの使用

CREATE TABLE を使用して Amazon Ion テーブルを作成する

Amazon Ion 形式で格納されたデータから Athena でテーブルを作成するには、CREATE TABLE ステートメントで次のいずれかの方法を使用します。

- `STORED AS ION` を指定します。この方法では、Amazon Ion Hive SerDe を明示的に指定する必要はありません。このオプションの方が簡単です。
- `ROW FORMAT SERDE`、`INPUTFORMAT`、および `OUTPUTFORMAT` の各フィールドで Amazon Ion のクラスパスを指定します。

また、`CREATE TABLE AS SELECT` (CTAS) ステートメントを使用して、Athena で Amazon Ion テーブルを作成することもできます。詳細については、CTAS と INSERT INTO を使用した Amazon Ion テーブルの作成 を参照してください。

STORED AS ION を指定する

次の例の `CREATE TABLE` ステートメントでは、`LOCATION` 句の前で `STORED AS ION` を使用し、Amazon Ion 形式でフライトデータに基づいたテーブルを作成しています。`LOCATION` 句では、Ion 形式の入力ファイルが置かれているバケットまたはフォルダを指定します。指定した場所にあるすべてのファイルをスキャンします。

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

Amazon Ion クラスパスを指定する

STORED AS ION 構文を使用する代わりに、次のように ROW FORMAT SERDE、INPUTFORMAT、および OUTPUTFORMAT 句に Ion クラスパスの値を明示的に指定することができます。

パラメータ	Ion クラスパス
ROW FORMAT SERDE	'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS INPUTFORMAT	'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT	'com.amazon.ionhiveserde.formats.IonOutputFormat'

次の DDL クエリでは、この手法を使用して、前の例と同じ外部テーブルを作成しています。

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS INPUTFORMAT
 'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT
 'com.amazon.ionhiveserde.formats.IonOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

Athena の CREATE TABLE ステートメントの SerDe プロパティの詳細については、「Amazon Ion SerDe プロパティの使用」を参照してください。

CTAS と INSERT INTO を使用した Amazon Ion テーブルの作成

Athena では、`CREATE TABLE AS SELECT` (CTAS) と `INSERT INTO` ステートメントを使用して、テーブルのデータを Amazon Ion 形式で新しいテーブルにコピーまたは挿入できます。

CTAS クエリでは、次の例のように、`WITH` 句に `format='ION'` を指定します。

```
CREATE TABLE new_table
WITH (format='ION')
AS SELECT * from existing_table
```

デフォルトでは、Athena は Amazon Ion の結果を Ion バイナリ形式でシリアル化しますが、テキスト形式を使用することもできます。テキスト形式を使用するには、次の例のように、CTAS `WITH` 句に `ion_encoding = 'TEXT'` を指定します。

```
CREATE TABLE new_table
WITH (format='ION', ion_encoding = 'TEXT')
AS SELECT * from existing_table
```

CTAS `WITH` 句の Amazon Ion 固有のプロパティの詳細については、次のセクションを参照してください。

CTAS WITH 句の Amazon Ion プロパティ

CTAS クエリで `WITH` 句を使用すると、Amazon Ion 形式を指定できるほか、必要に応じて Amazon Ion エンコードを指定したり、使用する圧縮アルゴリズムを記述したりできます。

format

CTAS クエリの `WITH` 句では、形式オプションとして `ION` キーワードを指定できます。このようにしてテーブルを作成すると、`IonInputFormat` に指定した形式でデータが読み取られ、`IonOutputFormat` に指定した形式でデータがシリアル化されます。

次の例では、CTAS クエリで Amazon Ion 形式を使用するように指定しています。

```
WITH (format='ION')
```

ion_encoding

オプションです。

デフォルト: BINARY

値: BINARY、TEXT

Amazon Ion バイナリ形式と Amazon Ion テキスト形式のどちらでデータをシリアル化するかを指定します。次の例では、Amazon Ion テキスト形式を指定しています。

```
WITH (format='ION', ion_encoding='TEXT')
```

write_compression

オプションです。

デフォルト: GZIP

値: GZIP、ZSTD、BZIP2、SNAPPY、NONE

出力ファイルの圧縮に使用する圧縮アルゴリズムを指定します。

次の例では、Zstandard 圧縮アルゴリズムを使用して CTAS クエリの出力を Amazon Ion 形式で生成するように指定しています。

```
WITH (format='ION', write_compression = 'ZSTD')
```

Athena での圧縮の使用の詳細については、「Athena での圧縮のサポート」を参照してください。

Athena のその他の CTAS プロパティについては、「CTAS テーブルのプロパティ」を参照してください。

Amazon Ion SerDe プロパティの使用

このトピックでは、Athena の CREATE TABLE ステートメントに使用できる SerDe プロパティについて説明します。Amazon Ion SerDe プロパティの使用方法の詳細と例については、GitHub にある Amazon Ion Hive SerDe ドキュメントの「SerDe properties」(SerDe プロパティ) を参照してください。

Amazon Ion SerDe プロパティの指定

CREATE TABLE ステートメントで Amazon Ion Hive SerDe のプロパティを指定するには、WITH SERDEPROPERTIES 句を使用します。WITH SERDEPROPERTIES が ROW FORMAT SERDE 句のサ

ブフィールドであるため、次の構文に示すように、まず ROW FORMAT SERDE と Amazon Ion Hive SerDe クラスパスを指定する必要があります。

```
...
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'property' = 'value',
 'property' = 'value',
...
)
```

ただし、WITH SERDEPROPERTIES を使用する場合には ROW FORMAT SERDE 句が必須ですが、STORED AS ION か、または長めの INPUTFORMAT と OUTPUTFORMAT Amazon Ion 構文を使用して Amazon Ion 形式を指定できます。

Amazon Ion SerDe プロパティ

Athena の CREATE TABLE ステートメントに使用できる Amazon Ion SerDe プロパティは次のとおりです。

ion.encoding

オプションです。

デフォルト: BINARY

値: BINARY、TEXT

このプロパティは、追加された新しい値が Amazon Ion バイナリと Amazon Ion テキストのどちらの形式としてシリアル化されるかを宣言します。

次の SerDe プロパティの例では、Amazon Ion テキスト形式を指定しています。

```
'ion.encoding' = 'TEXT'
```

ion.fail_on_overflow

オプションです。

デフォルト: true

値: true、false

Amazon Ion では任意の大きな数値型が許可されますが、Hive では許可されません。デフォルトでは、Amazon Ion の値が Hive 列に収まらないと SerDe が失敗しますが、`fail_on_overflow` 設定オプションを使用すると、失敗ではなく値をオーバーフローさせることができます。

このプロパティは、テーブルレベルまたは列レベルで設定できます。テーブルレベルで指定するには、以下の例のように `ion.fail_on_overflow` を指定します。これにより、すべての列にデフォルトの動作が設定されます。

```
'ion.fail_on_overflow' = 'true'
```

特定の列を制御するには、次の例のように、`ion` と `fail_on_overflow` の間に列名をピリオドで区切って指定します。

```
'ion.<column>.fail_on_overflow' = 'false'
```

ion.path_extractor.case_sensitive

オプションです。

デフォルト: `false`

値: `true`、`false`

Amazon Ion フィールド名の大文字と小文字を区別するかどうかを指定します。`false` の場合、SerDe は Amazon Ion フィールド名を解析するときに大文字と小文字を区別しません。

例えば、Hive テーブルスキーマに `alias` フィールドを小文字で定義し、Amazon Ion ドキュメントに `alias` フィールドと `ALIAS` フィールドの両方を設定できます。以下に例を示します。

```
-- Hive Table Schema
alias: STRING

-- Amazon Ion Document
{ 'ALIAS': 'value1'}
{ 'alias': 'value2'}
```

次の例は、大文字と小文字の区別が `false` に設定されている場合に抽出されるテーブルと、SerDe プロパティを示しています。

```
-- Serde properties
'ion.alias.path_extractor' = '(alias)'
```

```
'ion.path_extractor.case_sensitive' = 'false'

--Extracted Table
| alias    |
|----------|
| "value1" |
| "value2" |
```

次の例は、大文字と小文字の区別が `true` に設定されている場合に抽出されるテーブルと、SerDe プロパティを示しています。

```
-- Serde properties
'ion.alias.path_extractor' = '(alias)'
'ion.path_extractor.case_sensitive' = 'true'

--Extracted Table
| alias    |
|----------|
| "value2" |
```

2 番目のケースでは、大文字と小文字の区別が `true` に設定され、パスエクストラクタが `alias` に指定されている場合、`ALIAS` フィールドの `value1` は無視されます。

ion.*<column>*.path_extractor

オプションです。

デフォルト: NA

値: 検索パスを含む文字列

指定された列の指定された検索パスが含まれるパスエクストラクタを作成します。パスエクストラクタは、Amazon Ion フィールドを Hive 列にマッピングします。パスエクストラクタを指定しないと、Athena は列名に基づいて実行時に動的にパスエクストラクタを作成します。

次の例のパスエクストラクタは、`example_ion_field` を `example_hive_column` にマッピングしています。

```
'ion.example_hive_column.path_extractor' = '(example_ion_field)'
```

パスエクストラクタと検索パスの詳細については、「パスエクストラクタの使用」を参照してください。

ion.timestamp.serialization_offset

オプションです。

デフォルト: 'Z'

値: OFFSET。OFFSET は *<signal>*hh:mm として表される。値の例: 01:00、+01:00、-09:30、Z (UTC、00:00 と同じ)

Apache Hive のタイムスタンプは、タイムゾーンが組み込まれておらず、UNIX エポックからのオフセットとして保存されます。一方、Amazon Ion のタイムスタンプにはオフセットがあります。このプロパティを使用して、Amazon Ion にシリアル化するときのオフセットを指定します。

次の例では、1 時間のオフセットを追加しています。

```
'ion.timestamp.serialization_offset' = '+01:00'
```

ion.serialize_null

オプションです。

デフォルト: OMIT

値: OMIT、UNTYPED、TYPED

Amazon Ion SerDe は、Null 値が含まれる列をシリアル化または省略するように設定できます。厳密に型指定された null (TYPED) を書き出すことも、型指定されていない null (UNTYPED) を書き出すこともできます。厳密に型指定された null は、デフォルトの Amazon Ion から Hive への型マッピングに基づいて決定されます。

次の例では、厳密に型指定された null を指定しています。

```
'ion.serialize_null'='TYPED'
```

ion.ignore_malformed

オプションです。

デフォルト: false

値: true、false

`true` の場合、SerDe で読み取れない誤った形式があれば、該当するエントリまたはファイル全体が無視されます。詳細については、GitHub にあるドキュメントの「Ignore malformed」(誤った形式を無視する) を参照してください。

ion.*<column>*.serialize_as

オプションです。

デフォルト: 列のデフォルトの型。

値: Amazon Ion の型を含む文字列

値をシリアル化する Amazon Ion データ型を決定します。Amazon Ion 型と Hive 型が必ずしも直接マッピングされるわけではないため、Hive 型の中にはシリアル化できる有効なデータ型を複数含むものもあります。データをデフォルト以外のデータ型としてシリアル化するには、このプロパティを使用します。マッピングの詳細については、GitHub にある Amazon Ion の「型マッピング」ページを参照してください。

バイナリ Hive 列は、デフォルトでは Amazon Ion BLOB としてシリアル化されますが、Amazon Ion CLOB (キャラクタラージオブジェクト) としてシリアル化することもできます。次の例では、`example_hive_binary_column` 列を CLOB としてシリアル化しています。

```
'ion.example_hive_binary_column.serialize_as' = 'clob'
```

パスエクストラクタの使用

Amazon Ion はドキュメントスタイルのファイル形式ですが、Apache Hive はフラットな列指向形式です。`path extractors` という特別な Amazon Ion SerDe プロパティを使用して、この 2 つの形式をマッピングすることができます。パスエクストラクタは、階層的な Amazon Ion 形式をフラット化し、Amazon Ion の値を Hive 列にマッピングします。また、これを使用して、フィールドの名前を変更することもできます。

このエクストラクタは Athena で生成できますが、必要に応じて独自のエクストラクタを定義することもできます。

生成されるパスエクストラクタ

デフォルトでは、Athena は、Hive の列名と一致する最上位レベルの Amazon Ion 値を検索し、それらの一致する値に基づいてランタイムにパスエクストラクタを作成します。Amazon Ion のデータ形式が Hive テーブルスキーマと一致する場合、Athena は動的にエクストラクタを生成します。した

がって、それ以上パスエクストラクタを追加する必要はありません。このデフォルトのパスエクストラクタは、テーブルメタデータには保存されません。

次の例は、Athena が列名に基づいてエクストラクタを生成する方法を示しています。

```
-- Example Amazon Ion Document
{
    identification: {
        name: "John Smith",
        driver_license: "XXXX"
    },

    alias: "Johnny"
}

-- Example DDL
CREATE EXTERNAL TABLE example_schema2 (
    identification MAP<STRING, STRING>,
    alias STRING
)
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/path_extraction1/'
```

次に示すのは、Athena によって生成されるエクストラクタの例です。最初のエクストラクタで identification フィールドを identification 列に抽出し、次のエクストラクタで alias フィールドを alias 列に抽出しています。

```
'ion.identification.path_extractor' = '(identification)'
'ion.alias.path_extractor' = '(alias)'
```

次の例は、抽出されたテーブルを示しています。

```
|                  identification                    |  alias   |
|----------------------------------------------------|----------|
|{["name", "driver_license"],["John Smith", "XXXX"]} | "Johnny" |
```

独自のパスエクストラクタの指定

Amazon Ion フィールドが適切に Hive 列にマッピングされない場合は、独自のパスエクストラクタを指定することができます。CREATE TABLE ステートメントの WITH SERDEPROPERTIES 句で、次の構文を使用します。

```
WITH SERDEPROPERTIES (
   "ion.path_extractor.case_sensitive" = "<Boolean>",
   "ion.<column_name>.path_extractor" = "<path_extractor_expression>"
)
```

> **Note**
>
> デフォルトでは、パスエクストラクタで大文字と小文字は区別されません。この設定を上書きするには、ion.path_extractor.case_sensitive SerDe プロパティを `true` に設定します。

パスエクストラクタでの検索パスの使用

パスエクストラクタの SerDe プロパティの構文には *<path_extractor_expression>* が含まれています。

```
"ion.<column_name>.path_extractor" = "<path_extractor_expression>"
```

<path_extractor_expression> を使用し、Amazon Ion ドキュメントを解析して一致するデータを検索する検索パスを指定することができます。検索パスは括弧で囲み、次のコンポーネントをスペースで区切って 1 つ以上含めることができます。

- ワイルドカード — すべての値と一致します。
- インデックス — 指定された数値インデックスの値と一致します。インデックスは 0 から始まります。
- テキスト – 指定したテキストと同等のフィールド名を持つすべての値と一致します。
- 注釈 — 注釈が指定されているラップされたパスコンポーネントで指定された値と一致します。

次に示すのは、Amazon Ion ドキュメントと検索パスの例です。

```
-- Amazon Ion document
{
    foo: ["foo1", "foo2"] ,
    bar: "myBarValue",
    bar: A::"annotatedValue"
}
```

```
-- Example search paths
(foo 0)       # matches "foo1"
(1)           # matches "myBarValue"
(*)           # matches ["foo1", "foo2"], "myBarValue" and A::"annotatedValue"
()            # matches {foo: ["foo1", "foo2"] , bar: "myBarValue", bar:
 A::"annotatedValue"}
(bar)         # matches "myBarValue" and A::"annotatedValue"
(A::bar)      # matches A::"annotatedValue"
```

エクストラクタの例

フィールドのフラット化と名前の変更

次の例は、フィールドをフラット化して名前を変更する検索パスのセットを示しています。この例では、検索パスを使用して次の処理を実行します。

- nickname 列を alias フィールドにマッピングします。
- identification 構造体にある name サブフィールドに name 列をマッピングします。

次に、Amazon Ion ドキュメントの例を示します。

```
-- Example Amazon Ion Document
{
    identification: {
        name: "John Smith",
        driver_license: "XXXX"
    },

    alias: "Johnny"
}
```

次に、パスエクストラクタを定義する CREATE TABLE ステートメントの例を示します。

```
-- Example DDL Query
CREATE EXTERNAL TABLE example_schema2 (
    name STRING,
    nickname STRING
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
```

```
 'ion.nickname.path_extractor' = '(alias)',
 'ion.name.path_extractor' = '(identification name)'
 )
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/path_extraction2/'
```

次の例は、抽出されたデータを示しています。

```
-- Extracted Table
| name         |   nickname   |
|--------------|--------------|
| "John Smith" |  "Johnny"    |
```

検索パスの詳細、および検索パスのその他の例については、GitHub の「ion-java-path-extraction」(Ion Java パス抽出) ページを参照してください。

フライトデータのテキスト形式への抽出

次の例の CREATE TABLE クエリでは、WITH SERDEPROPERTIES を使用してパスエクストラクタを追加し、フライトデータを抽出して、出力エンコードを Amazon Ion テキストに指定しています。この例では STORED AS ION 構文を使用しています。

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'ion.encoding' = 'TEXT',
 'ion.yr.path_extractor'='(year)',
 'ion.quarter.path_extractor'='(results quarter)',
 'ion.month.path_extractor'='(date month)')
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

Avro SerDe

SerDe 名

Avro SerDe

ライブラリ名

org.apache.hadoop.hive.serde2.avro.AvroSerDe

例

セキュリティ上の理由から、Athena では、テーブルスキーマを指定するための avro.schema.url の使用がサポートされていません。avro.schema.literal を使用します。Avro 形式のデータからスキーマを抽出するには、Apache の avro-tools-<version>.jar で getschema パラメータを使用します。これによって返されるスキーマを WITH SERDEPROPERTIES ステートメントで使用できます。以下に例を示します。

```
java -jar avro-tools-1.8.2.jar getschema my_data.avro
```

avro-tools-<version>.jar ファイルは、インストールした Avro リリースの java サブディレクトリにあります。Avro をダウンロードする場合は、Apache Avro リリースを参照してください。Apache Avro Tools を直接ダウンロードする場合は、Apache Avro tools Maven リポジトリを参照してください。

スキーマを取得したら、CREATE TABLE ステートメントを使用して、Amazon S3 に保存されている基盤となる Avro データに基づいて Athena テーブルを作成します。Avro SerDe を指定するには、ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe' を使用します。次の例に示すように、テーブルの列名と対応するデータ型を指定する以外に、WITH SERDEPROPERTIES 句を使用してスキーマを指定する必要があります。

> **Note**
>
> s3://athena-examples-*myregion*/path/to/data/ の *myregion* を、Athena が実行されるリージョンの識別子 (s3://athena-examples-us-west-1/path/to/data/ など) に置き換えます。

```
CREATE EXTERNAL TABLE flights_avro_example (
   yr INT,
   flightdate STRING,
```

```
   uniquecarrier STRING,
   airlineid INT,
   carrier STRING,
   flightnum STRING,
   origin STRING,
   dest STRING,
   depdelay INT,
   carrierdelay INT,
   weatherdelay INT
)
PARTITIONED BY (year STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('avro.schema.literal'='
{
   "type" : "record",
   "name" : "flights_avro_subset",
   "namespace" : "default",
   "fields" : [ {
      "name" : "yr",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "flightdate",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "uniquecarrier",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "airlineid",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "carrier",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "flightnum",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "origin",
      "type" : [ "null", "string" ],
```

```
      "default" : null
   }, {
      "name" : "dest",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "depdelay",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "carrierdelay",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "weatherdelay",
      "type" : [ "null", "int" ],
      "default" : null
    } ]
}
')
STORED AS AVRO
LOCATION 's3://athena-examples-myregion/flight/avro/';
```

テーブルに対して MSCK REPAIR TABLE ステートメントを実行し、パーティションのメタデータを更新します。

```
MSCK REPAIR TABLE flights_avro_example;
```

出発の合計数で上位 10 の出発都市をクエリします。

```
SELECT origin, count(*) AS total_departures
FROM flights_avro_example
WHERE year >= '2000'
GROUP BY origin
ORDER BY total_departures DESC
LIMIT 10;
```

> **Note**
>
> フライトテーブルデータは、米国運輸省の運輸統計局が提供する Flights からのものです。オリジナルを白黒に変換しています。

Grok SerDe

Logstash Grok SerDe は、構造化されていないテキストデータ (通常はログ) の逆シリアル化専用のパターンセットで構成されるライブラリです。各 Grok パターンは名前付きの正規表現です。必要に応じて、これらの逆シリアル化パターンを特定し、再利用できます。このため、正規表現を使用するよりも Grok を使用するほうが簡単です。Grok は、定義済みパターンのセットを提供します。カスタムパターンを作成することもできます。

Athena でテーブルを作成するときに Grok SerDe を指定するには、`ROW FORMAT SERDE 'com.amazonaws.glue.serde.GrokSerDe'` 句を使用し、この句の後に、データで一致させるパターンを指定する `WITH SERDEPROPERTIES` 句を使用します。詳細を以下に示します。

- `input.format` 式は、データでマッチさせるパターンを定義します。この値は必須です。
- `input.grokCustomPatterns` 式は、カスタムの名前付きパターンを定義します。後で、このパターンを `input.format` 式内で使用できます。これは省略可能です。複数のパターンエントリを `input.grokCustomPatterns` 式に含めるには、改行のエスケープ文字 (\n) を使用して、次のように区切ります。`'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\\]]*)'`。
- `STORED AS INPUTFORMAT` 句と `OUTPUTFORMAT` 句は必須です。
- `LOCATION` 句は、Amazon S3 バケットを指定します。このバケットには、複数のデータオブジェクトを含めることができます。バケット内のすべてのデータオブジェクトが逆シリアル化されてテーブルが作成されます。

例

以下の例は、Grok の定義済みパターンのリストに依存します。「定義済みパターン」を参照してください。

例 1

この例では、`s3://DOC-EXAMPLE-BUCKET/groksample/` に保存された Postfix のメールログのエントリからソースデータを使用します。

```
Feb  9 07:15:00 m4eastmail postfix/smtpd[19305]: B88C4120838: connect from
 unknown[192.168.55.4]
Feb  9 07:15:00 m4eastmail postfix/smtpd[20444]: B58C4330038:
 client=unknown[192.168.55.4]
Feb  9 07:15:03 m4eastmail postfix/cleanup[22835]: BDC22A77854: message-
id=<31221401257553.5004389LCBF@m4eastmail.example.com>
```

以下のステートメントは、ユーザー指定のカスタムパターンと事前定義されたパターンを使用して、ソースデータから mygroktable という名前のテーブルを Athena で作成します。

```
CREATE EXTERNAL TABLE `mygroktable`(
   syslogbase string,
   queue_id string,
   syslog_message string
   )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.grokCustomPatterns' = 'POSTFIX_QUEUEID [0-9A-F]{7,12}',
   'input.format'='%{SYSLOGBASE} %{POSTFIX_QUEUEID:queue_id}:
 %{GREEDYDATA:syslog_message}'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://DOC-EXAMPLE-BUCKET/groksample/';
```

%{NOTSPACE:column} などのシンプルなパターンから始め、最初に列をマッピングし、次に必要に応じて列を特化します。

例 2

次の例では、Log4j ログのクエリを作成します。ログの例には、次の形式のエントリが含まれています。

```
2017-09-12 12:10:34,972 INFO  - processType=AZ, processId=ABCDEFG614B6F5E49,
 status=RUN,
threadId=123:amqListenerContainerPool23P:AJ|ABCDE9614B6F5E49||
2017-09-12T12:10:11.172-0700],
executionTime=7290, tenantId=12456, userId=123123f8535f8d76015374e7a1d87c3c,
 shard=testapp1,
jobId=12312345e5e7df0015e777fb2e03f3c, messageType=REAL_TIME_SYNC,
action=receive, hostname=1.abc.def.com
```

このログデータをクエリするには

- Grok パターンを各列の input.format に追加します。たとえば、timestamp に %{TIMESTAMP_ISO8601:timestamp} を追加します。loglevel に %{LOGLEVEL:loglevel} を追加します。
- ログ形式のエントリを区切るダッシュ (-) とカンマをマッピングすることで、input.format のパターンが口グの形式と正確に一致することを確認します。

```
CREATE EXTERNAL TABLE bltest (
 timestamp STRING,
 loglevel STRING,
 processtype STRING,
 processid STRING,
 status STRING,
 threadid STRING,
 executiontime INT,
 tenantid INT,
 userid STRING,
 shard STRING,
 jobid STRING,
 messagetype STRING,
 action STRING,
 hostname STRING
 )
ROW FORMAT SERDE 'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
"input.grokCustomPatterns" = 'C_ACTION receive|send',
"input.format" = "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:loglevel} - processType=
%{NOTSPACE:processtype}, processId=%{NOTSPACE:processid}, status=%{NOTSPACE:status},
 threadId=%{NOTSPACE:threadid}, executionTime=%{POSINT:executiontime}, tenantId=
%{POSINT:tenantid}, userId=%{NOTSPACE:userid}, shard=%{NOTSPACE:shard}, jobId=
%{NOTSPACE:jobid}, messageType=%{NOTSPACE:messagetype}, action=%{C_ACTION:action},
 hostname=%{HOST:hostname}"
) STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/samples/';
```

例 3

クエリ例 'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\\]]*)') からのこのスニペットにあるように、Amazon S3 ログをクエリする以下の例には、改行エスケープ文字 ('input.grokCustomPatterns') で区切られた 2 つのパターンエントリが含まれる \n 式が含まれています。

```
CREATE EXTERNAL TABLE `s3_access_auto_raw_02`(
  `bucket_owner` string COMMENT 'from deserializer',
  `bucket` string COMMENT 'from deserializer',
  `time` string COMMENT 'from deserializer',
  `remote_ip` string COMMENT 'from deserializer',
  `requester` string COMMENT 'from deserializer',
  `request_id` string COMMENT 'from deserializer',
  `operation` string COMMENT 'from deserializer',
  `key` string COMMENT 'from deserializer',
  `request_uri` string COMMENT 'from deserializer',
  `http_status` string COMMENT 'from deserializer',
  `error_code` string COMMENT 'from deserializer',
  `bytes_sent` string COMMENT 'from deserializer',
  `object_size` string COMMENT 'from deserializer',
  `total_time` string COMMENT 'from deserializer',
  `turnaround_time` string COMMENT 'from deserializer',
  `referrer` string COMMENT 'from deserializer',
  `user_agent` string COMMENT 'from deserializer',
  `version_id` string COMMENT 'from deserializer')
ROW FORMAT SERDE
  'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
  'input.format'='%{NOTSPACE:bucket_owner} %{NOTSPACE:bucket} \
\[%{INSIDE_BRACKETS:time}\\] %{NOTSPACE:remote_ip} %{NOTSPACE:requester}
 %{NOTSPACE:request_id} %{NOTSPACE:operation} %{NOTSPACE:key} \"?
%{INSIDE_QS:request_uri}\"? %{NOTSPACE:http_status} %{NOTSPACE:error_code}
 %{NOTSPACE:bytes_sent} %{NOTSPACE:object_size} %{NOTSPACE:total_time}
 %{NOTSPACE:turnaround_time} \"?%{INSIDE_QS:referrer}\"? \"?%{INSIDE_QS:user_agent}\"?
 %{NOTSPACE:version_id}',
  'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\\]]*)')
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET'
```

JSON SerDe ライブラリ

Athena では、SerDe ライブラリを使用して、JSON データを逆シリアル化できます。逆シリアル化では、JSON データを変換し、Parquet や ORC のような別の形式にシリアル化 (書き出し) できるようにします。

- ネイティブの Hive JSON SerDe
- OpenX JSON SerDe
- Amazon Ion Hive SerDe

> **Note**
>
> Hive および OpenX ライブラリでは、JSON データが単一行で (フォーマットされていない)、そのレコードは改行文字で区切られていることが想定されています。Ion データ形式は JSON のスーパーセットであるため、Amazon Ion Hive SerDe にはその要件がなく、代替として使用できます。

ライブラリ名

以下のいずれかを使用します。

org.apache.hive.hcatalog.data.JsonSerDe

org.openx.data.jsonserde.JsonSerDe

com.amazon.ionhiveserde.IonHiveSerDe

Hive JSON SerDe

Hive JSON SerDe は通常、イベントなどの JSON データを処理するために使用されます。これらのイベントは、改行で区切られ JSON 形式でエンコードされたテキストの 1 行の文字列として表現されます。Hive JSON SerDe では、`map` または `struct` キー名に重複するキーを使用できません。

> **Note**
>
> SerDe では、各 JSON ドキュメントが、レコード内のフィールドを区切る行終端文字なしの、1 行のテキストに収まっていることを想定しています。JSON テキストがプリティプリント形式の場合、テーブルを作成した後にクエリを実行しようとすると、以下のようなエラーメッセージが表示される場合があります。「HIVE_CURSOR_ERROR: Row is not a valid JSON Object」、または「HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT」。詳細については、GitHub の OpenX SerDe のドキュメントで「JSON Data Files」(JSON データファイル) を参照してください。

次の DDL ステートメントの例では、Hive JSON SerDe を使用して、サンプルオンライン広告データに基づいてテーブルを作成します。LOCATION 句で、s3://DOC-EXAMPLE-BUCKET.elasticmapreduce/samples/hive-ads/tables/impressions の *myregion* を、Athena を実行するリージョンの識別子 (s3://us-west-2.elasticmapreduce/samples/hive-ads/tables/impressions など) に置き換えます。

```
CREATE EXTERNAL TABLE impressions (
    requestbegintime string,
    adid string,
    impressionid string,
    referrer string,
    useragent string,
    usercookie string,
    ip string,
    number string,
    processid string,
    browsercookie string,
    requestendtime string,
    timers struct
                <
                 modellookup:string,
                 requesttime:string
                >,
    threadid string,
    hostname string,
    sessionid string
)
PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET.elasticmapreduce/samples/hive-ads/tables/
impressions';
```

Hive JSON SerDeでのタイムスタンプ形式の指定

文字列からタイムスタンプ値を解析するには、WITH SERDEPROPERTIES サブフィールドを ROW FORMAT SERDE 句に追加し、これを使用して、timestamp.formats パラメータを指定します。パラメータで、次の例のように、1 つまたは複数のタイムスタンプパターンのコンマ区切りリストを指定します。

```
...
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
```

```
WITH SERDEPROPERTIES ("timestamp.formats"="yyyy-MM-dd'T'HH:mm:ss.SSS'Z',yyyy-MM-
dd'T'HH:mm:ss")
...
```

詳細については、Apache Hive ドキュメントの「タイムスタンプ」を参照してください。

クエリのためのテーブルのロード

テーブルを作成したら、MSCK REPAIR TABLE を実行してテーブルをロードし、Athena からクエリできるようにします。

```
MSCK REPAIR TABLE impressions
```

CloudTrail のログをクエリします

Hive JSON SerDe を使って CloudTrail のログをクエリすることができます。詳細情報と例 CREATE TABLE については、「AWS CloudTrail ログのクエリ」を参照してください。

OpenX JSON SerDe

Hive JSON SerDe と同様に、OpenX JSON を使用して JSON データを処理できます。また、データは改行で区切られ JSON 形式でエンコードされたテキストの 1 行の文字列としても表現されます。Hive JSON SerDe と同様に、OpenX JSON SerDe では、`map` または `struct` キー名の重複キーは許可されません。

> **Note**
>
> SerDe では、各 JSON ドキュメントが、レコード内のフィールドを区切る行終端文字なしの、1 行のテキストに収まっていることを想定しています。JSON テキストがプリティプリント形式の場合、テーブルを作成した後にクエリを実行しようとすると、以下のようなエラーメッセージが表示される場合があります。「HIVE_CURSOR_ERROR: Row is not a valid JSON Object」、または「HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT」。詳細については、GitHub の OpenX SerDe のドキュメントで「JSON Data Files」(JSON データファイル) を参照してください。

オプションのプロパティ

Hive JSON SerDe とは異なり、OpenX JSON SerDe には、データの不整合に対処するのに役立つ次のオプションの SerDe プロパティもあります。

ignore.malformed.json

オプション。TRUE に設定すると、不正な形式の JSON 構文を無視できます。デフォルト: FALSE。

dots.in.keys

オプション。デフォルト: FALSE。TRUE に設定すると、SerDe はキー名のドットをアンダースコアに置き換えることができます。例えば、JSON データセットに "a.b" という名前のキーが含まれている場合は、このプロパティを使用して、Athena で列名が "a_b" になるように定義できます。デフォルトで (この SerDe がない場合)、Athena は列名にドットを許可しません。

case.insensitive

オプション。デフォルト: TRUE。TRUE に設定すると、SerDe はすべての大文字の列を小文字に変換します。

データで大文字と小文字を区別するキー名を使用するには、WITH SERDEPROPERTIES ("case.insensitive"= FALSE;) を使用します。次に、まだすべてが小文字になっていないすべてのキーに、次の構文を使用して列名からプロパティ名へのマッピングを指定します。

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("case.insensitive" = "FALSE", "mapping.userid" = "userId")
```

URL と Url のように小文字である 2 つのキーがある場合は、次のような エラーが発生する可能性があります。

HIVE_CURSOR_ERROR: Row is not a valid JSON Object - JSONException: Duplicate key "url"

これを解決するには、次の例のように、case.insensitive プロパティを FALSE に設定し、キーを異なる名前にマッピングします。

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("case.insensitive" = "FALSE", "mapping.url1" = "URL",
 "mapping.url2" = "Url")
```

マッピングを

オプション。列名を、列名と同一ではない JSON キーにマップします。mapping パラメータは、JSON データにキーワードのキーが含まれている場合に便利です。たとえば、timestamp

という名前の JSON キーがある場合、次の構文を使用して、キーを ts という名前の列にマッピングします。

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("mapping.ts" = "timestamp")
```

コロンを含むネストされたフィールド名を Hive 互換の名前にマッピングする

struct の内部にコロンを含むフィールド名がある場合は、mapping プロパティを使用してそのフィールドを Hive 互換の名前にマップできます。例えば、列タイプの定義に my:struct:field:string を含めることで、WITH SERDEPROPERTIES で次のエントリー定義を my_struct_field:string にマップできます。

```
("mapping.my_struct_field" = "my:struct:field")
```

次の例は、対応する CREATE TABLE ステートメントを示しています。

```
CREATE EXTERNAL TABLE colon_nested_field (
item struct<my_struct_field:string>)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("mapping.my_struct_field" = "my:struct:field")
```

例: 広告データ

次の DDL ステートメントの例では、OpenX JSON SerDe を使用して、Hive JSON SerDe の例で使用したのと同じサンプルオンライン広告データに基づいてテーブルを作成します。LOCATION 句で、*myregion* を Athena が実行されるリージョンの識別子に置き換えます。

```
CREATE EXTERNAL TABLE impressions (
    requestbegintime string,
    adid string,
    impressionId string,
    referrer string,
    useragent string,
    usercookie string,
    ip string,
    number string,
    processid string,
    browsercokie string,
```

```
    requestendtime string,
    timers struct<
       modellookup:string,
       requesttime:string>,
    threadid string,
    hostname string,
    sessionid string
)   PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET.elasticmapreduce/samples/hive-ads/tables/
impressions';
```

例: ネスト JSON の逆シリアル化

JSON SerDe を使用して、より複雑な JSON エンコードされたデータを解析できます。このためには、ネストされた構造を表すために `struct` 要素と `array` 要素を使用する `CREATE TABLE` ステートメントを使用する必要があります。

以下の例は、ネストされた構造を持つ JSON データから Athena テーブルを作成します。JSON でエンコードされたデータを Athena で解析するには、各 JSON ドキュメントが改行で区切られた独自の行にあることを確認します。

この例で、JSON エンコード形式のデータ構造は次のとおりとします。

```
{
"DocId": "AWS",
"User": {
        "Id": 1234,
        "Username": "bob1234",
        "Name": "Bob",
"ShippingAddress": {
"Address1": "123 Main St.",
"Address2": null,
"City": "Seattle",
"State": "WA"
   },
"Orders": [
   {
     "ItemId": 6789,
     "OrderDate": "11/11/2017"
   },
   {
     "ItemId": 4352,
```

```
      "OrderDate": "12/12/2017"
    }
   ]
  }
}
```

次の CREATE TABLE ステートメントは、Openx-JsonSerDe を struct および array コレクションデータ型とともに使用して、オブジェクトのグループを構築します。各 JSON ドキュメントは個別の行にリストされ、新しい行で区切られます。エラーを回避するため、クエリ中のデータには、struct またはマップキー名の重複するキーは含まれません。

```
CREATE external TABLE complex_json (
   docid string,
   `user` struct<
               id:INT,
               username:string,
               name:string,
               shippingaddress:struct<
                                address1:string,
                                address2:string,
                                city:string,
                                state:string
                                >,
               orders:array<
                           struct<
                                itemid:INT,
                                 orderdate:string
                                 >
                             >
               >
   )
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/myjsondata/';
```

追加リソース

Athena での JSON およびネストされた JSON の使用の詳細については、以下のリソースを参照してください。

- JSONSerDe を使用してネストされた JSON とマッピングから Amazon Athena でテーブルを作成する (AWS ビッグデータブログ)

- Amazon Athena で JSON データを読み込もうとするとエラーが発生する (AWS ナレッジセンターの記事)
- hive-json-schema (GitHub) – JSON ドキュメント例から CREATE TABLE ステートメントを生成する、Java で記述されたツールです。生成される CREATE TABLE ステートメントは、OpenX JSON Serde を使用します。

CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe

この SerDe の指定はオプションです。これは、Athena がデフォルトで使用する CSV、TSV、およびカスタム区切りの形式のデータ用の SerDe です。どの SerDe も指定せずに ROW FORMAT DELIMITED のみを指定すると、この SerDe が自動的に使用されます。データ内に引用符で囲まれた値がない場合は、この SerDe を使用します。

LazySimpleSerDe に関するリファレンスドキュメントについては、Apache Hive Developer Guide の「Hive SerDe」セクションを参照してください。

ライブラリ名

LazySimpleSerDe のクラスライブラリ名は org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe です。LazySimpleSerDe クラスの詳細については、GitHub.com で「LazySimpleSerDe.java」を参照してください。

ヘッダーの無視

テーブルを定義するときにデータ内のヘッダーを無視するには、以下の例にあるように、skip.header.line.count テーブルプロパティを使用できます。

```
TBLPROPERTIES ("skip.header.line.count"="1")
```

例については、「Amazon VPC フローログのクエリ」および「Amazon CloudFront ログのクエリ」の CREATE TABLE ステートメントを参照してください。

CSV の例

以下の例は、Athena で CSV データからテーブルを作成するために LazySimpleSerDe を使用する方法を示しています。この SerDe を使用してカスタム区切りファイルをデシリアライズするには、これらの例のパターンに従いますが、FIELDS TERMINATED BY 句を使用して異なる単一文字の区切り文字を指定します。LazySimpleSerDe は、複数文字の区切り文字をサポートしていません。

> **Note**
>
> s3://athena-examples-*myregion*/path/to/data/ の *myregion* を、Athena が実行されるリージョンの識別子 (s3://athena-examples-us-west-1/path/to/data/ など) に置き換えます。

CREATE TABLE ステートメントを使用して、Amazon S3 に保存されている CSV 形式の基盤となるデータから Athena テーブルを作成します。

```
CREATE EXTERNAL TABLE flight_delays_csv (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
```

```
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
```

```
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
    PARTITIONED BY (year STRING)
    ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      ESCAPED BY '\\'
      LINES TERMINATED BY '\n'
    LOCATION 's3://athena-examples-myregion/flight/csv/';
```

このテーブルに新しいパーティションを追加するたびに、MSCK REPAIR TABLE を実行してパーティションメタデータを更新します。

```
MSCK REPAIR TABLE flight_delays_csv;
```

遅延が 1 時間を超えた上位 10 便をクエリします。

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_csv
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

> **Note**
>
> フライトテーブルデータは、米国運輸省の運輸統計局が提供する Flights からのものです。オリジナルを白黒に変換しています。

TSV の例

Amazon S3 に保存されている TSV データから Athena テーブルを作成するには、ROW FORMAT DELIMITED を使用して \t をタブフィールドの区切り文字、\n を区切り文字、および \ をエスケープ文字として指定します。次の抜粋は構文例を示します。athena-examples の場所にはサンプルの TSV フライトデータはありませんが、CSV テーブルと同様に、MSCK REPAIR TABLE を実行して新しいパーティションが追加されるたびにパーティションメタデータを更新する必要があります。

```
...
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
ESCAPED BY '\\'
LINES TERMINATED BY '\n'
...
```

CSV を処理するための OpenCSVSerDe

CSV データ用の Athena テーブルを作成する場合は、データに含まれる値のタイプに基づいて、使用する SerDe を選択します。

- データに二重引用符 (") で囲まれた値が含まれる場合は、Athena で OpenCSV SerDe を使用して値を逆シリアル化できます。データに二重引用符 (") で囲まれた値が含まれていない場合は、SerDe の指定を省略できます。この場合、Athena はデフォルトの LazySimpleSerDe を使用します。詳細については、CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe を参照してください。
- データに UNIX の TIMESTAMP 数値 (1579059880000 など) がある場合は、OpenCSVSerDe を使用します。データが java.sql.Timestamp 形式を使用する場合は、LazySimpleSerDe を使用します。

CSV SerDe (OpenCSVSerDe)

OpenCSV SerDe には、文字列データに関して次の特性があります。

- デフルルトの引用文字は二重引用符 (") を使用し、区切り文字、引用符、および エスケープ文字を指定できます。

```
WITH SERDEPROPERTIES ("separatorChar" = ",", "quoteChar" = "`", "escapeChar" = "\\" )
```

- \t または \n を直接エスケープすることはできません。それらを エスケープするには、"escapeChar" = "\\" を使用します。このトピックの例を参照してください。
- CSV ファイルの埋め込み改行はサポートしません。

STRING 以外のデータ型の場合、OpenCSVSerDe が次のように動作します。

- BOOLEAN、BIGINT、INT、および DOUBLE データ型を認識します。
- 数値データ型として定義された列の空値または null 値を認識せず、string として残します。回避策の 1 つは、null 値を string とした列を作成してから、CAST を使用してクエリのフィールドを数値データ型に変換して、null の場合にデフォルト値の 0 を指定することです。詳細については、AWS ナレッジセンターの「Athena で CSV データをクエリすると、『HIVE_BAD_DATA: フィールド値の解析エラー』というエラーが表示されます」を参照してください。
- CREATE TABLE ステートメントで timestamp データ型で指定された列については、1579059880000 などミリ秒単位の UNIX 数値形式で指定されている場合は TIMESTAMP データを認識します。
  - OpenCSVSerDe は、"YYYY-MM-DD HH:MM:SS.fffffffff" (小数点以下 9 桁の精度) などの JDBC 準拠の java.sql.Timestamp 形式の TIMESTAMP をサポートしていません。

- CREATE TABLE ステートメントで DATE データ型で指定された列については、値が 1970 年 1 月 1 日からの経過日数を表す場合、値を日付として認識します。例えば、date データ型の列の値 18276 は、クエリを実行すると 2020-01-15 と出力されます。この UNIX 形式では、1 日は 86,400 秒です。
  - OpenCSVSerDe では、これ以外の形式の DATE は直接サポートしていません。他の形式のタイムスタンプデータを処理するには、列を string として定義してから、時刻変換関数を使用して SELECT クエリで求める結果を返します。詳細については、AWS ナレッジセンターの「Amazon Athena のテーブルにクエリを実行すると、TIMESTAMP の結果が空になる」を参照してください。
- テーブル内の列を希望の型にさらに変換するには、テーブル上にビューを作成し、CAST を使用して目的の型に変換します。

Example 例: UNIX 数値形式で指定された TIMESTAMP 型と DATE 型の使用。

次のカンマで区切ったデータから成る 3 つの列を検討してください。各列の値を二重引用符で囲みます。

```
"unixvalue creationdate 18276 creationdatetime 1579059880000","18276","1579059880000"
```

以下のステートメントは、指定された Amazon S3 バケットの場所から Athena にテーブルを作成します。

```
CREATE EXTERNAL TABLE IF NOT EXISTS testtimestamp1(
 `profile_id` string,
 `creationdate` date,
 `creationdatetime` timestamp
 )
 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
 LOCATION 's3://DOC-EXAMPLE-BUCKET'
```

次に、以下のクエリを実行します。

```
SELECT * FROM testtimestamp1
```

クエリは、日付と時刻のデータを示す次の結果を返します。

```
profile_id                                                    creationdate
 creationdatetime
```

```
unixvalue creationdate 18276 creationdatetime 1579146280000       2020-01-15
 2020-01-15 03:44:40.000
```

Example 例: **\t** または **\n** をエスケープ

以下のテストデータの場合を考えます。

```
" \\t\\t\\n 123 \\t\\t\\n ",abc
" 456 ",xyz
```

以下のステートメントは、Athena にテーブルを作成し、"escapeChar" = "\\" を指定します。

```
CREATE EXTERNAL TABLE test1 (
f1 string,
s2 string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES ("separatorChar" = ",", "escapeChar" = "\\")
LOCATION 's3://DOC-EXAMPLE-BUCKET/dataset/test1/'
```

次に、以下のクエリを実行します。

```
SELECT * FROM test1;
```

この結果は、\t または \n を正しくエスケープして返されます。

```
f1            s2
\t\t\n 123 \t\t\n            abc
456                          xyz
```

SerDe 名

CSV SerDe

ライブラリ名

この SerDe を使用するには、その完全修飾されたクラス名を ROW FORMAT SERDE の後に指定します。また、次に示すように SERDEPROPERTIES 内に区切り記号を指定します。

```
...
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
```

```
  "quoteChar"     = "`",
  "escapeChar"    = "\\"
)
```

ヘッダーの無視

テーブルを定義するときにデータ内のヘッダーを無視するには、以下の例にあるように、skip.header.line.count テーブルプロパティを使用できます。

```
TBLPROPERTIES ("skip.header.line.count"="1")
```

例については、「Amazon VPC フローログのクエリ」および「Amazon CloudFront ログのクエリ」の CREATE TABLE ステートメントを参照してください。

例

次の例では、CSV のデータが s3://DOC-EXAMPLE-BUCKET/mycsv/ に保存されており、コンテンツが以下のとおりであるとします。

```
"a1","a2","a3","a4"
"1","2","abc","def"
"a","a1","abc3","ab4"
```

CREATE TABLE ステートメントを使用して、このデータに基づいた Athena テーブルを作成します。以下の例にあるように、ROW FORMAT SERDE の後で OpenCSVSerDe クラスを参照し、文字の区切り文字、引用符文字、およびエスケープ文字を WITH SERDEPROPERTIES で指定します。

```
CREATE EXTERNAL TABLE myopencsvtable (
   col1 string,
   col2 string,
   col3 string,
   col4 string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   'separatorChar' = ',',
   'quoteChar' = '"',
   'escapeChar' = '\\'
   )
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/mycsv/';
```

テーブル内のすべての値に対してクエリを実行します。

```
SELECT * FROM myopencsvtable;
```

クエリは次の値を返します。

```
col1     col2    col3    col4
-----------------------------
a1       a2      a3      a4
1        2       abc     def
a        a1      abc3    ab4
```

ORC SerDe

SerDe 名

OrcSerDe

ライブラリ名

このライブラリは ORC 形式のデータに Apache Hive OrcSerde.java クラスを使用します。オブジェクトを ORC からリーダーに、および ORC からライターに渡します。

例

> **Note**
>
> s3://athena-examples-*myregion*/path/to/data/ の *myregion* を、Athena が実行されるリージョンの識別子 (s3://athena-examples-us-west-1/path/to/data/ など) に置き換えます。

次の例では、ORC で航空便の遅延データのテーブルを作成します。テーブルにはパーティションが含まれます。

```
DROP TABLE flight_delays_orc;
CREATE EXTERNAL TABLE flight_delays_orc (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
```

```
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
```

```
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
```

```
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year String)
STORED AS ORC
LOCATION 's3://athena-examples-myregion/flight/orc/'
tblproperties ("orc.compress"="ZLIB");
```

テーブルに対して MSCK REPAIR TABLE ステートメントを実行し、パーティションのメタデータを更新します。

```
MSCK REPAIR TABLE flight_delays_orc;
```

次のクエリでは、遅延が 1 時間を超えた上位 10 便を取得します。

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_orc
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

Parquet SerDe

SerDe 名

ParquetHiveSerDe は、Parquet 形式で保存されているデータに使用します。

> **Note**
>
> データを Parket 形式に変換するには、CREATE TABLE AS SELECT (CTAS) クエリを使用できます。詳細については、「クエリ結果からのテーブルの作成 (CTAS)」、「CTAS クエリの例」、および「ETL およびデータ分析での CTAS および INSERT INTO の使用」を参照してください。

ライブラリ名

Athena は Parquet で保存されているデータを逆シリアル化する必要があるときに、`org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe` クラスを使用します。

例: Parquet で保存されているファイルのクエリ

> **Note**
>
> `s3://athena-examples-myregion/path/to/data/` の *myregion* を、Athena が実行されるリージョンの識別子 (`s3://athena-examples-us-west-1/path/to/data/` など) に置き換えます。

以下の `CREATE TABLE` ステートメントを使用して、Amazon S3 に Parquet 形式で保存されている基盤となるデータから Athena テーブルを作成します。

```
CREATE EXTERNAL TABLE flight_delays_pq (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
```

```
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
```

```
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
```

```
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year STRING)
STORED AS PARQUET
LOCATION 's3://athena-examples-myregion/flight/parquet/'
tblproperties ("parquet.compression"="SNAPPY");
```

テーブルに対して MSCK REPAIR TABLE ステートメントを実行し、パーティションのメタデータを更新します。

```
MSCK REPAIR TABLE flight_delays_pq;
```

遅延が 1 時間を超えた上位 10 便をクエリします。

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_pq
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

> **Note**
>
> フライトテーブルデータは、米国運輸省の運輸統計局が提供する Flights からのものです。オリジナルを白黒に変換しています。

Parquet 統計を無視する

Parquet データを読み取ると、次のようなエラーメッセージが表示されることがあります。

```
HIVE_CANNOT_OPEN_SPLIT: Index x out of bounds for length y
HIVE_CURSOR_ERROR: Failed to read x bytes
HIVE_CURSOR_ERROR: FailureException at Malformed input: offset=x
```

```
HIVE_CURSOR_ERROR: FailureException at java.io.IOException:
can not read class org.apache.parquet.format.PageHeader: Socket is closed by peer.
```

この問題を回避するには、次の例のように CREATE TABLE ステートメントまたは ALTER TABLE SET TBLPROPERTIES ステートメントを使用して Parquet SerDe `parquet.ignore.statistics` プロパティを `true` に設定します。

CREATE TABLE の例

```
...
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES (
'parquet.ignore.statistics'='true')
STORED AS PARQUET
...
```

ALTER TABLE の例

```
ALTER TABLE ... SET TBLPROPERTIES ('parquet.ignore.statistics'='true')
```

Regex SerDe

Regex SerDe では、正規表現 (REGEX) を使用し、正規表現グループをテーブルの列に抽出してデータを逆シリアル化します。

データ内の行が正規表現と一致しない場合、その行内のすべての列が NULL として返されます。行が正規表現と一致するが、予想よりもグループが少ない場合、欠落しているグループは NULL です。データ内の行が正規表現と一致するが、正規表現内のグループよりも多くの列がある場合、追加の列は無視されます。

詳細については、Apache Hive ドキュメントの「Class RegexSerDe」を参照してください。

SerDe 名

RegexSerDe

ライブラリ名

RegexSerDe

例

以下の例は、RegExSerDe を使用して CloudFront ログからテーブルを作成します。s3://athena-examples-*myregion*/cloudfront/plaintext/ の *myregion* を、Athena が実行されるリージョンの識別子 (s3://athena-examples-us-west-1/cloudfront/plaintext/ など) に置き換えます。

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  os STRING,
  Browser STRING,
  BrowserVersion STRING
 )
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
 WITH SERDEPROPERTIES (
 "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\
\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^\/]+)[\/]
(.*)$"
 )
LOCATION 's3://athena-examples-myregion/cloudfront/plaintext/';
```

Amazon Athena を使用した SQL クエリの実行

AWS Glue Data Catalog に登録されているデータソース、および Hive メタストアや Amazon DocumentDB インスタンスなどの Athena の横串検索を使用して接続するデータソースで Amazon Athenaを使用して SQL クエリを実行できます。データソースの使用の詳細については、「データソースへの接続」を参照してください。スキーマを変更するデータ定義言語 (DDL) クエリを実行すると、Athena がメタデータをデータソースに関連付けられたメタストアに書き込みます。さらに、CREATE TABLE AS や INSERT INTO などの一部のクエリは、データセットにレコードを書き込むことができます。例えば、CSV レコードを Simple Storage Service (Amazon S3) ロケーションに追加できます。クエリを実行すると、Athena がユーザー指定のクエリ結果の場所にクエリの結果

を保存します。これにより、クエリ履歴を表示し、クエリ結果セットをダウンロードして表示できます。

このセクションでは、さまざまな SQL ステートメントを使用して、一般的なデータソースとデータ型で Athena クエリを実行するためのガイダンスを提供します。一般的な構造と演算子での作業 (例えば、配列の使用、連結、フィルタリング、フラット化、および並べ替えなど) に対する一般的なガイダンスが提供されます。その他の例としては、ネスト化された構造とマッピングのテーブル内のデータのクエリ、JSON エンコード形式のデータセットに基づくテーブル、AWS CloudTrail ログや Amazon EMR ログなどの AWS のサービス に関連付けられたデータセットなどがあります。標準的な SQL の使用に関する包括的な説明は、このドキュメントの対象範囲外です。SQL の詳細については、Trino および Presto の言語リファレンスを参照してください。

トピック

- SQL クエリの実行プランの表示
- クエリ結果、最近のクエリ、および出力ファイルの使用
- クエリの結果を再利用する
- 完了したクエリの統計と実行の詳細の表示
- ビューの使用
- 保存されたクエリの使用
- パラメータ化されたクエリの使用
- コストベースオプティマイザーの使用
- S3 Express One Zone データのクエリ
- 復元された Amazon S3 Glacier オブジェクトへのクエリ
- スキーマ更新の処理
- 配列のクエリ
- 地理空間データのクエリ
- JSON のクエリ
- Machine Learning (ML) with Amazon Athena の使用
- ユーザー定義関数を使用したクエリ
- リージョン間のクエリ
- AWS Glue Data Catalog のクエリ
- AWS のサービス ログのクエリ
- Simple Storage Service (Amazon S3) に保存されたウェブサーバーログのクエリ

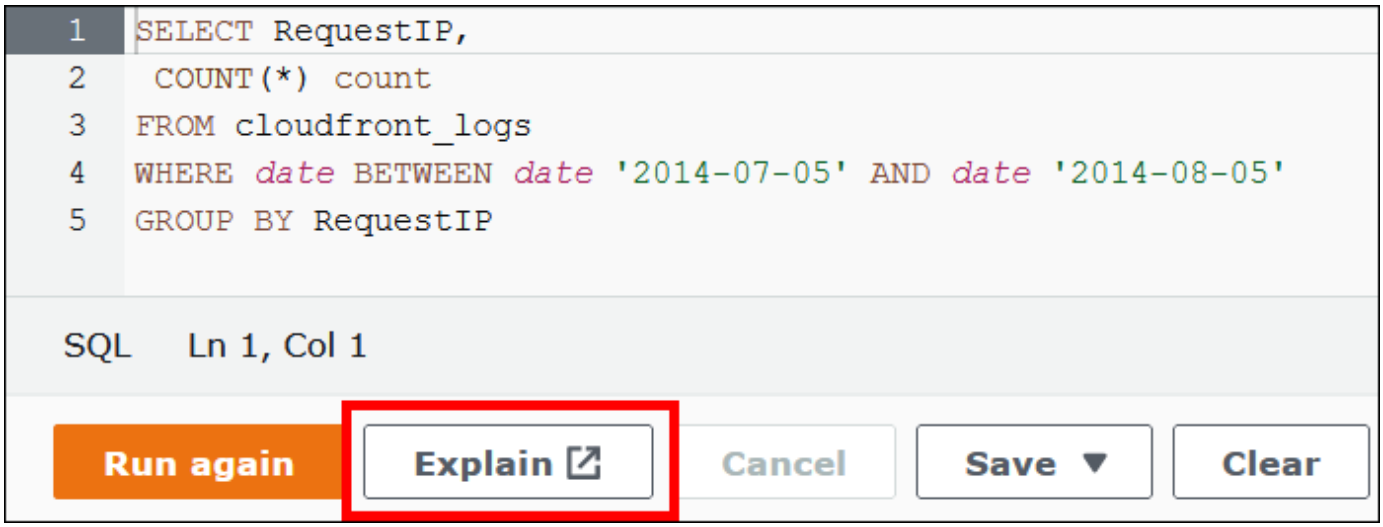
考慮事項と制限事項については、「Amazon Athena での SQL クエリに関する考慮事項と制約事項」を参照してください。

SQL クエリの実行プランの表示

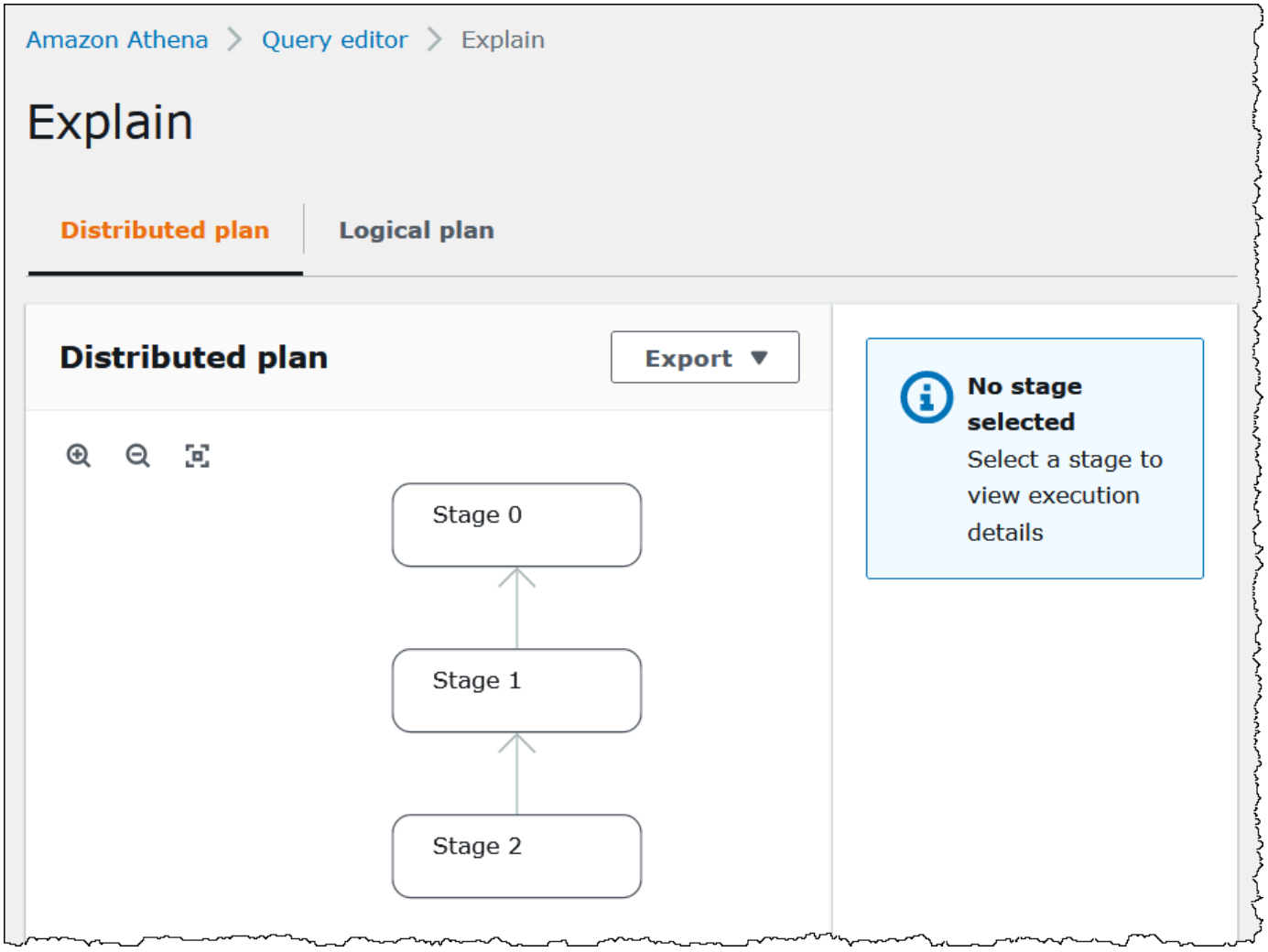
Athena クエリエディタを使用して、クエリがどのように実行されるかについてのグラフィック表現を表示できます。エディタでクエリを入力して [Explain] オプションを選択すると、Athena はクエリで EXPLAIN SQL ステートメントを使用して、分散実行プランと論理実行プランの 2 つの対応するグラフを作成します。クエリの分析、トラブルシューティング、および効率の向上のために、これらのグラフを使用できます。

クエリの実行プランを表示するには

1. Athena クエリエディタにクエリを入力し、[Explain] を選択します。



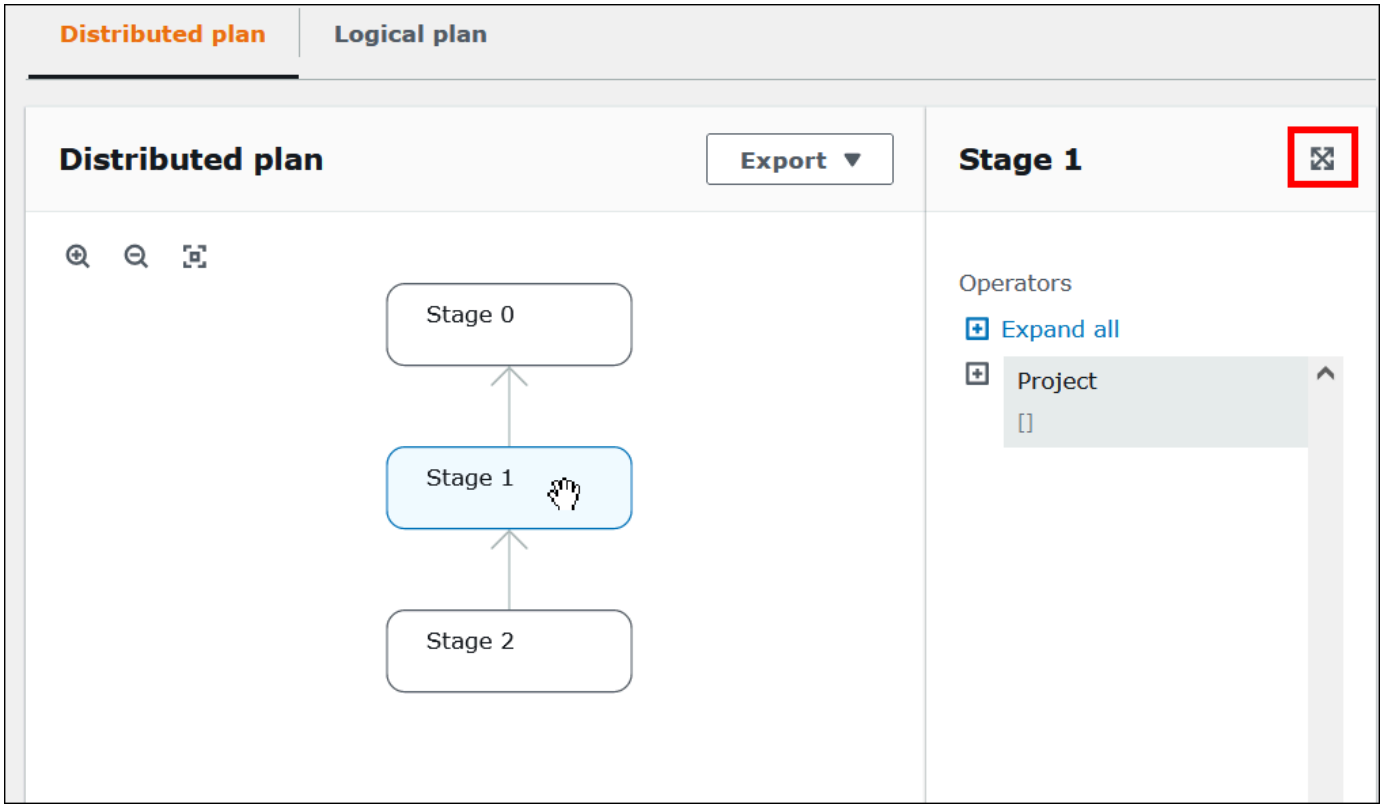
   [Distributed plan] (分散プラン) タブには、分散環境でのクエリの実行プランが表示されます。分散プランには、処理フラグメントまたはステージがあります。各ステージには 0 ベースのインデックス番号があり、1 つまたは複数のノードによって処理されます。データはノード間で交換できます。



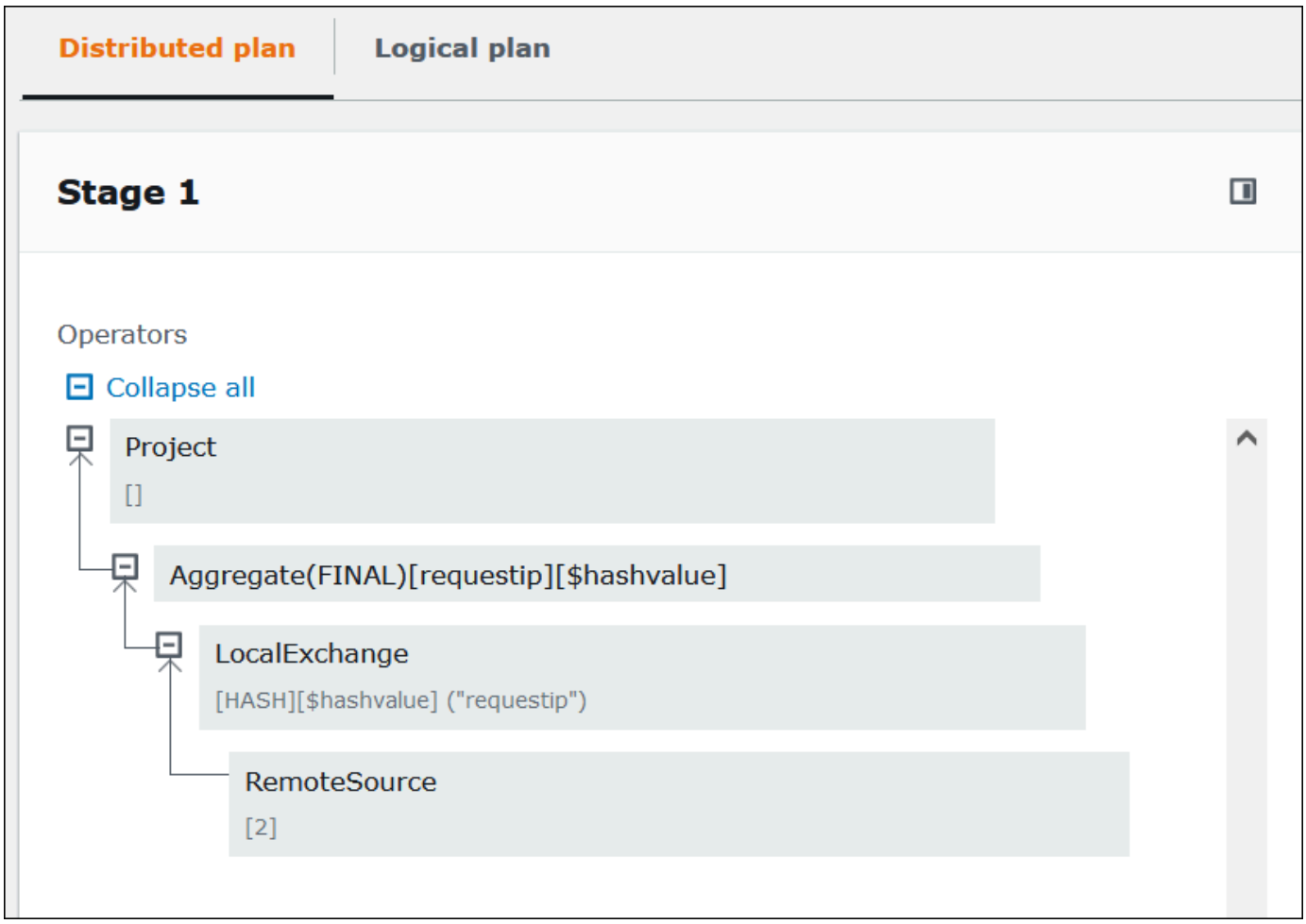
2. グラフを操作するには、次のオプションを使用します。

   - ズームインまたはズームアウトするには、マウスでスクロールするか、拡大アイコンを使用します。
   - 画面に合わせてグラフを調整するには、[Zoom to fit] (ズームして合わせる) アイコンを選択します。
   - グラフを移動するには、マウスポインタをドラッグします。

3. ステージの詳細を表示するには、ステージを選択します。

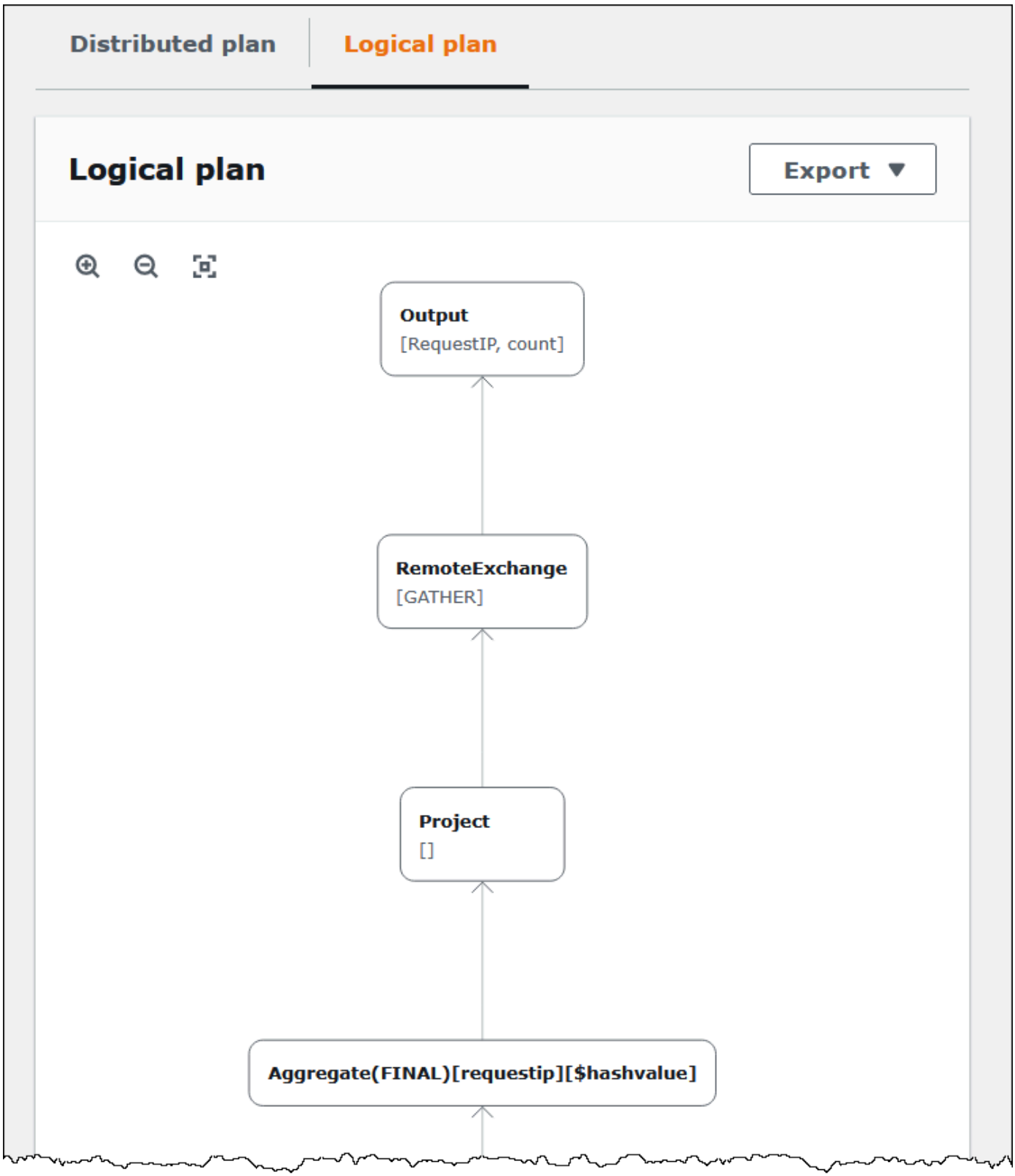


4. ステージの詳細を全幅で表示するには、詳細ペインの右上にある拡大アイコンを選択します。
5. 詳細を表示するには、オペレーターツリーで 1 つ以上の項目を展開します。分散プランフラグメントの詳細については、「EXPLAIN ステートメントの出力タイプ」を参照してください。



> **Important**
>
> 現在、一部のパーティションフィルターは、Athena がクエリに適用しても、ネストされたオペレーターツリーグラフに表示されない場合があります。このようなフィルターの効果を検証するには、クエリで EXPLAIN または EXPLAIN ANALYZE を実行し、結果を表示します。

6. [Logical plan] (論理プラン) タブを選択します。グラフには、クエリを実行するための論理プランが表示されます。運用条件については、「Athena EXPLAIN ステートメントの結果について」を参照してください。



7. プランを SVG または PNG 画像として、または JSON テキストとしてエクスポートするには、[Export] (エクスポート) を選択します。

追加リソース

詳細については、以下のリソースを参照してください。

Athena での EXPLAIN および EXPLAIN ANALYZE の使用

Athena EXPLAIN ステートメントの結果について

完了したクエリの統計と実行の詳細の表示

クエリ結果、最近のクエリ、および出力ファイルの使用

Amazon Athena は、Amazon S3 で指定できるクエリ結果の場所に、実行される各クエリのクエリ結果とメタデータ情報を自動的に保存します。必要に応じて、この場所にあるファイルにアクセスして操作できます。Athena コンソールからクエリ結果ファイルを直接ダウンロードすることもできます。

Amazon S3 にクエリ結果の場所を初めてセットアップするときは、「Athena コンソールを使用したクエリ結果の場所の指定」を参照してください。

出力ファイルは、実行されるすべてのクエリに対して自動的に保存されます。Athena コンソールを使用してクエリ出力ファイルにアクセスして表示するには、IAM プリンシパル (ユーザーとロール) に、クエリ結果ロケーションに対する Amazon S3 の GetObject アクションの許可と、Athenaの GetQueryResults アクションの許可が必要です。クエリの結果の場所は暗号化できます。場所が暗号化されている場合、ユーザーには、クエリ結果の場所を暗号化および復号するための適切なキーアクセス許可が必要です。

> **Important**
>
> クエリ結果の場所に対する Amazon S3 の `GetObject` アクションの許可を持つ IAM プリンシパルは、Athena の `GetQueryResults` アクションの許可が拒否された場合でも、Amazon S3 からクエリ結果を取得できます。

クエリ結果の場所の指定

Athena が使用するクエリ結果の場所は、ワークグループの設定とクライアント側の設定の組み合わせによって決定されます。クライアント側の設定は、クエリの実行方法に基づいています。

- Athena コンソールを使用してクエリを実行する場合は、ナビゲーションバーの [Settings] (設定) で入力した [Query result location] (クエリ結果の場所) がクライアント側の設定を決定します。

- Athena API を使用してクエリを実行する場合は、StartQueryExecution アクションの `OutputLocation` パラメータがクライアント側の設定を決定します。
- ODBC または JDBC ドライバーを使用してクエリを実行する場合、接続 URL で指定された `S3OutputLocation` プロパティによってクライアント側の設定が決まります。

> **Important**
>
> API または ODBC または JDBC ドライバーを使用してクエリを実行する場合、コンソールの設定は適用されません。

各ワークグループ設定には、[Override client-side settings (クライアント側設定の上書き)] オプションがあり、有効にできます。このオプションを有効にすると、そのワークグループに関連付けられている IAM プリンシパルがクエリを実行するときに、適切なクライアント側の設定よりもワークグループの設定が優先されます。

Athena コンソールを使用したクエリ結果の場所の指定

クエリを実行する前に、Amazon S3 のクエリ結果バケットの場所を指定しておく、または指定されたバケットがあり、その設定がクライアント設定を上書きするワークグループを使用する必要があります。

Athena コンソールを使用してクライアント側の設定のクエリ結果の場所を指定する

1. クエリ結果の場所を指定するワークグループに切り替えます。デフォルトのワークグループの名前は primary です。
2. ナビゲーションバーから [Settings] (設定) を選択します。
3. ナビゲーションバーで [Manage] (管理) を選択します。
4. [Manage settings] (設定の管理) で、以下のいずれかを実行します。
   - [Location of query result] (クエリ結果の場所) ボックスで、クエリ結果のために Amazon S3 に作成したバケットへのパスを入力します。パスの先頭に `s3://` を付けます。
   - [Browse S3] (S3 をブラウズ) をクリックし、現在のリージョンで作成した Amazon S3 バケットを選択した上で、[Choose] (選択) をクリックします。

> **Note**
>
> ワークグループのすべてのユーザーに対してクエリ結果の場所を指定するワークグループを使用している場合、クエリ結果の場所を変更するオプションは使用できません。

5. (オプション) [View lifecycle configuration] (ライフサイクル設定を表示) を選択すると、クエリ結果バケットの [Amazon S3 lifecycle rules] (Amazon S3 ライフサイクルルール) を表示および設定できます。作成する Amazon S3 ライフサイクルルールは、有効期限ルールまたは移行ルールのいずれでも可能です。有効期限ルールでは、一定の時間が経過するとクエリ結果は自動的に削除されます。移行ルールは別の Amazon S3 ストレージ階層に移動します。詳細については、Amazon Simple Storage Service ユーザーガイドの「バケットのライフサイクル設定の指定」を参照してください。
6. (オプション) [Expected bucket owner] (想定されるバケット所有者) に、出力ロケーションバケットの所有者になると想定されている AWS アカウント の ID を入力します。これは追加のセキュリティ対策です。バケット所有者のアカウント ID がここで指定した ID と一致しない場合、バケットに出力できません。詳細については、「Amazon S3 ユーザーガイド」の「バケット所有者条件によるバケット所有者の確認」を参照してください。

   > **Note**
   >
   > 予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。これは、外部 Simple Storage Service (Amazon S3) バケット内のデータソースの場所、`CTAS` や `INSERT INTO` の書き込み先のテーブルの場所、`UNLOAD` ステートメントの出力場所、串刺検索のバケットのスピルオペレーション、別のアカウントのテーブルに対して実行される `SELECT` クエリなど、他の Amazon S3 ロケーションには適用されません。

7. (オプション) Simple Storage Service (Amazon S3) に保存されているクエリ結果を暗号化するには、[Encrypt query results] (クエリ結果の暗号化) を選択します。Athena の暗号化に関する詳細については、保管中の暗号化 を参照してください。
8. (オプション) クエリ結果バケットの ACL が有効になっているときに、バケット所有者にクエリ結果に対するフルコントロールアクセスを割り当てるには、[Assign bucket owner full control over query results] (クエリ結果に対する完全なコントロールをバケット所有者に付与する) を選択します。たとえば、クエリ結果の場所を別のアカウントが所有している場合に、クエリ結果の所有権とフルコントロールを他のアカウントに付与することができます。詳細について

は、Simple Storage Service (Amazon S3) ユーザーガイドのオブジェクトの所有権のコントロールとバケットに対する ACL の無効化を参照してください。

9. [Save] を選択します。

以前に作成されたデフォルトの場所

これまで Athena では、クエリ結果の場所の値を指定せずにクエリを実行し、クエリ結果の場所の設定がワークグループによって上書きされなかった場合は、Athena がデフォルトの場所を作成していました。デフォルトの場所は `aws-athena-query-results-MyAcctID-MyRegion` でした。*MyAcctID* はクエリを実行した IAM プリンシパルの Amazon Web Services アカウント ID で、*MyRegion* はクエリが実行されたリージョン (`us-west-1` など) です。

今後は、以前に Athena を使用したことがないアカウントがあるリージョンで Athena クエリを実行する前に、クエリ結果の場所を指定するか、クエリ結果の場所の設定を上書きするワークグループを使用しなければならなくなります。Athena はデフォルトのクエリ結果の場所を作成しなくなりましたが、以前に作成されたデフォルトの `aws-athena-query-results-MyAcctID-MyRegion` の場所は引き続き有効で、使用を継続できます。

ワークグループを使用したクエリ結果の場所の指定

AWS Management Console、AWS CLI、または Athena API を使用して、ワークグループ設定でクエリ結果の場所を指定します。

AWS CLI を使用する場合、`aws athena create-work-group` または `aws athena update-work-group` コマンドを実行するときに `--configuration` オプションの `OutputLocation` パラメータを使用してクエリ結果の場所を指定します。

Athena コンソールを使用してワークグループのクエリ結果の場所を指定する

1. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

2. ナビゲーションペインで、[Workgroups] (ワークグループ) を選択します。
3. ワークグループのリストで、編集するワークグループのリンクを選択します。
4. [編集] を選択します。
5. クエリ結果の場所と暗号化については、次のいずれかを実行します。

   - [Location of query result] (クエリ結果の場所) ボックスに、クエリ結果のために Amazon S3 のバケットへのパスを入力します。パスの先頭に `s3://` を付けます。
   - [Browse S3] (S3 のブラウズ) で、使用する現在のリージョン用の Amazon S3 バケットを選択し、[Choose] (選択) を選択します。

6. (オプション) [Expected bucket owner] (想定されるバケット所有者) に、出力ロケーションバケットの所有者になると想定されている AWS アカウント の ID を入力します。これは追加のセキュリティ対策です。バケット所有者のアカウント ID がここで指定した ID と一致しない場合、バケットに出力できません。詳細については、「Amazon S3 ユーザーガイド」の「バケット所有者条件によるバケット所有者の確認」を参照してください。

   > **Note**
   >
   > 予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。これは、外部 Simple Storage Service (Amazon S3) バケット内のデータソースの場所、`CTAS` や `INSERT INTO` の書き込み先のテーブルの場所、`UNLOAD` ステートメントの出力場所、串刺検索のバケットのスピルオペレーション、別のアカウントのテーブルに対して実行される `SELECT` クエリなど、他の Amazon S3 ロケーションには適用されません。

7. (オプション) Simple Storage Service (Amazon S3) に保存されているクエリ結果を暗号化するには、[Encrypt query results] (クエリ結果の暗号化) を選択します。Athena の暗号化に関する詳細については、保管中の暗号化 を参照してください。
8. (オプション) クエリ結果バケットの ACL が有効になっているときに、バケット所有者にクエリ結果に対するフルコントロールアクセスを割り当てるには、[Assign bucket owner full control over query results] (クエリ結果に対する完全なコントロールをバケット所有者に付与する) を選択します。たとえば、クエリ結果の場所を別のアカウントが所有している場合に、クエリ結果の所有権とフルコントロールを他のアカウントに付与することができます。

   バケットの S3 オブジェクトの所有権の設定が [Bucket owner preferred] (バケット所有者推奨) となっている場合、バケット所有者は、このワークグループから書き込まれたすべてのクエリ結果オブジェクトも所有します。たとえば、外部アカウントのワークグループでこのオプションが有効になっており、そのクエリ結果の場所がアカウントの Simple Storage Service (Amazon S3) バケットに設定されている場合に、このバケットの S3 オブジェクトの所有権が [Bucket owner preferred] (バケット所有者推奨) と設定されていると、外部ワークグループのクエリ結果を所有し、フルコントロールアクセスを持つことになります。

   クエリ結果バケットの S3 オブジェクトの所有権設定が [Bucket owner enforced] (バケット所有者の強制) となっているときにこのプションを選択しても、効果はありません。詳細については、Simple Storage Service (Amazon S3) ユーザーガイドのオブジェクトの所有権のコントロールとバケットに対する ACL の無効化を参照してください。
9. 指定したクエリ結果の場所を使用するようにワークグループのすべてのユーザーに要求する場合は、[Settings] (設定) セクションまで下にスクロールして、[Override client-side settings] (クライアント側設定の上書き) を選択します。
10. [Save changes] (変更の保存) を選択します。

Athena コンソールを使用したクエリ結果ファイルのダウンロード

クエリを実行した直後に、クエリペインからクエリ結果の CSV ファイルをダウンロードできます。最近のクエリ結果も、[Recent queries] (最近のクエリ) タブからダウンロードできます。

> **Note**
>
> Athena クエリ結果ファイルは、個々のユーザーによって設定できる情報を含むデータファイルです。このデータの読み込みと分析を行うプログラムの一部は、データの一部をコマンドと解釈する可能性があります (CSV インジェクション)。このため、クエリ結果の CSV データをスプレッドシートプログラムにインポートすると、そのプログラムから、セキュリティ

上の問題について警告を受ける場合があります。システムを安全に保つには、ダウンロードされたクエリ結果からリンクまたはマクロを無効にするように常に選択する必要があります。

クエリを実行してクエリ結果をダウンロードするには

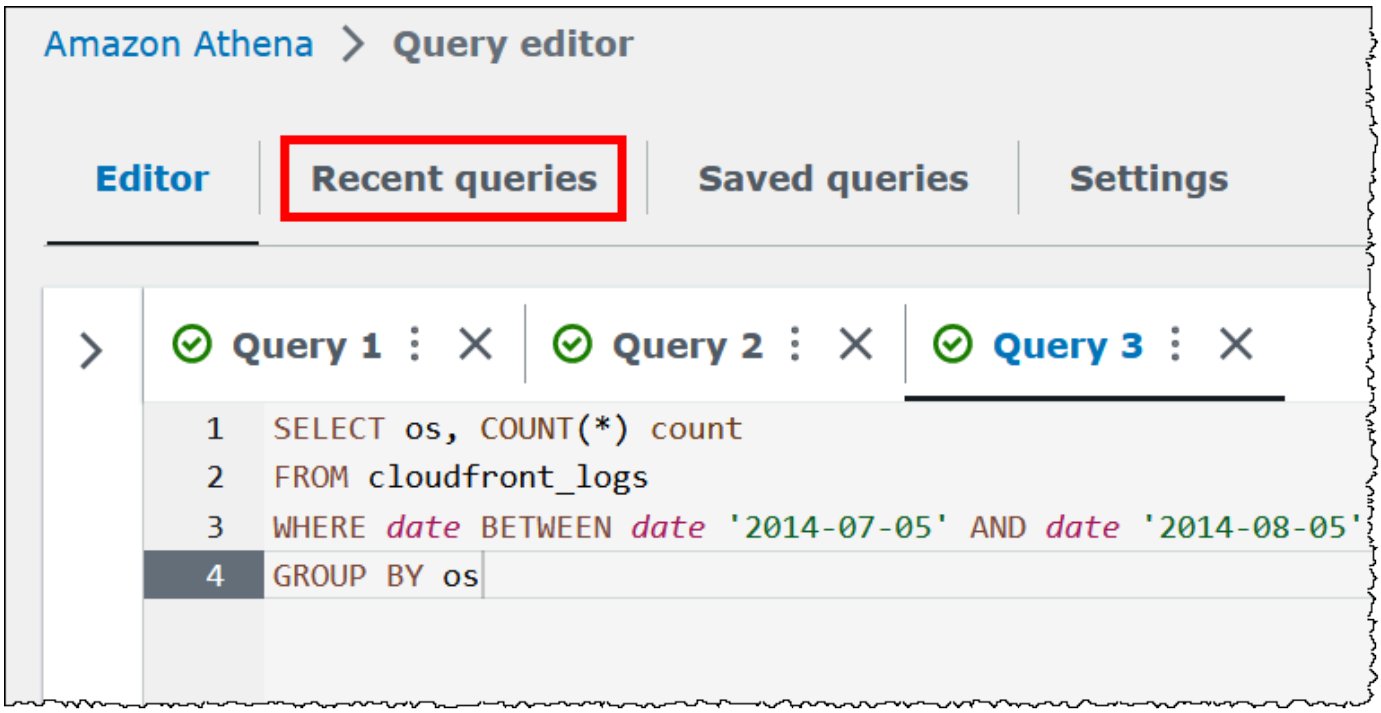
1. クエリエディタにクエリを入力し、[Run] (実行) をクリックします。

   クエリの実行が終了すると、[Results (結果)] ペインにクエリ結果が表示されます。

2. クエリ結果の CSV ファイルをダウンロードするには、クエリ結果ペインの上にある [Download results] (結果のダウンロード) を選択します。ブラウザとブラウザの設定によっては、ダウンロードの確認が必要になる場合があります。

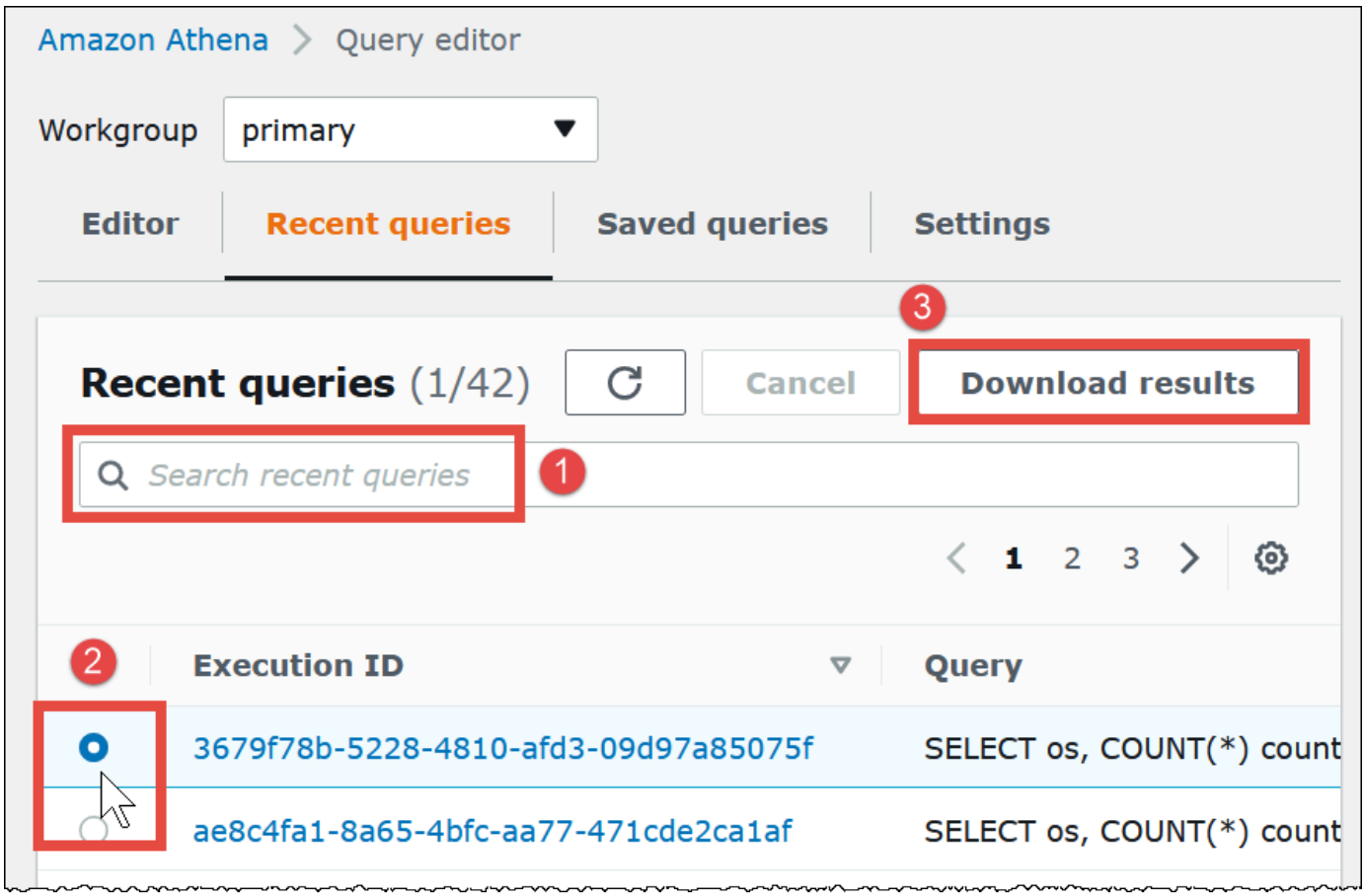
以前のクエリのクエリ結果ファイルをダウンロードするには

1. [Recent queries] (最近のクエリ) を選択します。



2. 検索ボックスを使用してクエリを見つけ、クエリを選択してから、[Download results] (結果のダウンロード) を選択します。

> **Note**
>
> [Download results] (結果のダウンロード) オプションを使用して、手動で削除されたクエリ結果を取得したり、Amazon S3 ライフサイクルルールによって削除または別の場所に移動されたクエリ結果を取得したりすることはできません。

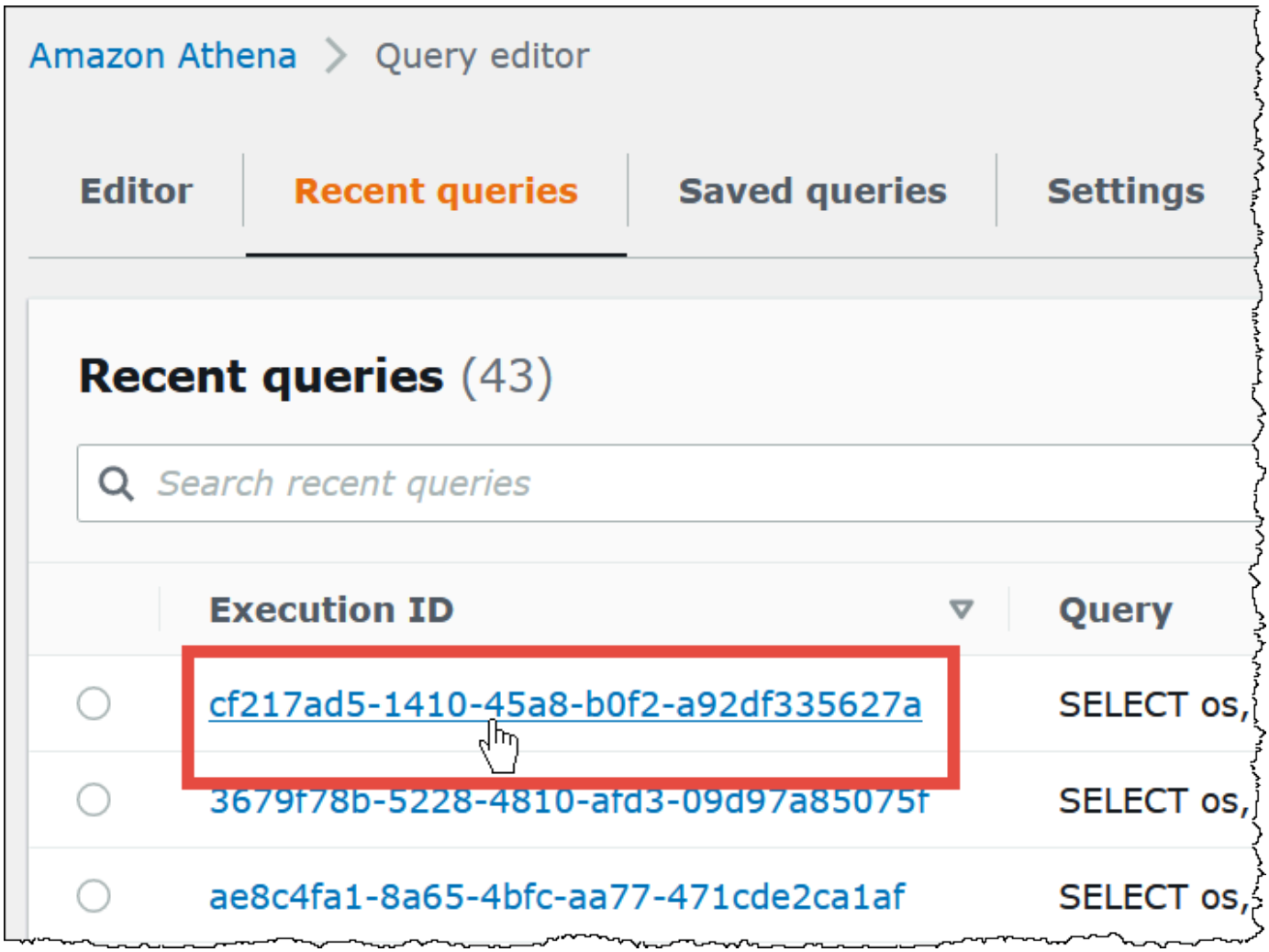


最近のクエリの表示

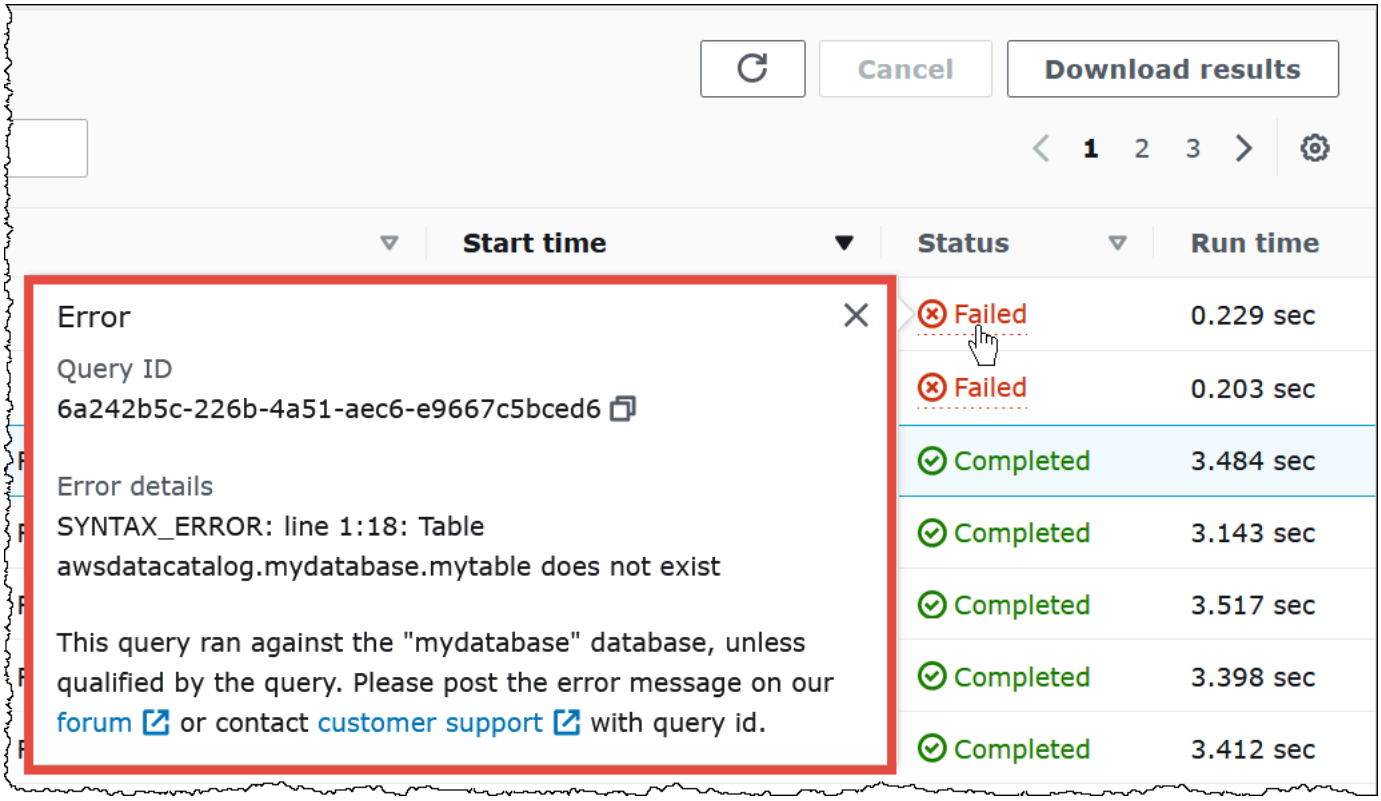
Athena コンソールを使用して、成功したクエリと失敗したクエリの表示、および失敗したクエリのエラー詳細の表示を行うことができます。Athena では、クエリ履歴が 45 日間保持されます。

Athena コンソールで最近のクエリを表示するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. [Recent queries] (最近のクエリ) を選択します。[Recent queries] (最近のクエリ) タブには、実行された各クエリに関する情報が表示されます。
3. クエリエディタでクエリステートメントを開くには、クエリの実行 ID を選択します。



4. 失敗したクエリの詳細を表示するには、クエリの [Failed] (失敗) リンクを選択します。

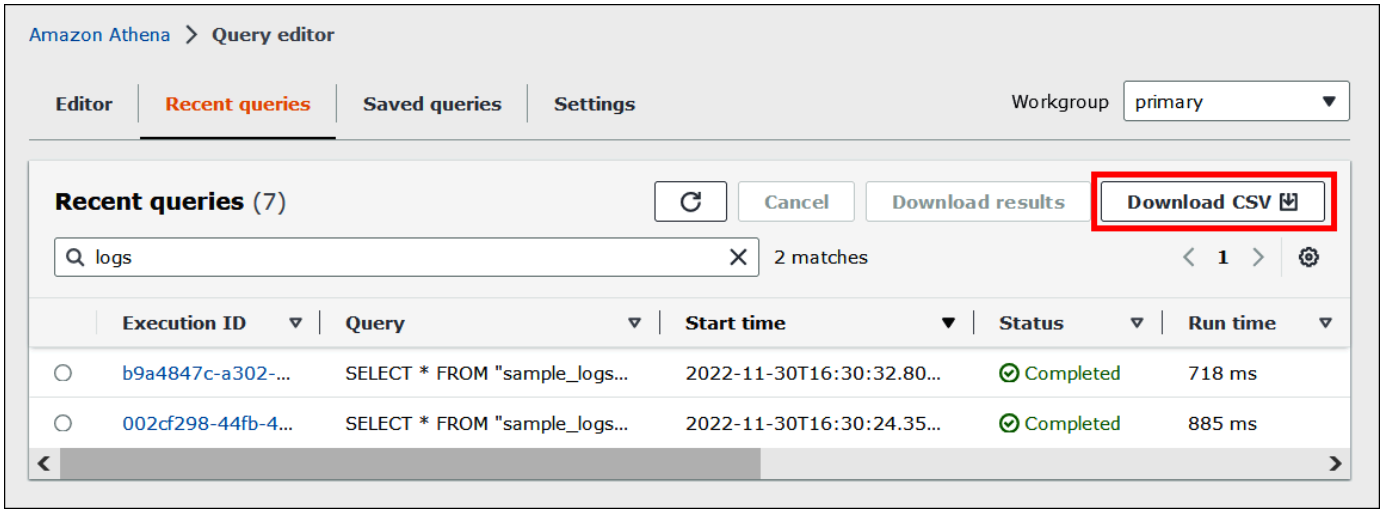


最近の複数のクエリを CSV ファイルにダウンロードする

Athena コンソールの [Recent queries] (最近のクエリ) タブを使用すると、1 つ以上の最近のクエリを CSV ファイルにエクスポートして、表形式で表示できます。ダウンロードしたファイルには、クエリ結果ではなく、SQL クエリ文字列自体とクエリに関するその他の情報が含まれています。エクスポートされるフィールドには、実行 ID、クエリ文字列のコンテンツ、クエリの開始時間、ステータス、実行時間、スキャンされたデータ量、使用されたクエリエンジンのバージョン、および暗号化方法が含まれます。最近のクエリを最大 500 個、または検索ボックスに入力した条件を使用してフィルター処理をした最大 500 個のクエリをエクスポートできます。

1 つ以上の最近のクエリを CSV ファイルにエクスポートするには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. [Recent queries] (最近のクエリ) を選択します。
3. (オプション) 検索ボックスを使用して、ダウンロードする最近のクエリを絞り込みます。
4. [Download CSV] を選択します。



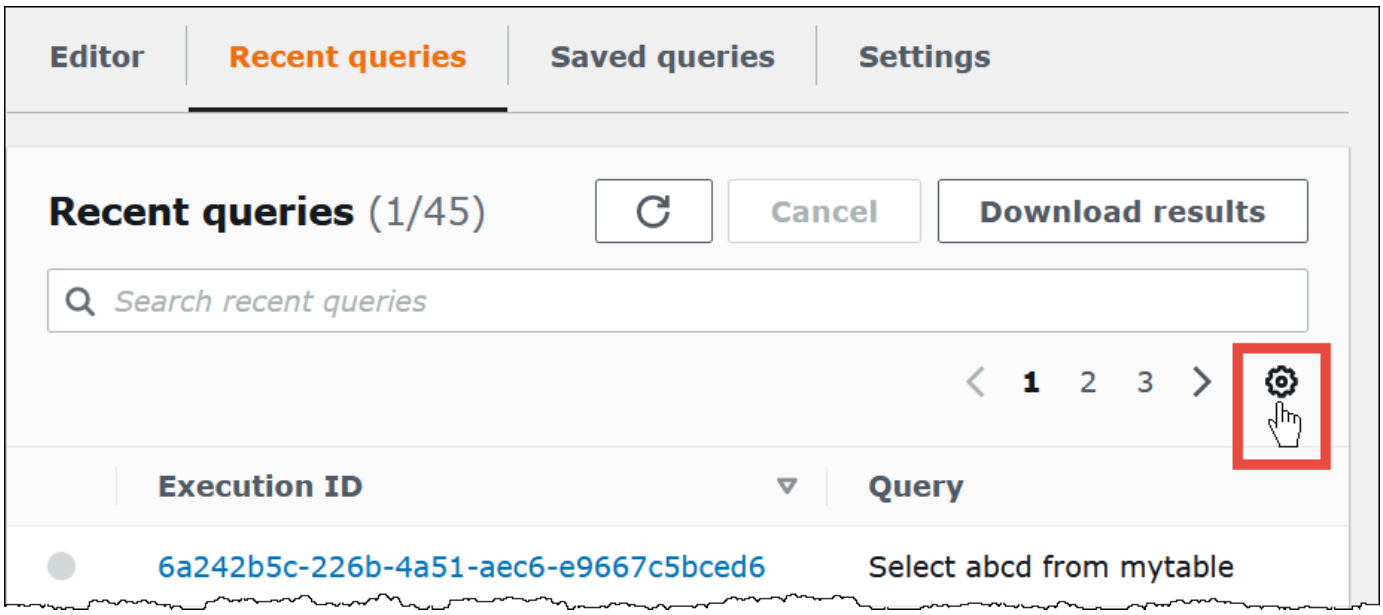
5. ファイル保存のプロンプトで、[Save] (保存) を選択します。デフォルトのファイル名は、タイムスタンプによって続く `Recent Queries` です (例: `Recent Queries 2022-12-05T16 04 27.352-08 00.csv`)

最近のクエリ表示オプションの設定

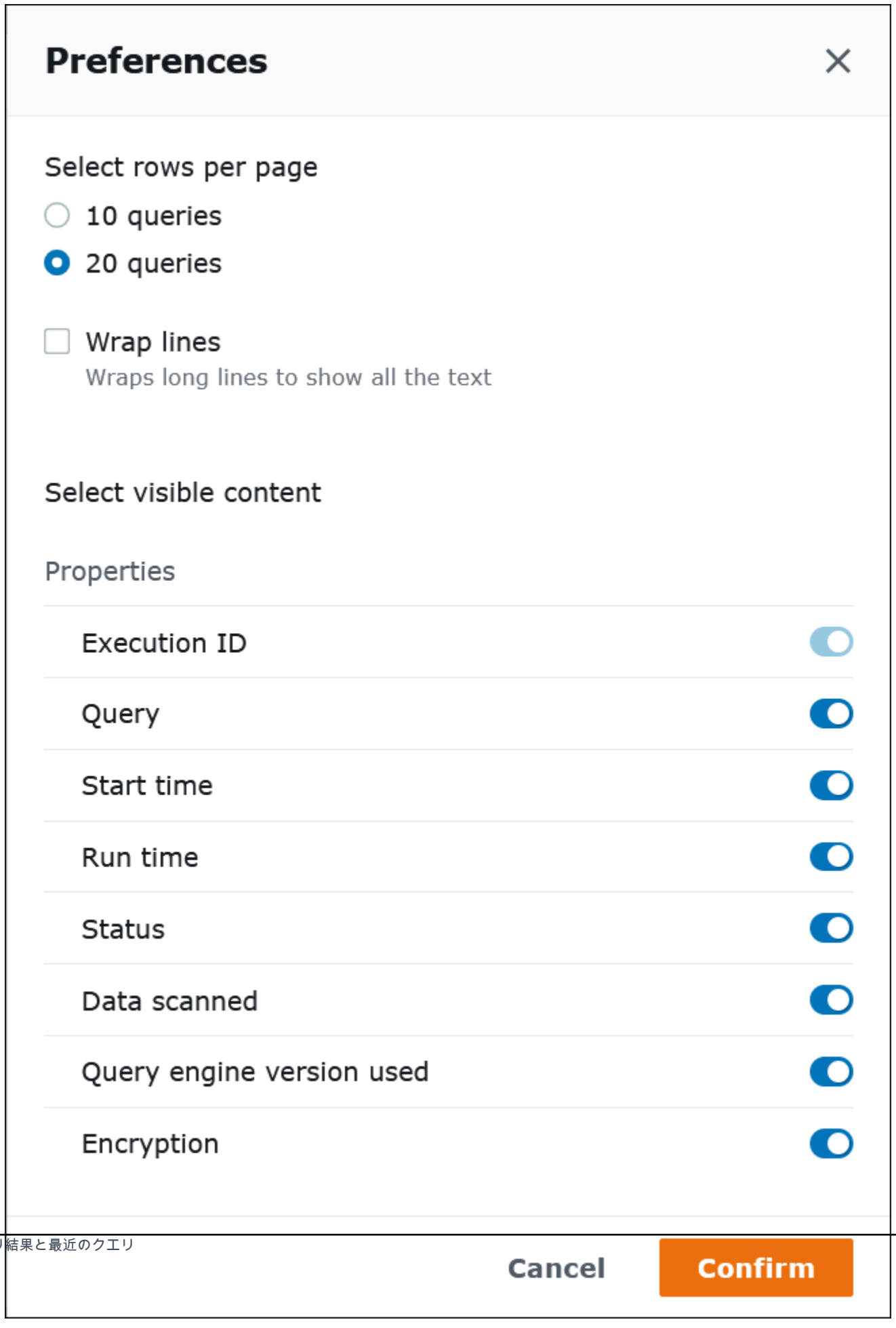
表示する列やテキストの折り返しなど、[Recent queries] (最近のクエリ) タブのオプションを設定できます。

[Recent queries] (最近のクエリ) タブのオプションを設定するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. [Recent queries] (最近のクエリ) を選択します。
3. オプションボタン (歯車のアイコン) を選択します。



4. [Preferences] (設定) ダイアログボックスで、ページあたりの行数、行折り返しの動作、および表示する列を選択します。


Preferences
Select rows per page
10 queries
20 queries
Wrap lines
Wraps long lines to show all the text
Select visible content
Properties
Execution ID
Query
Start time
Run time
Status
Data scanned
Query engine version used
Encryption
Cancel
Confirm

5. [確認] を選択します。

クエリ履歴を 45 日より長く保持

クエリ履歴を 45 日より長く保持する場合は、クエリ履歴を取得して、Amazon S3 などのデータストアに履歴を保存できます。このプロセスを自動化するには、Athena および Amazon S3 の API アクションと CLI コマンドを使用できます。以下の手順は、これらのステップをまとめたものです。

クエリ履歴をプログラムで取得して保存するには

1. Athena の ListQueryExecutions API アクション、または list-query-executions CLI コマンドを使用して、クエリ ID を取得します。
2. Athena の GetQueryExecution API アクション、または get-query-execution CLI コマンドを使用して、ID に基づいた各クエリに関する情報を取得します。
3. Simple Storage Service (Amazon S3) の PutObject API アクション、または put-object CLI コマンドを使用して、その情報を Amazon S3 に保存します。

Simple Storage Service (Amazon S3) でのクエリ出力ファイルの検索

ワークグループでクエリが発生し、そのワークグループの設定がクライアント側の設定よりも優先される場合でない限り、クエリ出力ファイルは次のパスパターンで Amazon S3 サブフォルダに保存されます。ワークグループの設定がクライアント側の設定よりも優先される場合、クエリはワークグループによって指定された結果パスを使用します。

```
QueryResultsLocationInS3/[QueryName|Unsaved/yyyy/mm/dd/]
```

- *QueryResultsLocationInS3* は、ワークグループ設定またはクライアント側設定で指定されたクエリ結果の場所です。詳細については、このドキュメントで後述する「the section called “クエリ結果の場所の指定”」を参照してください。
- 次のサブフォルダは、ワークグループ設定が結果パスよりも優先されていないコンソールから実行されるクエリに対してのみ作成されます。AWS CLI から実行されるクエリまたは Athena API を使用して実行されるクエリは、*QueryResultsLocationInS3* に直接保存されます。
  - *QueryName* は結果を保存するクエリの名前です。クエリが実行されたものの、保存されなかった場合は、Unsaved が使用されます。
  - *yyyy/mm/dd* は、クエリが実行された日付です。

CREATE TABLE AS SELECT クエリに関連付けられたファイルは、上記のパターンの tables サブフォルダに保存されます。

クエリ出力ファイルの識別

ファイルは、クエリの名前、クエリ ID、およびクエリが実行された日付に基づいて、Amazon S3 のクエリ結果の場所に保存されます。各クエリのファイルは *QueryID* を使用して命名されます。QueryID は、クエリの実行時に Athena が各クエリに割り当てる一意の識別子です。

次のファイルタイプが保存されます。

ファイルタイプ	ファイル命名パターン	説明
クエリ結果ファイル	*QueryID*.csv *QueryID*.txt	DML クエリ結果ファイルはカンマ区切り値 (CSV) 形式で保存されます。 DDL クエリ結果は、プレーンテキストファイルとして保存されます。 結果ファイルは、コンソールの使用時には [Results] (結果) ペインから、またはクエリの [History] (履歴) からダウンロードできます。詳細については、「Athena コンソールを使用したクエリ結果ファイルのダウンロード」を参照してください。
クエリメタデータファイル	*QueryID*.csv.metadata *QueryID*.txt.metadata	DML および DDL クエリメタデータファイルはバイナリ形式で保存され、人間が読めるものではありません。ファイル拡張子は、関連するクエリ結果ファイルに対応しています。Athena は、GetQueryResults アクションを使用

ファイルタイプ	ファイル命名パターン	説明
		してクエリ結果を読み込むときに、メタデータを使用します。これらのファイルは削除できますが、クエリに関する重要な情報が失われるため、お勧めしません。
データマニフェストファイル	*QueryID*-manifest.csv	データマニフェストファイルは、INSERT INTO クエリの実行時に Athena が Simple Storage Service (Amazon S3) データソースの場所に作成するファイルを追跡するために生成されます。クエリが失敗した場合、マニフェストはクエリが書き込むことを意図したファイルも追跡します。マニフェストは、失敗したクエリの結果として孤立したファイルを識別するのに役立ちます。

AWS CLI を使用してクエリ出力の場所とファイルを特定

AWS CLI を使用してクエリ出力の場所と結果ファイルを特定するには、次の例のように、aws athena get-query-execution コマンドを実行します。*abc1234d-5efg-67hi-jklm-89n0op12qr34* をクエリ ID に置き換えます。

```
aws athena get-query-execution --query-execution-id abc1234d-5efg-67hi-
jklm-89n0op12qr34
```

このコマンドにより、以下のような出力が返されます。各出力パラメータの説明については、AWS CLI コマンドリファレンスの「get-query-execution」を参照してください。

```
{
    "QueryExecution": {
```

```
        "Status": {
            "SubmissionDateTime": 1565649050.175,
            "State": "SUCCEEDED",
            "CompletionDateTime": 1565649056.6229999
        },
        "Statistics": {
            "DataScannedInBytes": 5944497,
            "DataManifestLocation": "s3://DOC-EXAMPLE-BUCKET/athena-query-
results-123456789012-us-west-1/MyInsertQuery/2019/08/12/abc1234d-5efg-67hi-
jklm-89n0op12qr34-manifest.csv",
            "EngineExecutionTimeInMillis": 5209
        },
        "ResultConfiguration": {
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_S3"
            },
            "OutputLocation": "s3://DOC-EXAMPLE-BUCKET/athena-query-
results-123456789012-us-west-1/MyInsertQuery/2019/08/12/abc1234d-5efg-67hi-
jklm-89n0op12qr34"
        },
        "QueryExecutionId": "abc1234d-5efg-67hi-jklm-89n0op12qr34",
        "QueryExecutionContext": {},
        "Query": "INSERT INTO mydb.elb_log_backup SELECT * FROM mydb.elb_logs LIMIT
 100",
        "StatementType": "DML",
        "WorkGroup": "primary"
    }
}
```

クエリの結果を再利用する

Athena でクエリを再実行する場合、オプションで最後に保存されたクエリ結果を再利用することを選択できます。このオプションにより、パフォーマンスが向上し、スキャンされるバイト数によりコストが削減されます。クエリの結果を再利用することは、たとえば、特定の時間枠内で結果に変化がないことがわかっている場合に役立ちます。クエリの結果を再利用できる最大有効期間を指定できます。Athena では、指定した経過日数を超えない限り、保存された結果を使用します。詳細については、AWS Big Data Blog の「Reduce cost and improve query performance with Amazon Athena」を参照してください。

> **Note**
>
> クエリ結果の再利用機能には、Athena エンジンのバージョン 3 が必要です。エンジンバージョンの変更の詳細については、「Athena エンジンバージョンの変更」を参照してください。

主な特徴

- クエリの結果の再利用はクエリごとのオプトイン機能です。クエリの結果の再利用はクエリごとに有効にできます。
- クエリの結果を再利用できる最大期間は、分数、時間数、または日数で指定できます。指定可能な最大期間は、使用する時間単位に関係なく、同様に 7 日間です。のデフォルトは 60 分です。
- クエリの結果の再利用を有効にすると、Athena は同じワークグループ内で以前に実行されたクエリを検索します。Athena では対応する保存済みクエリ結果を見つけた場合、クエリを再実行せず、以前の結果の場所を指すか、そこからデータを取得します。

- 結果の再利用オプションを有効にするクエリにおいて、Athena は次の条件がすべて満たされる場合にのみ、ワークグループフォルダに保存された最後のクエリ結果を再利用します。
  - クエリ文字列は完全に一致します。
  - データベースとカタログ名は一致します。
  - 以前の結果は、指定された最大経過時間を超えていないか、または最大経過時間が指定されていない場合は 60 分を超えていません。
  - Athena は、現在の実行とまったく同じ結果の設定の実行のみを再利用します。
  - クエリで参照されているすべてのテーブルにアクセスできます。
  - 前回の結果が保存されている S3 ファイルの場所にアクセスできます。

これらの条件のいずれかが満たされない場合、Athena はキャッシュされた結果を使用せずにクエリを実行します。

考慮事項と制約事項

クエリ結果の再利用機能を使用する場合は、次の点に注意してください。

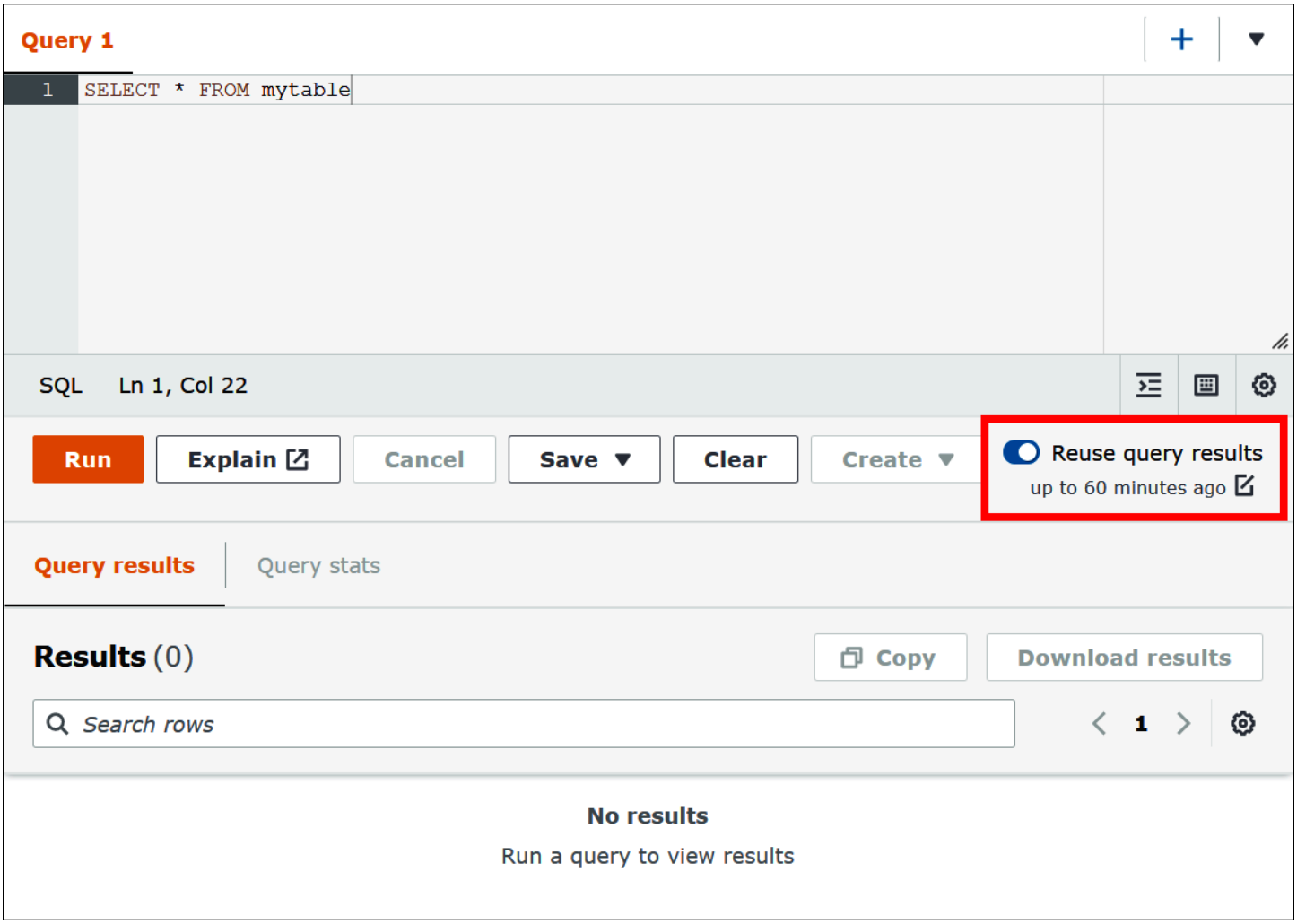
- Athena は同じワークグループ内でのみクエリ結果を再利用します。

- クエリ結果の再利用機能は、ワークグループの設定を優先します。クエリの結果の設定をオーバーライドすると、この機能は無効になります。
- AWS Glue で登録された Apache Hive テーブル、Apache Hudi テーブル、Apache Iceberg テーブル、および Linux Foundation Delta Lake テーブルがサポートされています。外部 Hive メタストアはサポートされていません。
- フェデレーションカタログ、または外部の Hive メタストアを参照するクエリはサポートされていません。
- Lake Formation の管理対象テーブルでは、クエリ結果の再利用はサポートされていません。
- テーブルソースの Amazon S3 の場所が Lake Formation のデータの場所として登録されている場合、クエリ結果の再利用はサポートされません。
- 行と列のアクセス許可を持つテーブルはサポートされていません。
- きめ細かいアクセス制御 (列や行のフィルタリングなど) があるテーブルはサポートされていません。
- サポートされていないテーブルを参照するクエリでは、クエリ結果の再利用は対象外です。
- 以前に生成された出力ファイルを再利用するには、Athena では Amazon S3 の読み込みアクセス許可が必要です。
- クエリ結果の再利用機能は、以前の結果の内容が変更されていないことを前提としています。Athena では以前の結果を使用する前にその結果の整合性をチェックしません。
- 前回の実行でのクエリ結果が削除されているか、Amazon S3 の別の場所に移動されている場合、同じクエリを次に実行してもクエリ結果は再利用されません。
- 古い結果が返される可能性があります。Athena では指定した最大再利用期間に達するまで、ソースデータの変更を確認しません。
- 複数の結果を再利用できる場合、Athena は最新の結果を使用します。
- `rand()` または `shuffle()` のような確定的ではない演算子または関数を使用するクエリは、キャッシュされた結果を使用しません。たとえば、`ORDER BY` を持たない `LIMIT` は非決定論的でキャッシュされませんが、`ORDER BY` を持たない `LIMIT` は決定論的でキャッシュされます。
- クエリ結果の再利用は、Athena コンソール、Athena API、および JDBC ドライバーでサポートされています。現在、クエリ結果を再利用するための ODBC ドライバーサポートは Windows でのみ利用可能です。
- JDBC でクエリ結果の再利用機能を使用するには、最低限必要なドライバーバージョンは 2.0.34.1000 です。ODBC に最低限必要なドライバー バージョンは 1.1.19.1002 です。ドライバーのダウンロード情報については、「ODBC および JDBC ドライバーを使用した Amazon Athena への接続」を参照してください。

- 複数のデータカタログを使用するクエリでは、クエリ結果の再利用はサポートされません。
- 20 を超えるテーブルが含まれるクエリでは、クエリ結果の再利用はサポートされません。

Athena コンソールでのクエリ結果の再利用

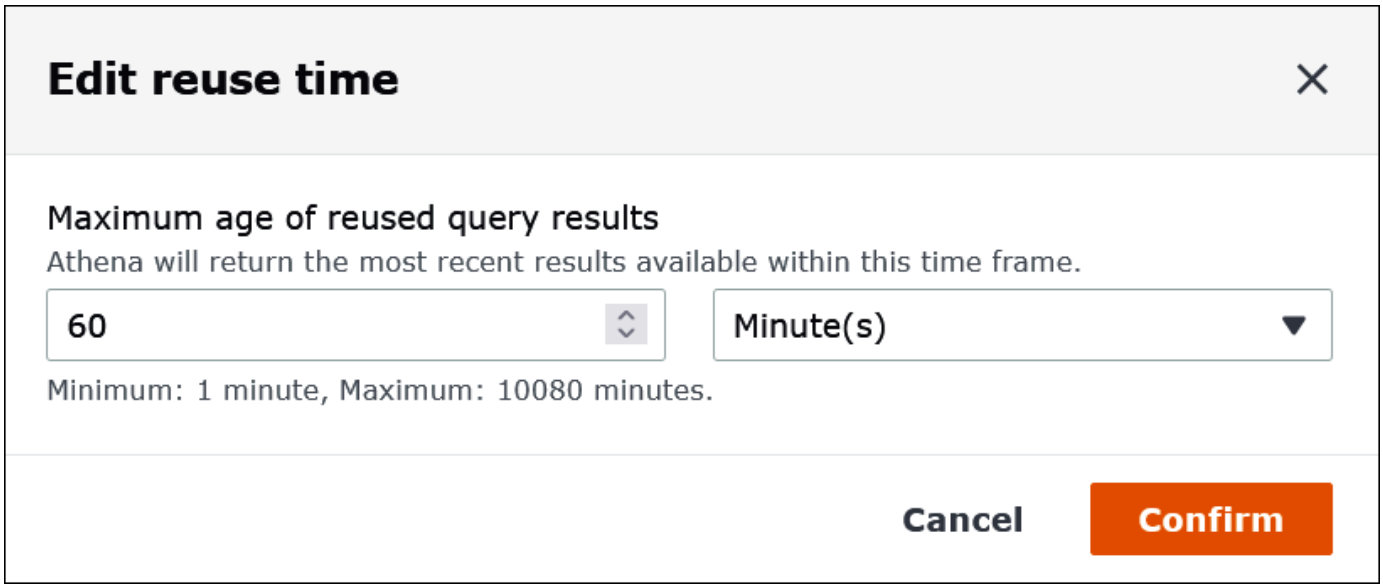
この機能を使用するには、Athena クエリエディタの [Reuse query results] (クエリ結果を再利用) オプションを有効にします。



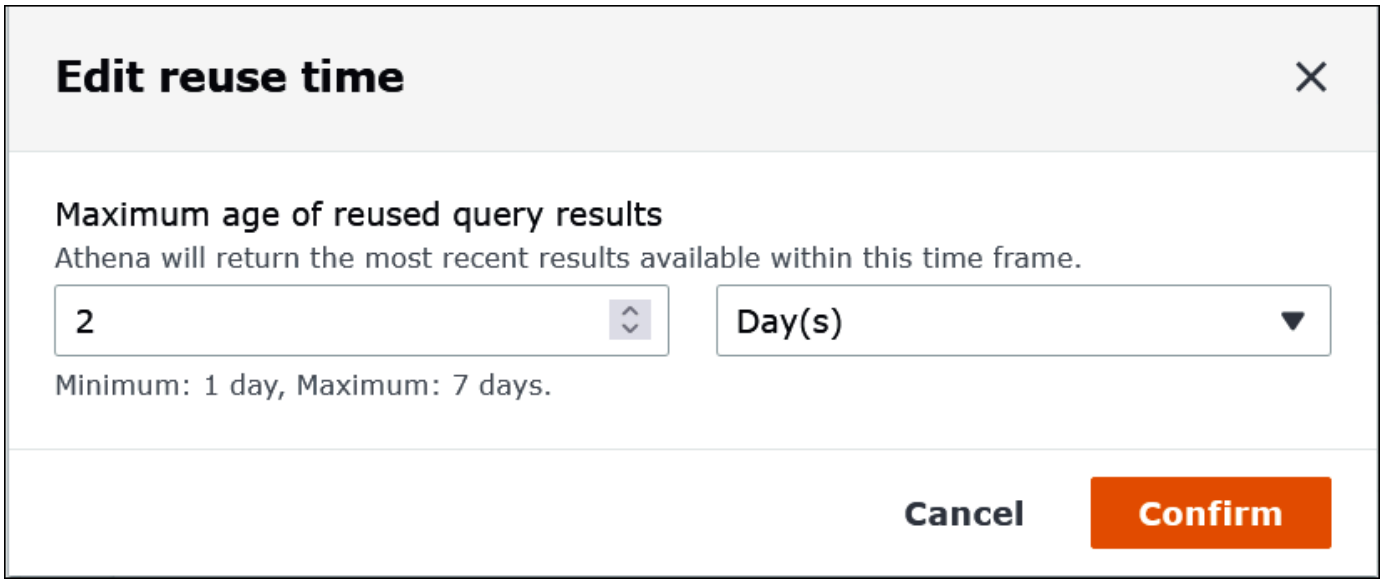
クエリ結果の再利用機能を設定するには

1. Athena クエリエディタの [Reuse query results] (クエリ結果を再利用) オプションで、[up to 60 minutes ago] (最大 60 分前まで) の横にある編集アイコンを選択します。
2. [Edit reuse time] (再利用時間の編集) ダイアログボックスの右側のボックスから、時間の単位 (分、時間、または日) を選択します。

3. 左側のボックスで、指定する時間の単位数を入力または選択します。入力できる最大時間は、選択した時間の単位に関係なく 7 日間分に相当します。

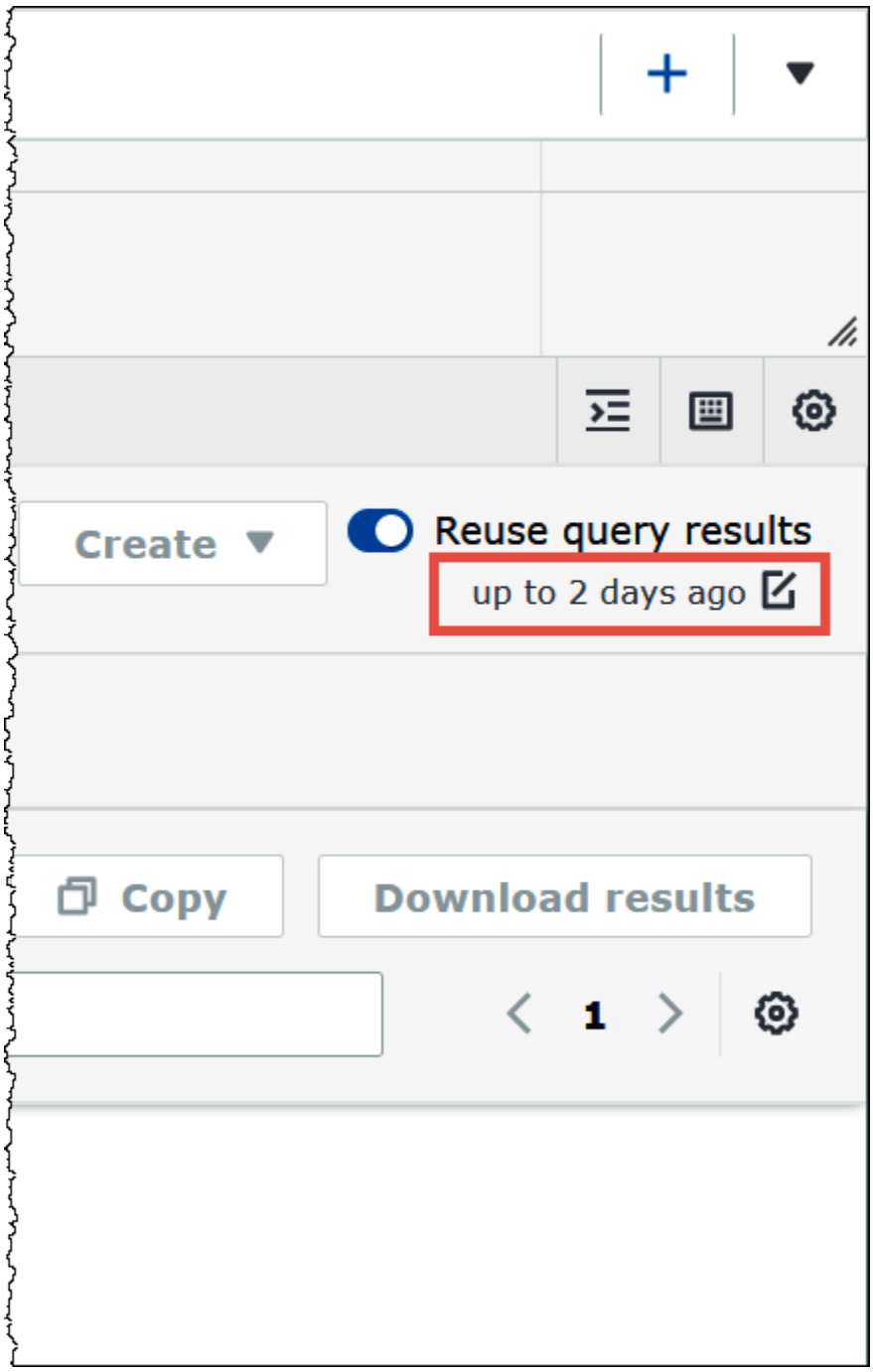


次の例では、最大再利用期間を 2 日間に指定しています。



4. [確認] を選択します。

設定の変更を確認するバナーが表示され、[Reuse query results] (クエリ結果を再使用) オプションに新しい設定が表示されます。

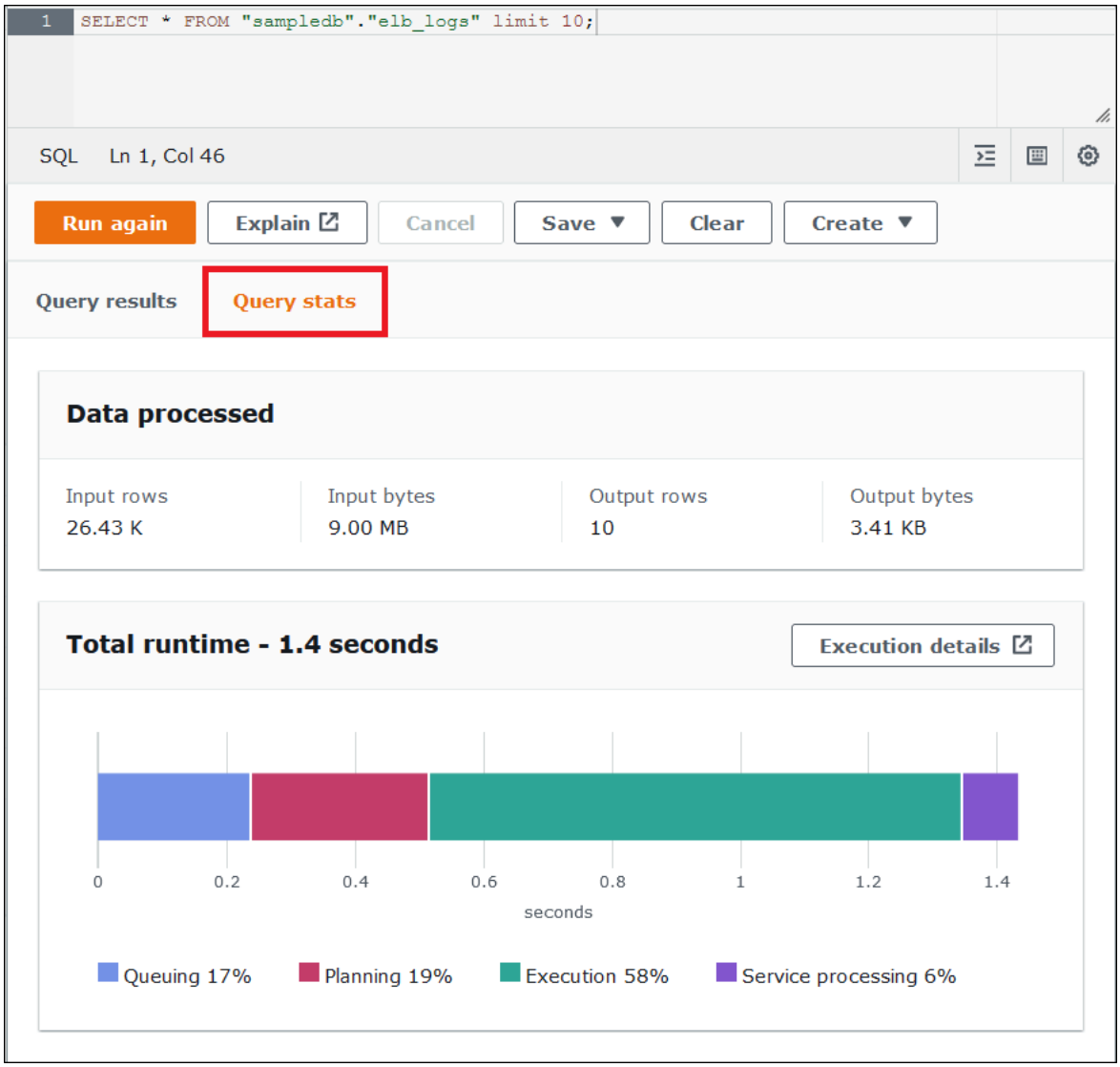


完了したクエリの統計と実行の詳細の表示

クエリを実行した後、処理された入力データと出力データに関する統計を取得し、クエリの各フェーズに要した時間のグラフィック表現を表示し、実行の詳細をインタラクティブに調べることができます。

完了したクエリのクエリ統計を表示するには

1. Athena クエリエディタでクエリを実行した後、[Query stats] (クエリ統計) タブを選択します。



[Query stats] (クエリの統計) タブには、次の情報が表示されます。

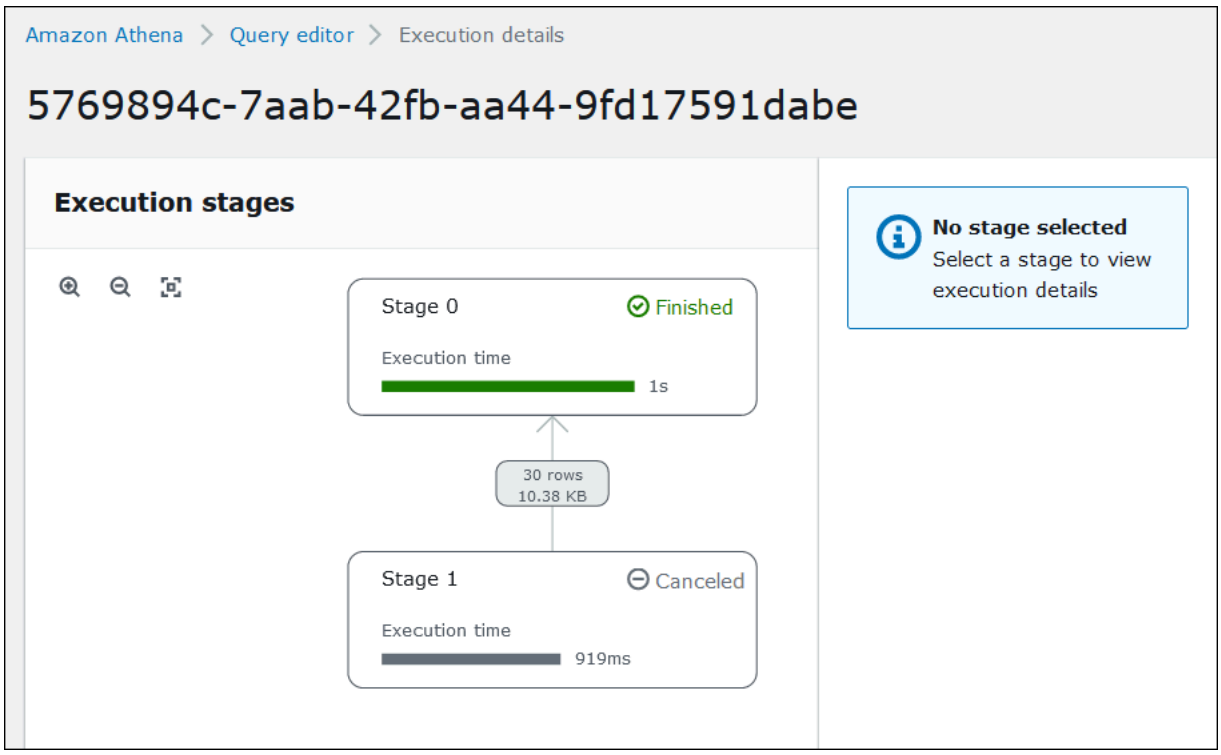
- [Data processed] (処理されたデータ) – 処理された入力行とバイト数、および出力された行数とバイト数を表示します。
- [The Total runtime] (合計実行時間) – クエリの実行に要した合計時間と、その時間のうち、キュー、計画、実行、およびサービス処理に費やされた時間のグラフィック表現を表示します。

> **Note**
>
> Lake Formation で行レベルのフィルターがクエリに定義されている場合、ステージレベルの入出力行数とデータサイズの情報は表示されません。

2. クエリの実行方法に関する情報をインタラクティブに調べるには、[Execution details] (実行の詳細) を選択します。

[Execution details] (実行の詳細) ページには、クエリの実行 ID と、クエリのゼロベースのステージのグラフが表示されます。ステージは、下から開始順に並べられます。各ステージのラベルは、ステージの実行に要した時間を示します。

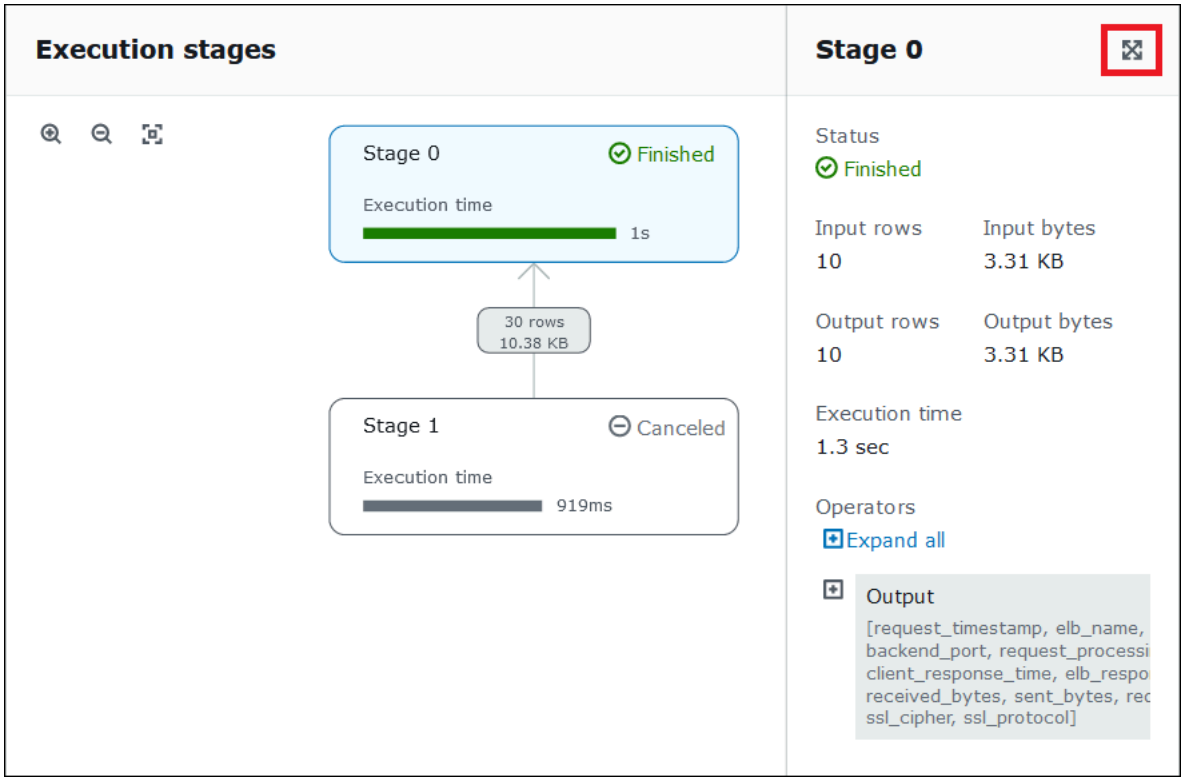
> **Note**
>
> クエリの合計実行時間と実行ステージ時間は、大きく異なることがよくあります。たとえば、合計実行時間が分単位になっているクエリが、ステージの実行時間を時間単位で表示する場合があります。ステージは、多数のタスク全体で並列実行される計算の論理ユニットであるため、ステージの実行時間は、そのタスクすべての実行時間を集約したものになります。このような相違はあるものの、ステージ実行時間は、クエリのどのステージが最も計算集約的かを示す相対的な指標として役立つ場合があります。



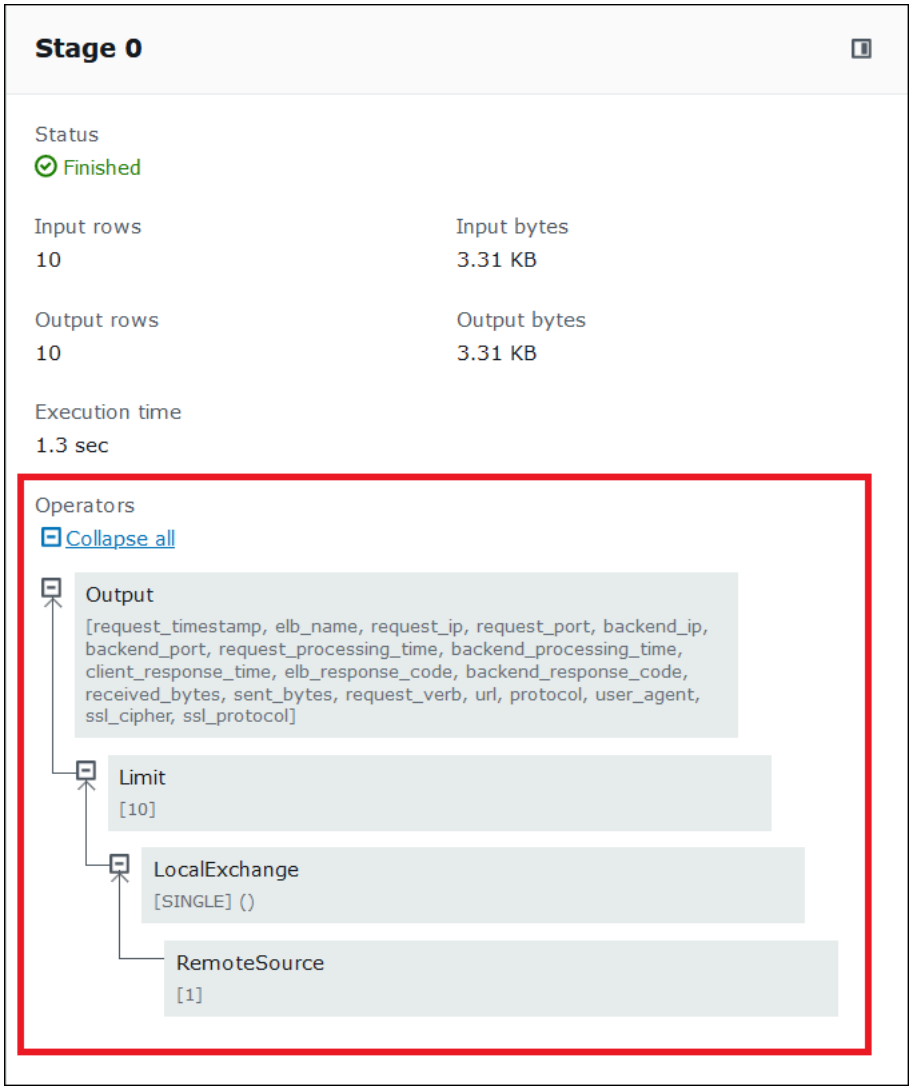
グラフを操作するには、次のオプションを使用します。

- ズームインまたはズームアウトするには、マウスでスクロールするか、拡大アイコンを使用します。
- 画面に合わせてグラフを調整するには、[Zoom to fit] (ズームして合わせる) アイコンを選択します。
- グラフを移動するには、マウスポインタをドラッグします。

3. ステージの詳細を表示するには、ステージを選択します。右側のステージ詳細ペインには、入力と出力の行数とバイト数、およびオペレーターツリーが表示されます。



4. ステージの詳細を全幅で表示するには、詳細ペインの右上にある拡大アイコンを選択します。
5. ステージの各部分に関する情報を取得するには、オペレーターツリーで 1 つ以上の項目を展開します。



実行の詳細については、「Athena EXPLAIN ステートメントの結果について」を参照してください。

追加リソース

詳細については、以下のリソースを参照してください。

SQL クエリの実行プランの表示

Athena での EXPLAIN および EXPLAIN ANALYZE の使用

ビューの使用

Amazon Athena のビューは、物理的なテーブルではなく、論理的なテーブルです。ビューを定義するクエリは、1 つのクエリでビューが参照されるたびに実行されます。

SELECT クエリからビューを作成でき、今後のクエリでこのビューを参照できます。詳細については、「CREATE VIEW」を参照してください。

トピック

- どのようなときにビューを使うのでしょう。
- Athena のビューでサポートされているアクション
- ビューの考慮事項
- ビューの制限
- コンソールでビューを操作する
- ビューの作成
- ビューの例
- AWS Glue Data Catalog ビューの使用

どのようなときにビューを使うのでしょう。

以下の目的でビューを作成できます。

- データのサブネットをクエリする。例えば、元のテーブルからの列のサブネットでビューを作成して、データのクエリを単純化できます。
- 複数のテーブルを 1 つのクエリに結合する。複数のテーブルを持ち、それらを UNION ALLと組み合わせる場合は、その式でビューを作成して、結合テーブルに対するクエリを簡素化できます。
- 既存の基本クエリの複合型を解消して、ユーザーによるクエリの実行を単純化する。基本クエリには、テーブル間の結合、列リストの表現や他の SQL 構文が含まれていることより、その理解とデバッグが困難なことがよくあります。複合型を解消して、クエリを単純化するビューを作成できます。
- 最適化手法を試用して、最適化されたクエリを作成する。たとえば、WHERE 条件、JOIN 順序、あるいはベストパフォーマンスを示す他の表現の組み合わせを見つけた場合には、これらの句および表現でビューを作成することができます。こうして、アプリケーションはこのビューに対して比較的な単純なクエリを実行できます。後に、元のクエリを最適化するためのより良い方法を見つけた場合には、ビューを再作成するときに、すべてのアプリケーションは最適化された基本クエリを即時に採用します。
- 基礎となるテーブルおよび列の名前を隠し、メンテナンス問題を最小限に抑える (この名前が変更された場合)。この場合、新しい名前を使用して、ビューを再作成します。基礎となるテーブルではなく、このビューを使用するすべてのクエリは、変更なしで実行を続けます。

Athena のビューでサポートされているアクション

Athena は、ビューで次のアクションをサポートします。このコマンドはクエリエディタで実行できます。

Statement	説明
CREATE VIEW	指定する SELECT クエリから新しいビューを作成します。詳細については、「ビューの作成」を参照してください。 オプションの OR REPLACE 句は、既存のビューを更新して置き換えます。
DESCRIBE VIEW	名前が付けられたビューの列のリストを表示します。これにより、複合型のビューの属性を確認できます。
DROP VIEW	既存のビューを削除します。オプションの IF EXISTS 句は、ビューが存在しない場合に、エラーを抑制します。
SHOW CREATE VIEW	指定するビューを作成する SQL ステートメントを表示します。
SHOW VIEWS	指定するデータベース、あるいは、データベース名を省略する場合の現在のデータベースのビューのリスト。オプションの LIKE 句を正規表現で使用して、ビュー名のリストを制限します。コンソールの左ペインでビューのリストを表示することもできます。
SHOW COLUMNS	ビュー用のスキーマの列を一覧表示します。

ビューの考慮事項

Athena でビューを作成して使用する場合は、次の考慮事項が適用されます。

- Athena では、Athena コンソール、AWS Glue Data Catalog (移行して使用している場合)、または同じカタログに接続されている Amazon EMR クラスターで実行されている Presto で作成されたビューをプレビューして使用できます。これら以外の方法で作成された Athena ビューをプレビューしたり、ビューに追加したりすることはできません。
- AWS Glue データカタログ経由でビューを作成している場合は、"PartitionKeys":[] のように、PartitionKeys パラメータを含めてその値を空のリストに設定する必要があります。そう

しない場合、ビューは Athena で失敗します。以下の例は、"PartitionKeys":[] を使用してデータカタログから作成されたビューを示しています。

```
aws glue create-table
--database-name mydb
--table-input '{
"Name":"test",
  "TableType": "EXTERNAL_TABLE",
  "Owner": "hadoop",
  "StorageDescriptor":{
     "Columns":[{
           "Name":"a","Type":"string"},{"Name":"b","Type":"string"}],
   "Location":"s3://DOC-EXAMPLE-BUCKET/Oct2018/25Oct2018/",
   "InputFormat":"org.apache.hadoop.mapred.TextInputFormat",
   "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
   "SerdeInfo":{"SerializationLibrary":"org.apache.hadoop.hive.serde2.OpenCSVSerde",
   "Parameters":{"separatorChar": "|", "serialization.format":
 "1"}}},"PartitionKeys":[]}'
```

- データカタログで Athena ビューを作成した場合、データカタログはビューをテーブルとして扱います。データカタログでテーブルレベルのきめ細かなアクセスコントロールを使用して、これらのビューへのアクセスを制限できます。
- Athena は再帰的なビューの実行を妨げ、そのような場合にはエラーメッセージを表示します。再帰的なビューとは、Athena 自体を参照するビュークエリです。
- Athena は、古いビューを検出するとエラーメッセージを表示します。以下のいずれかが発生すると、古いビューが報告されます。
  - このビューは、存在しないテーブルまたはデータベースを参照します。
  - スキーマまたはメタデータの変更は、参照されるテーブルで行われます。
  - 参照されるテーブルは削除され、異なるスキーマまたは設定で再度作成されます。
- ネストされたビューのクエリが有効であり、テーブルおよびデータベースが存在する限り、ネストされたビューを作成して実行できます。

ビューの制限

- Athena ビューの名前には、アンダースコア (_) 以外の特殊文字を使用できません。詳細については、「テーブル、データベース、および列の名前」を参照してください。

- ビューの名前に予約キーワードを使用しないようにします。予約キーワードを使用している場合、ビューでのクエリでは予約キーワードを二重引用符で囲みます。「予約済みキーワード」を参照してください。
- Athena で作成したビューを外部 Hive メタストア、UDF で使用することはできません。Hive で外部で作成したビューを使用する方法については、「Hive ビューの使用」を参照してください。
- 地理空間関数ではビューを使用できません。
- Amazon S3 のデータに対するアクセスコントロールを管理するためにビューを使用することはできません。ビューをクエリするには、Amazon S3 に保存されているデータにアクセスするためのアクセス許可が必要です。詳細については、「Amazon S3 へのアクセス権」を参照してください。
- アカウント間でのビューのクエリは Athena エンジンバージョン 2 と Athena エンジンバージョン 3 の両方でサポートされていますが、クロスアカウント AWS Glue Data Catalog を含むビューを作成することはできません。クロスアカウントデータカタログへのアクセスについては、「AWS Glue データカタログへのクロスアカウントアクセス」を参照してください。
- Hive または Iceberg の非表示メタデータ列 `$bucket`、`$file_modified_time`、`$file_size`、および `$partition` は Athena のビューでサポートされていません。Athena の `$path`メタデータ列の使用方法については「Amazon S3 内にあるソースデータのファイルの場所の取得 」を参照してください。

コンソールでビューを操作する

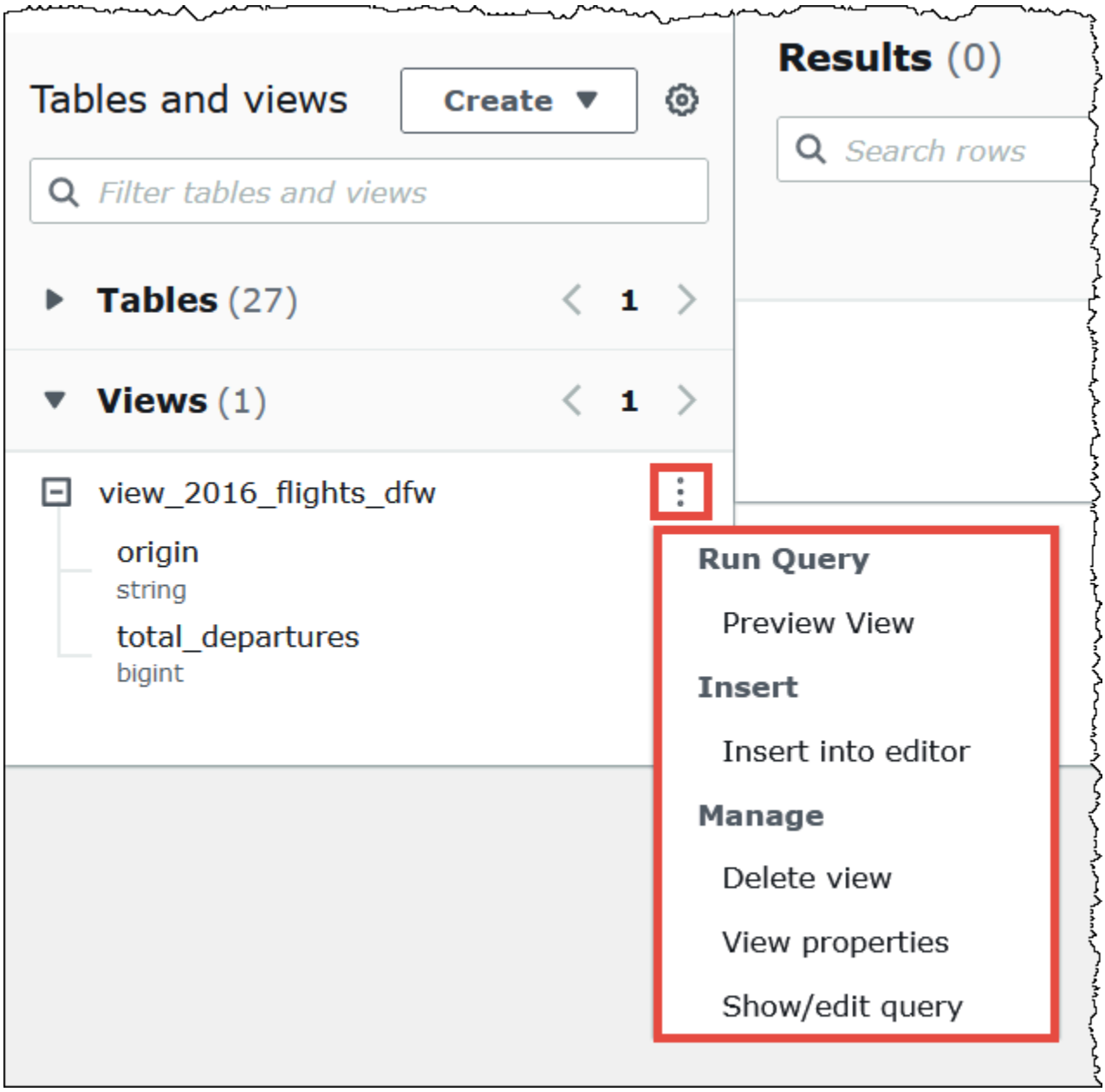
Athena コンソールでは、以下を実行することができます。

- 左側のペインで、テーブルがリストされているすべてのビューを見つける。
- ビューをフィルタリングする。
- ビューをプレビューしてそのプロパティを表示し、編集または削除する。

ビューのアクションを表示するには

ビューは、既に作成してある場合にのみコンソールに表示されます。

1. Athena コンソールで [Views] (ビュー) を選択し、ビューを選択して展開し、ビュー内の列を表示します。
2. ビューの横にある 3 つの縦のドットを選択して、ビューのアクションのリストを表示します。



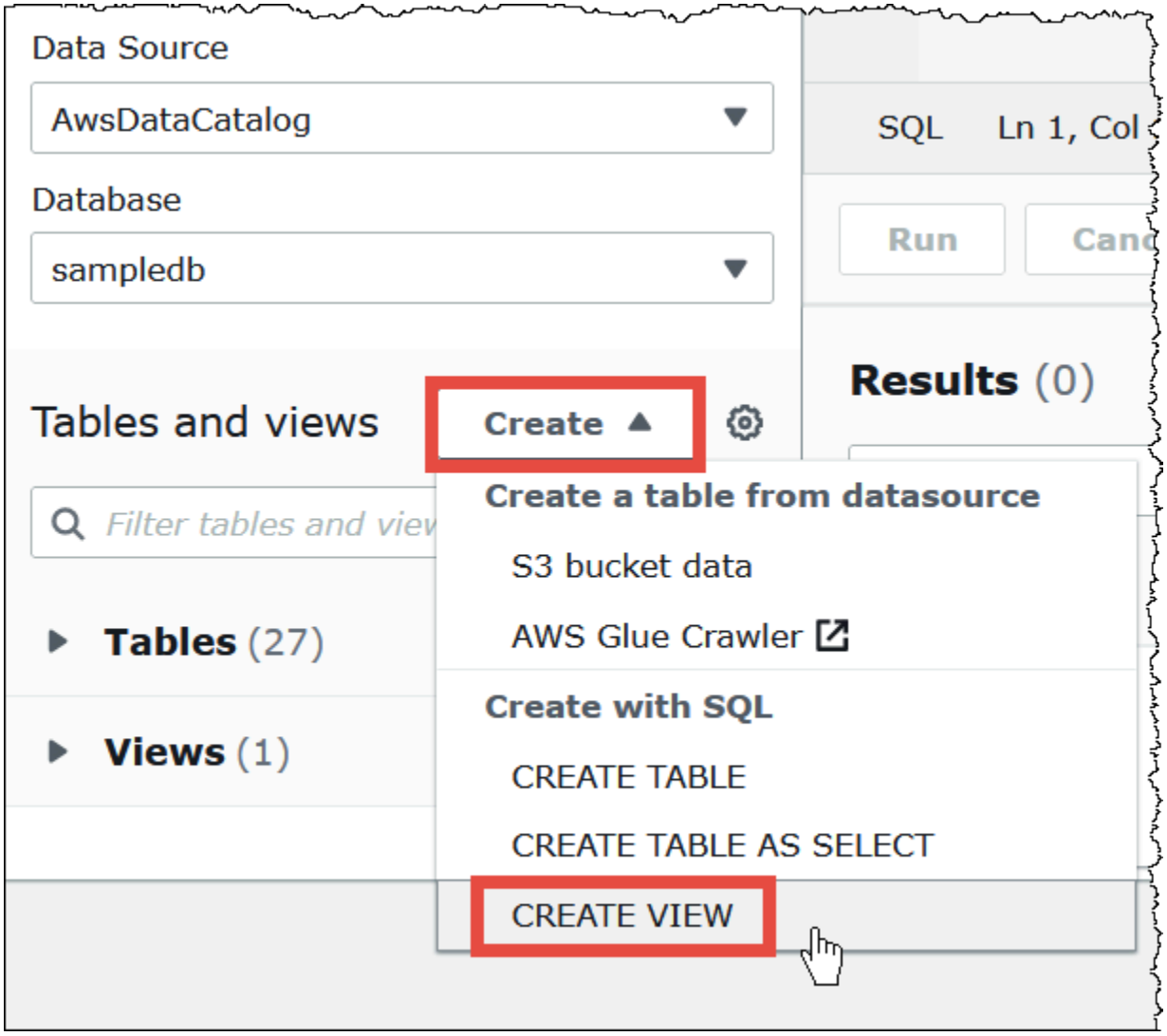
3. アクションを選択して、ビューのプレビュー、クエリエディタへのビュー名の挿入、ビューの削除、ビューのプロパティの表示、またはクエリエディタでのビューの表示と編集を行います。

ビューの作成

Athena コンソールでテンプレートを使用するか、既存のクエリを実行して、ビューを作成できます。

テンプレートを使用してビューを作成するには

1. Athena コンソールで、[Tables and views] (テーブルとビュー) の横にある [Create] (作成) を選択してから、[Create view] (ビューの作成) を選択します。



   このアクションにより、編集可能なビューテンプレートがクエリエディタに配置されます。

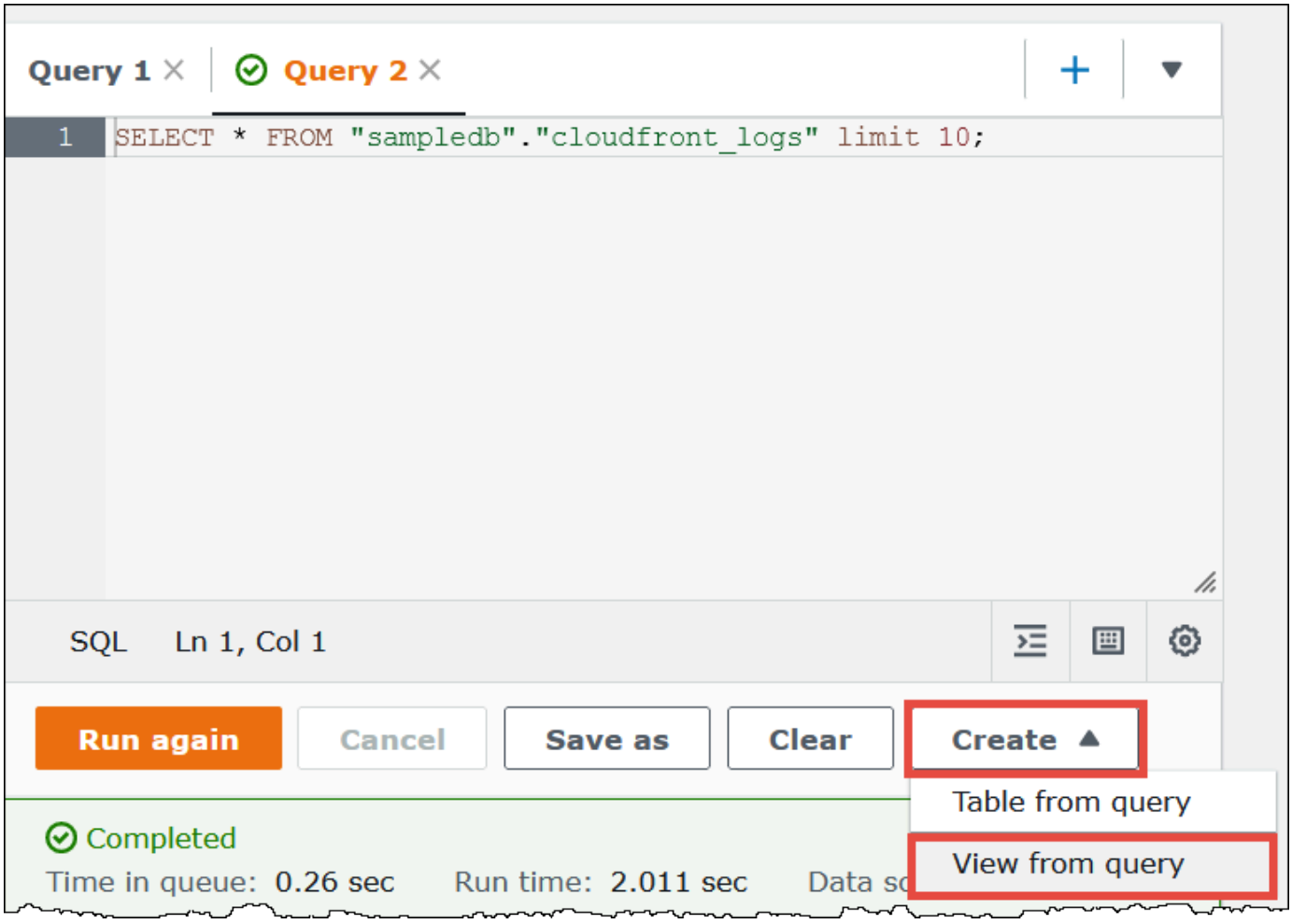
2. 要件に応じてビューテンプレートを編集します。ステートメントにビューの名前を入力するときは、ビュー名にアンダースコア (_) 以外の特殊文字を含めることはできません。「テーブル、データベース、および列の名前」を参照してください。ビューの名前に 予約済みキーワード を使用しないようにします。

   ビューの作成の詳細については、「CREATE VIEW」および「ビューの例」を参照してください。

3. [Run] (実行) をクリックしてビューを作成します。Athena コンソールのビューのリストにビューが表示されます。

既存のクエリからビューを作成するには

1. Athena クエリエディタを使用して、既存のクエリを実行します。
2. クエリエディタウィンドウで、[Create] (作成) を選択してから、[View from query] (クエリから表示する) を選択します。



3. [Create View] (ビューの作成) ダイアログボックスにビュー名を入力し、[Create] (作成) を選択します。ビュー名にはアンダースコア (_) 以外の特殊文字は使用できません。「テーブル、データベース、および列の名前」を参照してください。ビューの名前に 予約済みキーワード を使用しないようにします。

   Athena はコンソール内のビューのリストにビューを追加し、クエリエディタのビューの `CREATE VIEW` ステートメントを表示します。

メモ

- テーブルの基になるテーブルを削除してからそのビューの実行を試みると、Athena がエラーメッセージを表示します。
- ネストされたビューを作成することができます。これは、既存のビューの上にあるビューです。Athena は、それ自体を参照する再帰的なビューの実行を妨げます。

ビューの例

ビュークエリの構文を表示するには、SHOW CREATE VIEW を使用します。

Example 例 1

次の 2 つのテーブルを考えてみます。2 つの列 (employees と id) がある name テーブルと 2 つの列 (salaries と id) がある salary テーブルです。

この例では、name_salary という名前のビューを SELECT テーブル および employees テーブルから給料にマッピングされた ID のリストを取得する salaries クエリとして作成します。

```
CREATE VIEW name_salary AS
SELECT
 employees.name,
 salaries.salary
FROM employees, salaries
WHERE employees.id = salaries.id
```

Example 例 2

次の例では、view1 という名前のビューを作成して、複合型のクエリ構文を非表示にします。

このビューは 2 つのテーブル (table1 および table2) の上部で実行され、各テーブルは異なる SELECT クエリです。このビューは、table1 からの列を選択し、結果を table2 に結合します。この結合は、両方のテーブルにある a 列に基づいています。

```
CREATE VIEW view1 AS
WITH
  table1 AS (
         SELECT a,
         MAX(b) AS the_max
         FROM x
```

```
        GROUP BY a
        ),
  table2 AS (
        SELECT a,
        AVG(d) AS the_avg
        FROM y
        GROUP BY a)
SELECT table1.a, table1.the_max, table2.the_avg
FROM table1
JOIN table2
ON table1.a = table2.a;
```

フェデレーティッドビューの詳細については、「フェデレーションされたビューのクエリ」を参照してください。

AWS Glue Data Catalog ビューの使用

この機能はプレビューリリースであり、変更される可能性があります。詳細については、「AWS のサービス条件」ドキュメントの「ベータ版とプレビュー」セクションを参照してください。

Amazon Athena や Amazon Redshift などの AWS のサービス 全体で 1 つの共通ビューが必要な場合は AWS Glue Data Catalog ビューを使用します。データカタログビューでは、アクセス許可はビューをクエリするユーザーではなく、ビューを作成したユーザーによって定義されます。この権限の付与方法は、definer セマンティクスと呼ばれます。

以下のユースケースは、データカタログビューの使用例です。

- アクセスコントロールの強化 — ユーザーが必要とするアクセス許可のレベルに基づいてデータアクセスを制限するビューを作成します。例えば、データカタログビューを使用して、人事 (HR) 部門に属さない従業員が個人を特定できる情報を表示できないようにすることができます。
- 完全なレコードを確保 — データカタログビューに特定のフィルターを適用することで、データカタログビューのデータレコードを常に完全な状態にすることができます。
- セキュリティの強化 — データカタログビューでビューを作成するには、ビューを作成するクエリ定義をそのまま維持する必要があります。これにより、データカタログビューは悪意のある行為者からの SQL コマンドの影響を受けにくくなります。
- 基礎となるテーブルへのアクセスを防ぐ — Definer セマンティクスにより、ユーザーは基礎となるテーブルを利用できるようにしなくてもビューにアクセスすることができます。ビューを定義したユーザーのみがテーブルにアクセスする必要があります。

データカタログビュー定義は AWS Glue Data Catalog に保存されます。つまり、AWS Lake Formation を使用して、リソース許可、列付与、またはタグベースのアクセスコントロールを通じてアクセスを許可できます。Lake Formation でのアクセス権の付与と取り消しについての詳細は、「AWS Lake Formation デベロッパーガイド」の「Data Catalog リソースに対する許可の付与と取り消し」を参照してください。

アクセス許可

データカタログビューには `Lake Formation Admin`、`Definer`、`Invoker` の 3 つのロールが必要です。

- **`Lake Formation Admin`** — すべてのLake Formation 権限を設定するためのアクセス権があります。
- **`Definer`** — データカタログビューを作成します。`Definer` ロールには、ビュー定義が参照するすべての基礎となるテーブルに対して付与可能で完全な `SELECT` 権限が必要です。
- **`Invoker`** — データカタログビューにクエリを実行したり、そのメタデータを確認したりできます。

`Definer` ロールの信頼関係によって、以下の例のように AWS Glue と Lake Formation のサービスプリンシパルに対する `sts:AssumeRole` アクションを許可する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "glue.amazonaws.com",
                    "lakeformation.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Athena にアクセスするための IAM アクセス許可も必要です。詳細については、「Amazon Athena の AWS 管理ポリシー」を参照してください。

制限事項

- データカタログビューは他のビューを参照できません。
- ビュー定義では最大 10 個のテーブルを参照できます。
- 基礎となるテーブルは、Lake Formation に登録されている必要があります。
- DEFINER プリンシパルは IAM ロールのみにできます。
- DEFINER ロールには、基礎となるテーブルに対する完全な SELECT (付与可能な) 権限が必要です。
- UNPROTECTED データカタログビューはサポートされていません。
- ユーザー定義関数 (UDF) は、ビュー定義ではサポートされていません。
- Athena 統合データソースはデータカタログビューでは使用できません。
- Data Catalog ビューは外部 Hive メタストアではサポートされていません。
- Athena は、古いビューを検出するとエラーメッセージを表示します。以下のいずれかが発生すると、古いビューが報告されます。
  - このビューは、存在しないテーブルまたはデータベースを参照します。
  - スキーマまたはメタデータの変更は、参照されるテーブルで行われます。
  - 参照されるテーブルは削除され、異なるスキーマまたは設定で再度作成されます。

データカタログビューの作成

次の構文例は、Definer ロールのユーザーが orders_by_date データカタログビューを作成する方法を示しています。この例では、Definer ロールが default データベース内の orders テーブルに対するすべての SELECT 権限を持っていることを前提としています。

```
CREATE PROTECTED MULTI DIALECT VIEW orders_by_date
SECURITY DEFINER
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

データカタログビューをクエリする

ビューが作成されると、Lake Formation Admin はデータカタログビューに対する SELECT 権限を Invoker プリンシパルに付与できます。その後、Invoker プリンシパルは、ビューが参照する

基になるベーステーブルにアクセスしなくてもビューをクエリできます。以下は Invoker クエリの例です。

```
SELECT * from orders_by_date where price > 5000
```

データカタログビューの更新

Lake Formation Admin または Definer は、ALTER VIEW UPDATE DIALECT 構文を使用してビュー定義を更新できます。次の例では、ビュー定義を変更して、orders テーブルではなく returns テーブルから列を選択します。

```
ALTER VIEW orders_by_date UPDATE DIALECT
AS
SELECT return_date, sum(totalprice) AS price
FROM returns
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

データカタログビューを作成および管理するための構文の詳細については、「Glue データカタログビュー構文」を参照してください。

Glue データカタログビュー構文

この機能はプレビューリリースであり、変更される可能性があります。詳細については、「AWS のサービス条件」ドキュメントの「ベータ版とプレビュー」セクションを参照してください。

このセクションでは、AWS Glue Data Catalog ビューを作成および管理するためのデータ定義言語 (DDL) コマンドについて説明します。

ALTER VIEW DIALECT

データカタログビューを更新するには、エンジンダイアレクトを追加するか、既存のエンジンダイアレクトを更新または削除します。データカタログビューで ALTER VIEW DIALECT ステートメントを使用する権限を持っているのは、Lake FormationAdmin と Definer (ビューを作成したユーザー) のみです。

構文

```
ALTER VIEW name [ FORCE ] [ ADD|UPDATE ] DIALECT AS query
```

```
ALTER VIEW name [ DROP ] DIALECT
```

FORCE

FORCE キーワードにより、ビュー内の競合するエンジンダイアレクト情報が新しい定義で上書きされます。FORCE キーワードは、データカタログビューの更新後に、既存のエンジンダイアレクト間でビュー定義に競合が発生した場合に役立ちます。例えば、データカタログビューに Athena と Amazon Redshift の両方のダイアレクトがあり、更新後に、ビュー定義で Amazon Redshift と競合が発生したとします。この場合は、FORCE キーワードを使用して更新を完了させ、Amazon Redshift ダイアレクトを古いものとしてマークできます。古いとマークされたエンジンがビューをクエリすると、クエリは失敗します。エンジンは例外をスローして、古い結果を許可しません。これを修正するには、ビュー内の古いダイアレクトを更新します。

ADD

データカタログビューに新しいエンジンダイアレクトを追加します。指定するエンジンは、データカタログビューに既に存在していない必要があります。

UPDATE

データカタログビューに既に存在するエンジンダイアレクトを更新します。

DROP

既存のエンジンダイアレクトをデータカタログビューから削除します。データカタログビューからエンジンを削除すると、削除されたエンジンからデータカタログビューをクエリできなくなります。ビュー内の他のエンジンダイアレクトは引き続きビューをクエリできます。

DIALECT AS

エンジン固有の SQL クエリを導入します。

例

```
ALTER VIEW orders_by_date FORCE ADD DIALECT
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

```
ALTER VIEW orders_by_date FORCE UPDATE DIALECT
```

```
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

```
ALTER VIEW orders_by_date DROP DIALECT
```

保護されたマルチダイアレクトの作成

AWS Glue Data Catalog にデータカタログビューを作成します。データカタログビューは、Athena と Amazon Redshift や Amazon EMR のような他の SQL エンジン間でシームレスに機能する単一のビュースキーマです。

構文

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW view_name
 [ SECURITY DEFINER ]
AS query
```

PROTECTED

キーワードが必須です。ビューをデータ漏洩から保護するように指定します。データカタログビューは PROTECTED ビューとしてのみ作成できます。

MULTI DIALECT

ビューがさまざまなクエリエンジンの SQL ダイアレクトをサポートし、それらのエンジンで読み取れるように指定します。

SECURITY DEFINER

このビューに対して definer セマンティクスが有効であることを指定します。definer セマンティクスとは、基礎となるテーブルに対する有効な読み取り権限が、実際の読み取りを実行するプリンシパルではなく、ビューを定義したプリンシパルまたはロールに属することを意味します。

OR REPLACE

データカタログビューに他のエンジンの SQL ダイアレクトが含まれている場合、そのビューを置き換えることはできません。呼び出し側のエンジンの SQL ダイアレクトがビュー内に存在する場合は、そのビューを置き換えることができます。

例

次の例では、orders テーブルに対するクエリに基づいて orders_by_date データカタログビューを作成します。

```
CREATE PROTECTED MULTI DIALECT VIEW orders_by_date
SECURITY DEFINER
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

DESCRIBE

指定されたデータカタログビューの列のリストを表示します。DESCRIBE ステートメントは、Athena ビューの DESCRIBE ステートメントと似ています。Athena ビューとは異なり、ステートメントの出力は Lake Formation アクセスコントロールによって制御されます。そのため、このクエリの出力はビューのすべての列ではなく、呼び出し元がアクセスできる列のみになります。

構文

```
DESCRIBE [db_name.]view_name
```

例

```
DESCRIBE orders
```

DROP VIEW

データカタログビューを削除するのは、呼び出し側のエンジンダイアレクトがデータカタログビューに表示されている場合のみです。例えば、ユーザーが Athena から DROP VIEW を呼び出した場合、ビューに Athena のダイアレクトが存在する場合にのみビューが削除されます。それ以外の場合は、このオペレーションは失敗します。Lake Formation 管理者とビュー定義者のみが、データカタログビューで DROP VIEW ステートメントを使用する権限を持っています。

構文

```
DROP VIEW [ IF EXISTS ] view_name
```

例

```
DROP VIEW orders_by_date
```

```
DROP FORCE VIEW IF EXISTS orders_by_date
```

オプションの IF EXISTS 句は、ビューが存在しない場合に、エラーが制御される原因となります。

SHOW COLUMNS

単一の指定したデータカタログビューの列の名前のみを表示します。SHOW COLUMNS ステートメントは、Athena ビューの SHOW COLUMNS ステートメントと似ています。Athena ビューとは異なり、ステートメントの出力は Lake Formation アクセスコントロールによって制御されます。そのため、このクエリの出力はビューのすべての列ではなく、呼び出し元がアクセスできる列のみになります。

構文

```
SHOW COLUMNS {FROM|IN} database_name.view_name
```

```
SHOW COLUMNS {FROM|IN} view_name [{FROM|IN} database_name]
```

ビューの作成を表示する

データカタログビューを作成した SQL 構文が表示されます。返される SQL には、Athena で使用されているビュー作成構文が表示されます。データカタログビューに対して SHOW CREATE VIEW を呼び出す権限があるのは、Lake Formation 管理者とビュー定義者のプリンシパルのみです。

構文

```
SHOW CREATE VIEW view_name
```

例

```
SHOW CREATE VIEW orders_by_date
```

ビューの表示

データベース内のすべてのビューの名前を一覧表示します。Athena エンジン SQL ダイアレクトを使用するデータベース内のすべてのデータカタログビューが一覧表示されます。ビューに Athena エンジンダイアレクトがない他のデータカタログビューは除外されます。

構文

```
SHOW VIEWS [IN database_name] [LIKE 'regular_expression']
```

例

```
SHOW VIEWS
```

```
SHOW VIEWS IN marketing_analytics LIKE 'orders*'
```

保存されたクエリの使用

Athena クエリエディタで作成したクエリを保存、編集、実行、名前変更、および削除するには、Athena コンソールを使用することができます。

考慮事項と制約事項

- 保存したクエリの名前、説明、およびクエリテキストを更新できます。
- 自分のアカウントのクエリのみを更新できます。
- クエリが属するワークグループまたはデータベースを変更することはできません。
- Athena にクエリの変更履歴は保存されません。特定のバージョンのクエリを保持する必要がある場合は、別の名前で保存してください。

Athena コンソールでの保存されたクエリの操作

クエリを保存して名前を付けるには

1. Athena コンソールのクエリエディタで、クエリを入力するか実行します。
2. クエリエディタウィンドウの上にあるクエリのタブで 3 つの縦のドットを選択し、[Save as] (名前を付けて保存) を選択します。
3. [Save query] (クエリの保存) ダイアログボックスに、クエリの名前と任意で説明を入力します。展開できる [Preview SQL query] (SQL クエリのプレビュー) ウィンドウを使用して、クエリを保存する前にクエリの内容を確認します。
4. [Save query] (クエリの保存) を選択します。

   クエリエディタでは、クエリのタブに指定した名前が表示されます。

保存されたクエリを実行するには

1. Athena コンソールを開き、[Saved queries] (保存されたクエリ) タブを選択します。
2. [Saved queries] (保存されたクエリ) リストで、実行するクエリの ID を選択します。

   クエリエディタに、選択したクエリが表示されます。
3. [Run] (実行) を選択します。

保存されたクエリを編集するには

1. Athena コンソールを開き、[Saved queries] (保存されたクエリ) タブを選択します。
2. [Saved queries] (保存されたクエリ) リストで、編集するクエリの ID を選択します。
3. クエリエディタでクエリを編集します。
4. 次のいずれかのステップを実行します。

   - クエリを実行するには、[Run] (実行) を選択します。
   - クエリを保存するには、クエリのタブで、3 つの縦のドットを選択し、[Save] (保存) を選択します。
   - 別の名前でクエリを保存するには、クエリのタブで、[Save as] (名前を付けて保存) を選択します。

クエリエディタに既に表示されている保存済みのクエリの名前を変更またはクエリを削除するには

1. クエリのタブにある 3 つの縦のドットを選択し、[Rename] (名前の変更) または [Delete] (削除) を選択します。
2. プロンプトに従って、クエリの名前を変更またはクエリを削除します。

クエリエディタに表示されていない保存済みクエリの名前を変更するには

1. Athena コンソールを開き、[Saved queries] (保存されたクエリ) タブを選択します。
2. 名前を変更するクエリのチェックボックスを選択します。
3. [名前の変更] を選択します。
4. [Rename query] (クエリの名前変更) ダイアログボックスで、クエリ名とクエリの説明を編集します。展開できる [Preview SQL query] (SQL クエリのプレビュー) ウィンドウを使用して、クエリ名を変更する前にクエリの内容を確認します。

5. [Rename query] (クエリの名前変更) を選択します。

   名前を変更したクエリが [Saved queries] (保存されたクエリ) リストに表示されます。

クエリエディタに表示されていない保存済みクエリを削除するには

1. Athena コンソールを開き、[Saved queries] (保存されたクエリ) タブを選択します。
2. 削除するクエリのチェックボックスを 1 つ以上選択します。
3. [削除] を選択します。
4. 確認プロンプトで、[Delete] (削除) を選択します。

   1 つ以上のクエリが、[Saved queries] (保存されたクエリ) リストから削除されます。

Athena API を使用して保存されたクエリを更新する

Athena API を使用して保存されたクエリを更新する方法の詳細については、Athena API リファレンスの UpdateNamedQuery アクションを参照してください。

パラメータ化されたクエリの使用

Athena のパラメータ化されたクエリを使用して、実行時に同じクエリを異なるパラメータ値で再実行し、SQL インジェクション攻撃を防ぐことができます。Athena では、パラメータ化されたクエリは、任意の DML クエリまたは SQL プリペアドステートメントの実行パラメータの形式をとることができます。

- 実行パラメータを持つクエリは 1 つのステップで実行でき、ワークグループ固有ではありません。パラメータ化する値の DML クエリには、疑問符を付けます。クエリを実行するときは、実行パラメータ値を順番に宣言します。パラメータの宣言とパラメータへの値の割り当ては、同じクエリで、しかし分離された方法で行えます。プリペアドステートメントとは異なり、実行パラメータを含むクエリを送信するときにワークグループを選択できます。
- プリペアドステートメントには、PREPARE と EXECUTE の 2 つの別個の SQL ステートメントが必要です。まず、PREPARE ステートメントでパラメータを定義します。その後、定義したパラメータの値を指定する EXECUTE ステートメントを実行します。プリペアドステートメントはワークグループ固有です。それらが属するワークグループのコンテキスト外では実行できません。

考慮事項と制約事項

- パラメータ化されたクエリは、Athena エンジンバージョン 2 以降でサポートされています。Athena エンジンバージョンの詳細については、「Athena エンジンのバージョニング」を参照してください。
- 現在、パラメータ化されたクエリは、SELECT、INSERT INTO、CTASおよび UNLOAD ステートメントでのみサポートされています。
- パラメータ化されたクエリでは、パラメータは位置パラメータであり、? で示されます。パラメータには、クエリ内の順序によって値が割り当てられます。名前が挙げられているパラメータはサポートされていません。
- 現在、? パラメータは WHERE 句にのみ配置できます。SELECT ? FROM table のような構文はサポートされていません。
- 疑問符パラメータを二重引用符または一重引用符で囲むことはできません (つまり、'?' および "?" は有効な構文ではありません)。
- SQL 実行パラメータを文字列として扱うには、二重引用符ではなく一重引用符で囲む必要があります。
- 必要に応じて、パラメータ化された条件に値を入力するときに CAST 関数を使用できます。たとえば、クエリでパラメータ化した date 型の列があり、データ 2014-07-05 をクエリする場合は、パラメータ値に CAST('2014-07-05' AS DATE) を入力すると結果が返されます。
- 準備済みステートメントはワークグループ固有であり、準備済みステートメント名はワークグループ内で一意である必要があります。
- 準備済みステートメントには IAM 許可が必要です。詳細については、「準備済みステートメントへのアクセスを許可する」を参照してください。
- Athena コンソールの実行パラメータを持つクエリでは、疑問符の数が最大 25 個に制限されます。

実行パラメータを使用したクエリ

DML クエリで疑問符のプレースホルダーを使用すると、先にプリペアドステートメントを作成しなくても、パラメータ化されたクエリを作成できます。これらのクエリを実行するには、Athena コンソールを使用するか、AWS CLI または AWS SDK を使用して、execution-parameters 引数で変数を宣言します。

Athena コンソールでの実行パラメータを使用したクエリの実行

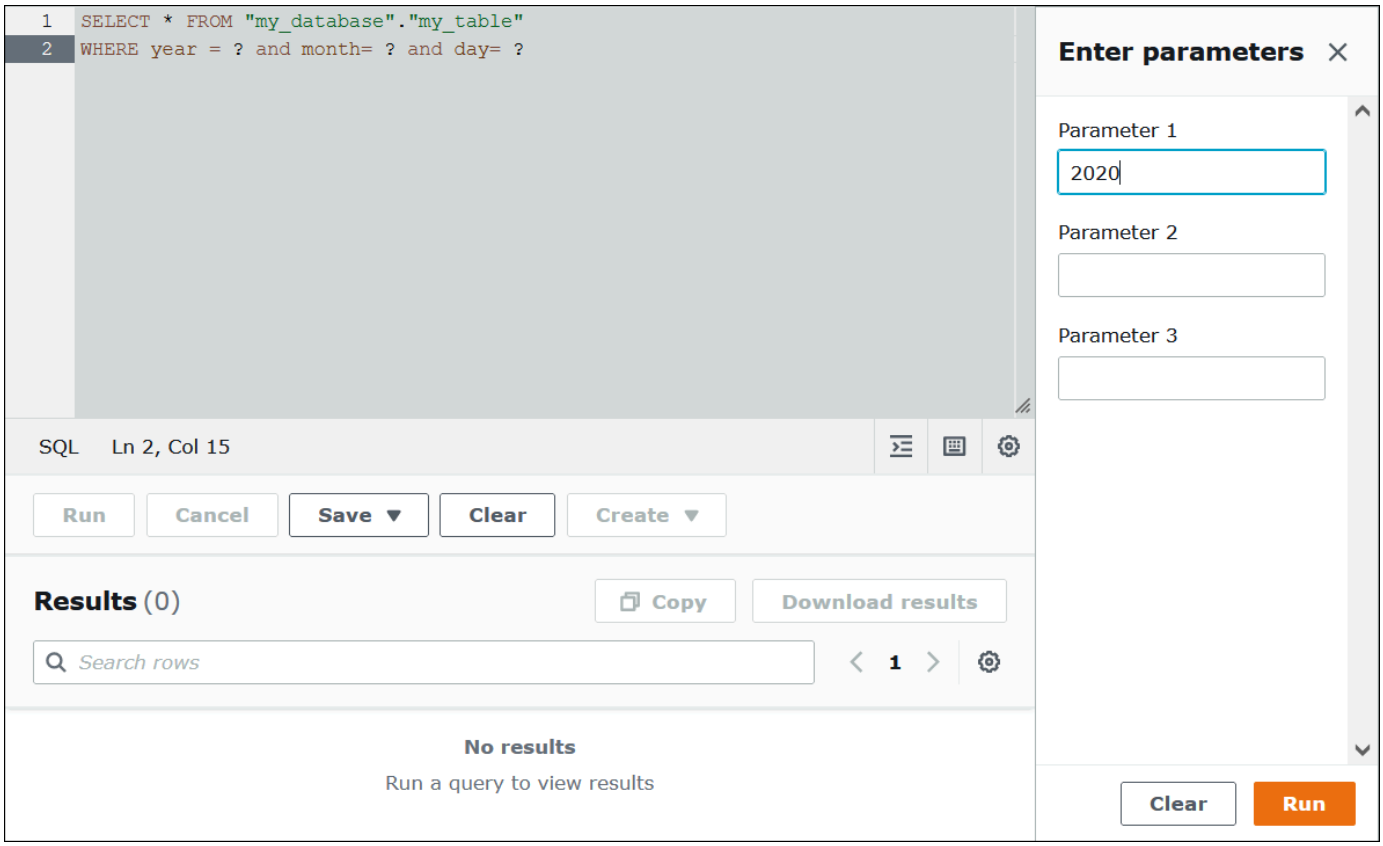
Athena コンソールで実行パラメータ (疑問符) を含むパラメータ化されたクエリを実行すると、クエリで疑問符が出現する順序で値の入力を求められます。

実行パラメータを持つクエリを実行するには

1. 次の例のように、疑問符のプレースホルダーを含むクエリを Athena エディタで入力します。

```
SELECT * FROM "my_database"."my_table"
WHERE year = ? and month= ? and day= ?
```

2. [Run] を選択します。
3. [Enter parameters] (パラメータを入力) ダイアログボックスで、クエリ内の各疑問符について順番に値を入力します。



4. パラメータの入力が終了したら、[Run] (実行) を選択します。エディタには、入力したパラメータ値のクエリ結果が表示されます。

この時点で、次のいずれかを実行できます。

- 同じクエリに異なるパラメータ値を入力してから、[Run again] (再度実行) を選択します。
- 入力したすべての値を一度にクリアするには、[Clear] (クリア) を選択します。
- クエリを直接編集するには (たとえば、疑問符を追加または削除するには)、最初に [Enter parameters] (パラメータを入力) ダイアログボックスを閉じます。
- パラメータ化されたクエリを後で使用するために保存するには、[Save] (保存) または [Save as] (名前を付けて保存) を選択し、クエリに名前を付けます。保存されるクエリの使用に関する詳細については、「保存されたクエリの使用」を参照してください。

便宜のため、[Enter parameters] (パラメータを入力) ダイアログボックスでは、クエリエディタで同じタブを使用する限り、クエリ用に以前に入力した値が記憶されます。

AWS CLI と実行パラメータを使用したクエリの実行

AWS CLI で実行パラメータを使用してクエリを実行するには、`start-query-execution` コマンドを使用して、`query-string` 引数でパラメータ化されたクエリを指定します。その後、`execution-parameters` 引数で実行パラメータの値を指定します。以下に示しているのはこの手法の例です。

```
aws athena start-query-execution
--query-string "SELECT * FROM table WHERE x = ? AND y = ?"
--query-execution-context "Database"="default"
--result-configuration "OutputLocation"="s3://DOC-EXAMPLE-BUCKET;/..."
--execution-parameters "1" "2"
```

準備済みステートメントを使用したクエリ

準備済みステートメントを使用して、異なるクエリパラメータを使用して同じクエリを繰り返し実行できます。準備済みステートメントにはパラメータプレースホルダが含まれており、それらの値は実行時に提供されます。

> **Note**
>
> ワークグループ内の準備済みステートメントの最大数は 1,000 です。

SQL ステートメント

Athena コンソールのクエリエディタでのパラメータ化されたクエリの実行には、PREPARE、EXECUTE、および DEALLOCATE PREPARE の SQL ステートメントを使用できます。

- 通常はリテラル値を使用する箇所にパラメータを指定するには、PREPARE ステートメントで疑問符を使用します。
- クエリの実行時にパラメータを値に置き換えるには、EXECUTE ステートメントで USING 句を使用します。
- ワークグループ内の準備済みステートメントから準備済みステートメントを削除するには、DEALLOCATE PREPARE ステートメントを使用します。

以下のセクションでは、これらの各ステートメントをさらに詳しく説明します。

PREPARE

後ほど実行されるステートメントを準備します。準備済みステートメントは、ユーザー指定の名前で現在のワークグループに保存されます。ステートメントには、リテラルの代わりに、クエリの実行時に置き換えられるパラメータを含められます。値に置き換えられるパラメータは、疑問符で表されます。

構文

```
PREPARE statement_name FROM statement
```

以下の表は、これらのパラメータの説明です。

パラメータ	説明
statement_name	準備されるステートメントの名前です。この名前は、ワークグループ内で一意である必要があります。
statement	SELECT、CTAS、または INSERT INTO クエリです。

PREPARE の例

以下の例では、PREPARE ステートメントを使用しています。疑問符は、クエリの実行時に EXECUTE ステートメントによって提供される値を表します。

```
PREPARE my_select1 FROM
SELECT * FROM nation
```

```
PREPARE my_select2 FROM
SELECT * FROM "my_database"."my_table" WHERE year = ?
```

```
PREPARE my_select3 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
```

```
PREPARE my_insert FROM
INSERT INTO cities_usa (city, state)
SELECT city, state
FROM cities_world
WHERE country = ?
```

```
PREPARE my_unload FROM
UNLOAD (SELECT * FROM table1 WHERE productid < ?)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format='PARQUET')
```

EXECUTE

準備済みステートメントを実行します。USING 句にパラメータの値を指定しています。

構文

```
EXECUTE statement_name [USING value1 [ ,value2, ... ] ]
```

statement_name は、準備済みステートメントの名前です。*value1* と *value2* は、ステートメント内のパラメータに指定される値です。

EXECUTE の例

以下の例では、パラメータが含まれない my_select1 準備済みステートメントを実行しています。

```
EXECUTE my_select1
```

次の例では、単一のパラメータを含む my_select2 準備済みステートメントを実行しています。

```
EXECUTE my_select2 USING 2012
```

次の例では、2 つのパラメータを含む my_select3 準備済みステートメントを実行しています。

```
EXECUTE my_select3 USING 346078, 12
```

以下の例は、準備済みステートメント my_insert 内のパラメータに対する文字列値を提供します。

```
EXECUTE my_insert USING 'usa'
```

次の例では、準備済みステートメント my_unload 内の productid パラメータの数値を提供しています。

```
EXECUTE my_unload USING 12
```

DEALLOCATE PREPARE

現在のワークグループの準備済みステートメントのリストから、指定された名前を持つ準備済みステートメントを削除します。

構文

```
DEALLOCATE PREPARE statement_name
```

statement_name は、削除される準備済みステートメントの名前です。

例

以下の例では、現在のワークグループから my_select1 準備済みステートメントを削除しています。

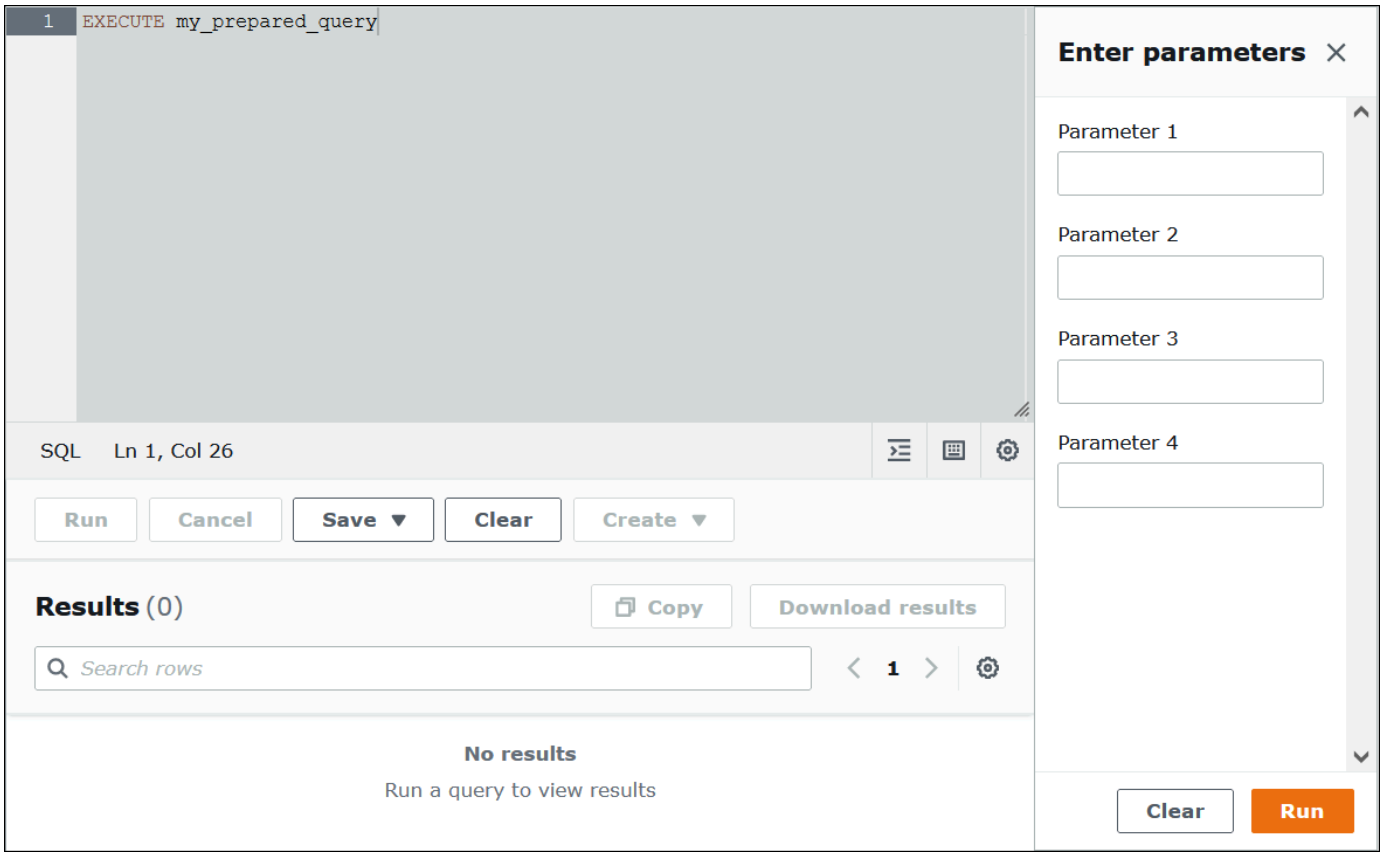
```
DEALLOCATE PREPARE my_select1
```

Athena コンソールにおける USING 句なしでのプリペアドステートメントの実行

クエリエディタで構文 `EXECUTE` *`prepared_statement`* を使用して既存のプリペアドステートメントを実行すると、Athena は [Enter parameters] (パラメータを入力) ダイアログボックスを開き、`EXECUTE ... USING` ステートメントの `USING` 句に通常含まれる値を入力できるようにします。

[Enter parameters] (パラメータを入力) ダイアログボックスを使用してプリペアドステートメントを実行するには

1. クエリエディタでは、構文 `EXECUTE prepared_statement USING` *`value1`*`,` *`value2`* `...` を使用する代わりに、構文 `EXECUTE` *`prepared_statement`* を使用します。
2. [Run] (実行) を選択します。[Enter parameters] (パラメータを入力) ダイアログボックスが表示されます。



3. [Execution parameters] (実行パラメータ) ダイアログボックスに値を順番に入力します。クエリの元のテキストは表示されないため、各位置パラメータの意味を覚えておくか、プリペアドステートメントを参照できるようにしておく必要があります。
4. [Run] (実行) を選択します。

AWS CLI を使用したプリペアドステートメントの作成

AWS CLI を使用してプリペアドステートメントを作成するには、次のいずれかの athena コマンドを使用できます。

- create-prepared-statement コマンドを使用して、実行パラメータを持つクエリステートメントを指定します。
- start-query-execution コマンドを使用して、PREPARE 構文を使用するクエリ文字列を指定します。

create-prepared-statement の使用

次の例のように、create-prepared-statement コマンドにおいて、query-statement 引数でクエリテキストを定義します。

```
aws athena create-prepared-statement
--statement-name PreparedStatement1
--query-statement "SELECT * FROM table WHERE x = ?"
--work-group athena-engine-v2
```

start-query-execution および PREPARE 構文の使用

start-query-executionコマンドを実行します。次の例のように、PREPARE ステートメントを query-string 引数に含めます。

```
aws athena start-query-execution
--query-string "PREPARE PreparedStatement1 FROM SELECT * FROM table WHERE x = ?"
--query-execution-context '{"Database": "default"}'
--result-configuration '{"OutputLocation": "s3://DOC-EXAMPLE-BUCKET/..."}'
```

AWS CLI を使用したプリペアドステートメントの実行

AWS CLI を使用してプリペアドステートメントを実行するために、次のいずれかの方法を使用してパラメータの値を指定できます。

- execution-parameters 引数を使用します。
- query-string 引数で EXECUTE ... USING SQL 構文を使用します。

実行パラメータ引数の使用

このアプローチでは、start-query-execution コマンドを使用して、query-string 引数で既存のプリペアドステートメントの名前を指定します。その後、execution-parameters 引数で実行パラメータの値を指定します。次の例は、この方法を示しています。

```
aws athena start-query-execution
--query-string "Execute PreparedStatement1"
--query-execution-context "Database"="default"
--result-configuration "OutputLocation"="s3://DOC-EXAMPLE-BUCKET/..."
--execution-parameters "1" "2"
```

EXECUTE ..。USING SQL 構文の使用

EXECUTE ... USING 構文を使用して既存のプリペアドステートメントを実行するには、次の例のように、start-query-execution コマンドを使用して、プリペアドステートメントの名前とパラメータ値の両方を query-string 引数に含めます。

```
aws athena start-query-execution
--query-string "EXECUTE PreparedStatement1 USING 1"
--query-execution-context '{"Database": "default"}'
--result-configuration '{"OutputLocation": "s3://DOC-EXAMPLE-BUCKET/..."}'
```

準備済みステートメントのリスト表示

特定のワークグループの準備済みステートメントをリスト表示するには、Athena の AWS CLI コマンド list-prepared-statements または Athena API のアクション ListPreparedStatements を使用できます。--work-group パラメータは必須です。

```
aws athena list-prepared-statements --work-group primary
```

追加リソース

「AWS ビッグデータブログ」の次の関連記事を参照してください。

- Improve reusability and security using Amazon Athena parameterized queries (Amazon Athena のパラメータ化されたクエリを使用して、再利用性とセキュリティを向上させる)
- Use Amazon Athena parameterized queries to provide data as a service (Amazon Athena のパラメータ化されたクエリを使用して、Data as a Service (DaaS) を提供する)

コストベースオプティマイザーの使用

Athena SQL のコストベースオプティマイザー (CBO) 機能を使用してクエリを最適化できます。オプションとして、Athena がテーブルを収集するようにリクエストするか、AWS Glue にあるいずれかのテーブルの列レベル統計を収集するようにリクエストできます。クエリのすべてのテーブルに統計がある場合、Athena はその統計を使用して最もパフォーマンスが高いと判断した実行プランを作成します。クエリオプティマイザーは、統計モデルに基づいて代替プランを計算し、クエリを最も速く実行できると思われるプランを選択します。

AWS Glue テーブルの統計は収集されて AWS Glue Data Catalog に保存され、Athena で利用可能になるため、クエリプランニングおよび実行が改善されます。これらの統計は、Parquet、ORC、JSON、ION、CSV、XML などのファイルタイプの、null 値、最大値、最小値といった個別の値の個数などの列レベル統計です。Amazon Athena は、クエリ処理のできるだけ早い段階で最も制限の厳しいフィルタを適用することにより、これらの統計を使用してクエリを最適化します。このフィルタリングにより、メモリ使用量、ならびにクエリ結果を配信するために読み取る必要のあるレコード数が制限されます。

CBO と併せて、Athena はルールベースオプティマイザー (RBO) と呼ばれる機能を使用します。RBO は、クエリのパフォーマンス向上が期待されるルールを自動的に適用します。RBO の変換はクエリプランの簡略化を目的としているため、一般的にメリットがあります。ただし、RBO はコスト計算やプラン比較を行わないため、クエリが複雑になると RBO が最適なプランを作成することが難しくなります。

このため、Athena は RBO および CBO の両方を使用してクエリを最適化します。Athena はクエリ実行を改善する機会を特定した後、最適なプランを作成します。実行プラン内容の詳細については、「SQL クエリの実行プランの表示」を参照してください。CBO の仕組みの詳細な論議については、AWS ビッグデータの「コストベースオプティマイザーのブログ記事」を参照してください。

AWS Glue Catalog テーブルの統計を生成するには、Athena コンソール、AWS Glue コンソール、AWS Glue API を使用できます。Athena は AWS Glue Catalog と統合されているため、Amazon Athena からクエリを実行すると、対応するクエリパフォーマンスが自動的に向上します。

考慮事項と制約事項

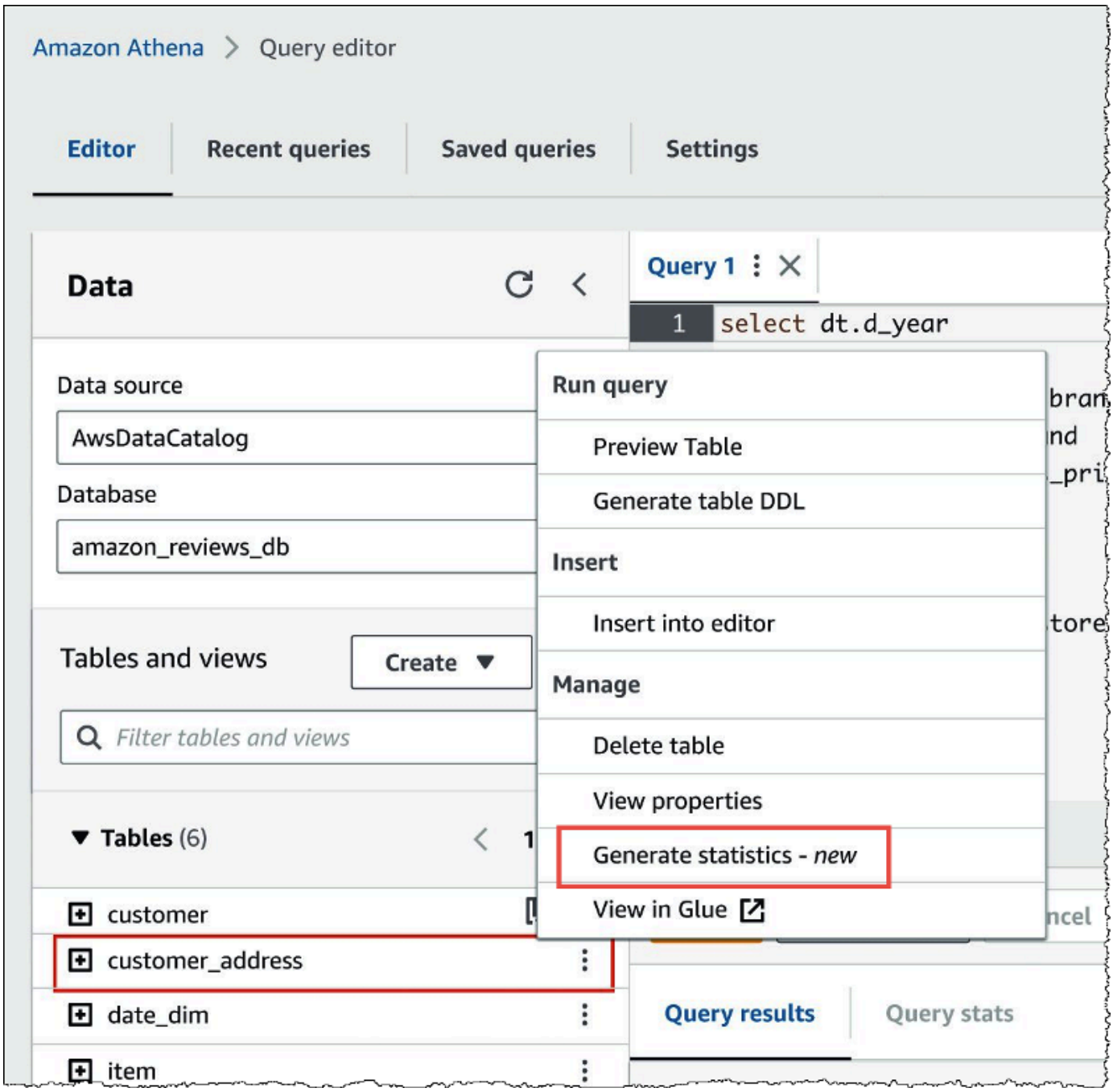
- テーブルタイプ — 現在、Athena の CBO 機能は AWS Glue Data Catalog に含まれている Hive テーブルのみをサポートしています。
- Athena for Spark — CBO 機能は Athena for Spark では利用できません。
- 料金 — 料金の詳細については、「AWS Glue 料金表ページ」を参照してください。

Athena コンソールを使用したテーブル統計の生成

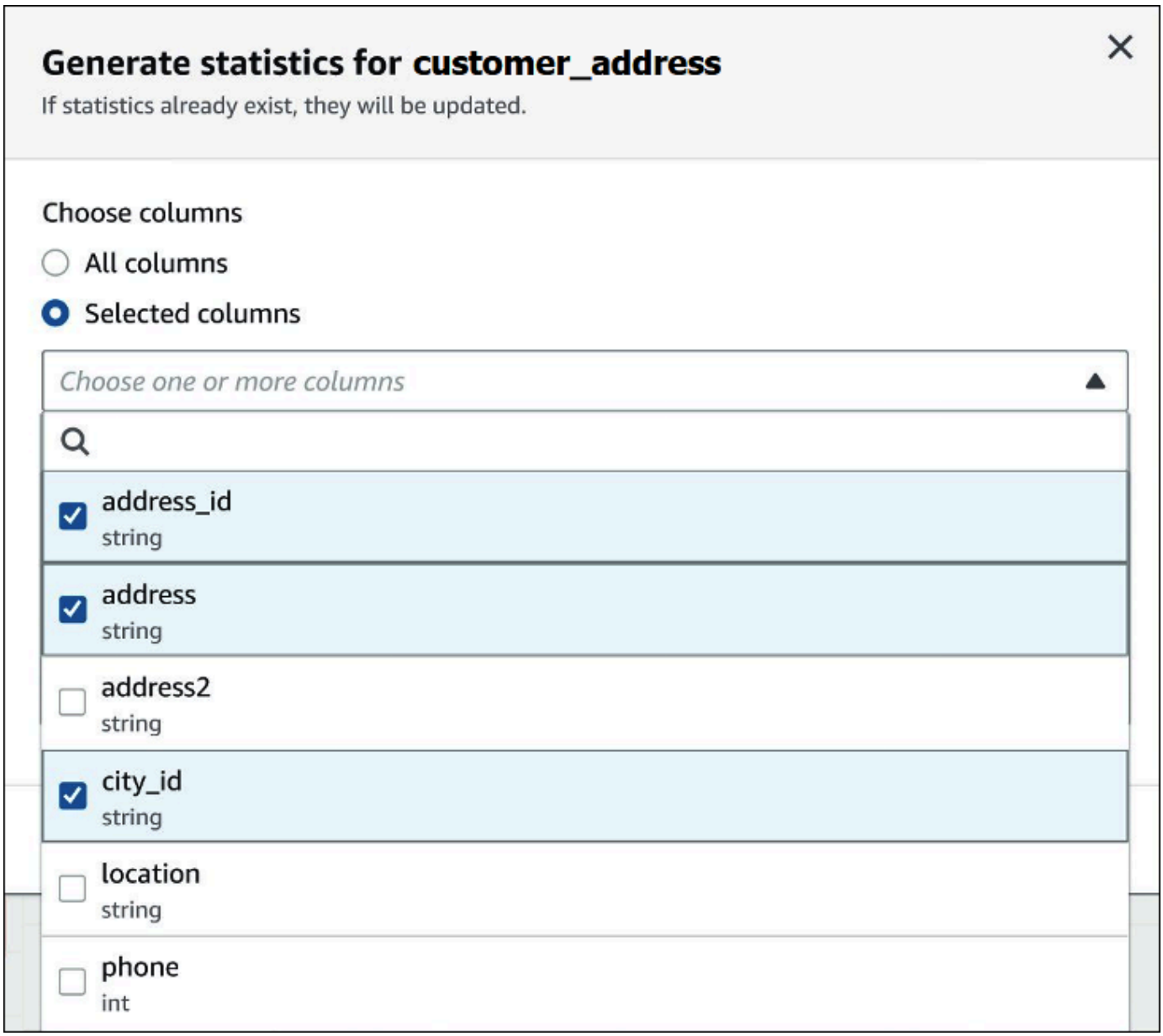
このセクションでは、Athena コンソールを使用してテーブルまたは AWS Glue のテーブルに列レベル統計を生成する方法について説明します。AWS Glue を使用してテーブル統計を生成する詳細については、「AWS Glue デベロッパーガイド」の「列統計の使用」を参照してください。

Athena コンソールを使用してテーブルの統計を生成する方法

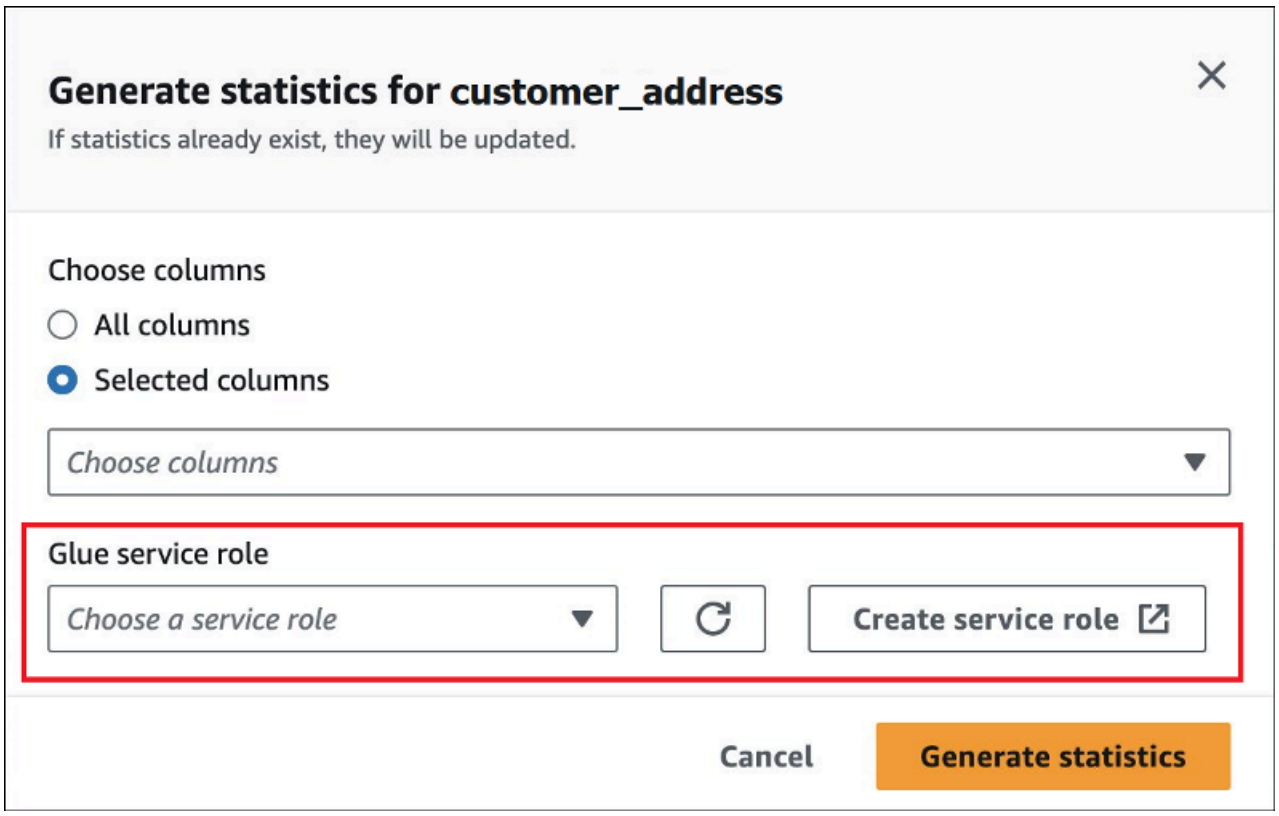
1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. Athena クエリエディタの [テーブル] リストで、目的のテーブルにある縦 3 つの点を選択し、[統計の生成] を選択します。



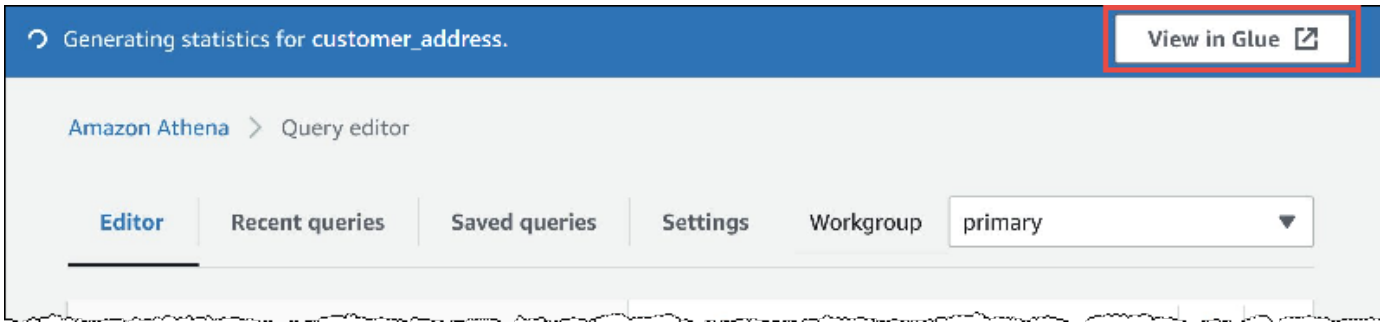
3. [統計の生成] ダイアログボックスで、[すべての列] を選択してテーブルにあるすべての列の統計を生成するか、[選択した列] を選択して特定の列を選択します。[全ての列] がデフォルト設定です。



4. [AWS Glue サービスロール] については、既存のサービスロールを作成または選択し、AWS Glue が統計を生成する許可を付与します。AWS Glue サービスロールには、テーブルのデータを含む Amazon S3 バケットへの `S3:GetObject` 権限も必要です。



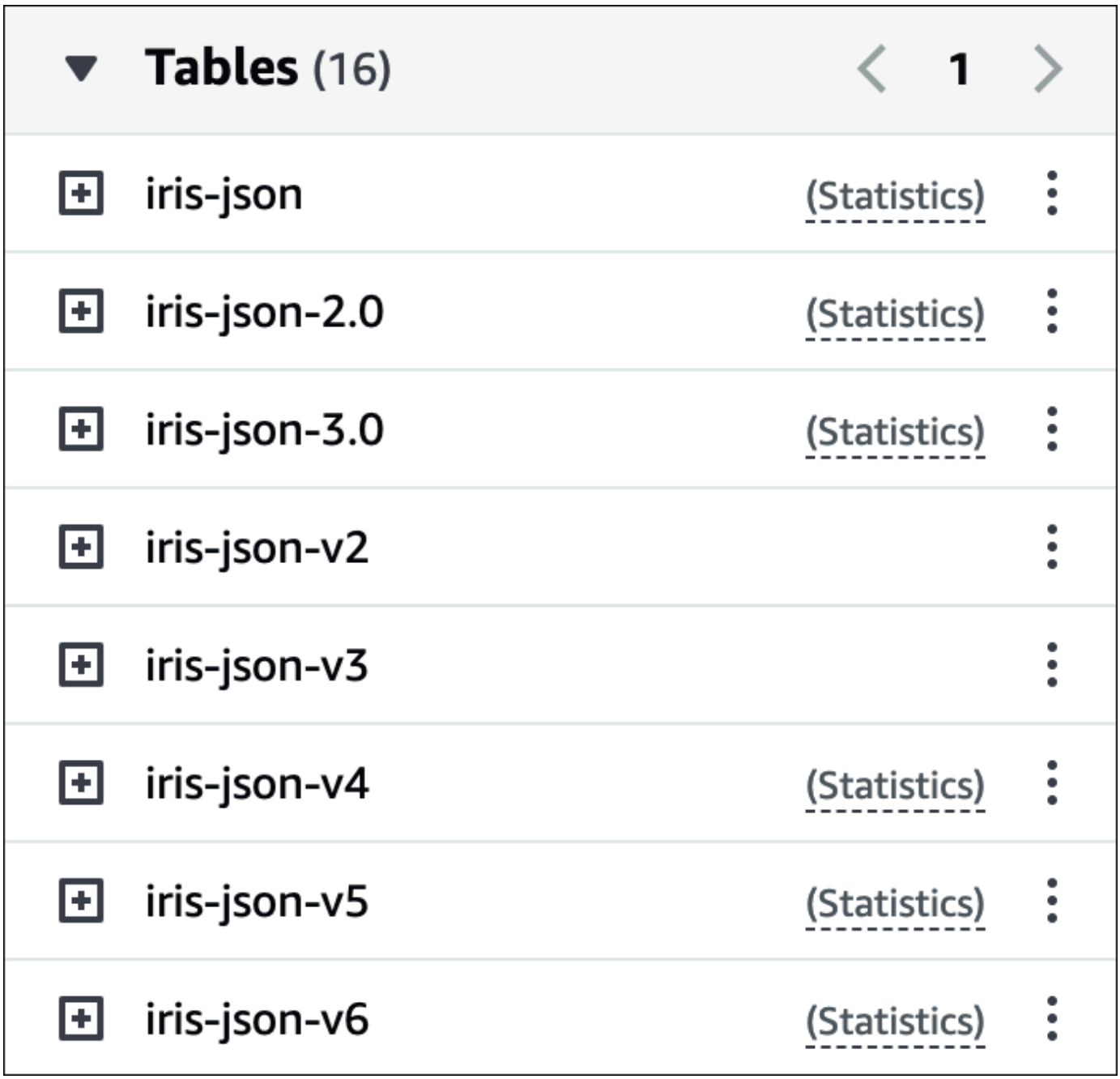
5. [統計の生成] を選択します。[*table_name* の統計を生成中] 通知バナーがタスクステータスを表示します。



6. AWS Glue コンソールに詳細を表示するには、[Glue で表示] を選択します。

   AWS Glue コンソールで統計の表示に関する詳細については、「AWS Glue デベロッパーガイド」の「列の統計の表示」を参照してください。

7. 統計が生成された後、次の画像のように、統計を含むテーブルと列には括弧内に [統計] という単語が表示されます。



クエリを実行すると、Athena は統計が生成されたテーブルおよび列に対し、コストベースの最適化が実行されるようになりました。

追加リソース

その他の情報については、次のリソースを参照してください。

S3 Express One Zone データのクエリ

Amazon S3 Express One Zone ストレージクラスは、1 桁のミリ秒単位の応答時間を実現する高性能の Amazon S3 ストレージクラスです。そのため、1 秒あたり数十万のリクエストを実行してデータに頻繁にアクセスするアプリケーションに役立ちます。

S3 Express One Zone は、同じアベイラビリティーゾーン内でデータをレプリケートして保存し、速度とコストを最適化します。これは、AWS リージョン 内の 3 つ以上の AWS アベイラビリティーゾーンにデータを自動的にレプリケートする Amazon S3 リージョンストレージクラスとは異なります。

詳細については、「Amazon S3 ユーザーガイド」の「What is S3 Express One Zone?」を参照してください。

前提条件

開始する前に、以下の条件が満たされていることを確認してください。

- Athena エンジンバージョン 3 — Athena SQL で S3 Express One Zone を使用するには、Athena エンジンバージョン 3 を使用するようにワークグループを設定する必要があります。
- S3 Express One ゾーンのアクセス許可 — S3 Express One Zone が Amazon S3 オブジェクトに対して `GET`、`LIST`、`PUT` などのアクションを呼び出すと、ストレージクラスがユーザーに代わって `CreateSession` を呼び出します。このため、IAM ポリシーでは `s3express:CreateSession` アクションを許可する必要があります。これにより、Athena は対応する API オペレーションを呼び出すことができます。

考慮事項と制約事項

Athena で S3 Express 1 Zone をクエリする際は、以下の点を考慮してください。

- S3 Express One Zone バケットは `SSE_S3` 暗号化のみをサポートします。Athena のクエリ結果は、ワークグループ設定で選択したクエリ結果を暗号化するオプションに関係なく、`SSE_S3` 暗号化を使用して書き込まれます。この制限には、`CREATE TABLE AS` (CTAS) や `INSERT INTO` ステートメントなど、Athena が S3 Express One Zone バケットにデータを書き込むすべてのシナリオが含まれます。
- S3 Express One Zone のデータにテーブルを作成する場合、AWS Glue クローラーはサポートされていません。

- `MSCK REPAIR TABLE` ステートメントはサポートされていません。回避方法として、ALTER TABLE ADD PARTITION を使用します。
- 次のファイルおよびテーブル形式はサポートされていないか、サポートが制限されています。リストにないが、Athena でサポートされているフォーマット (Parquet、ORC、JSON など) であれば、S3 Express One Zone ストレージでの使用もサポートされています。

ファイルまたはテーブル形式	制限
Apache Avro	サポートされていません
CloudTrail ログ	サポートされていません
Apache Hudi	サポートされていません
Amazon Ion	サポートされていません
Logstash ログ	サポートされていません
Apache WebServer ログ	サポートされていません
Delta Lake	DDL はサポートされていません。ダミースキーマを使用して Delta Lake テーブルを作成する方法については、「Delta Lake メタデータの同期」を参照してください。テーブルに対する `SELECT` クエリはサポートされています。

開始

Athena を使用すると、S3 Express One Zone のデータを簡単にクエリすることができます。使用を開始するには、以下の手順を実行します。

Athena SQL を使用して S3 Express One Zone のデータをクエリするには

1. データを S3 Express One Zone のストレージに移行します。詳細については、「Amazon S3 ユーザーガイド」の「オブジェクトのストレージクラスを設定」を参照してください。

2. Athena の CREATE TABLE ステートメントを使用して AWS Glue Data Catalog でデータをカタログ化します。Athena でのテーブルの作成に関する詳細については、「Athena でのテーブルの作成」と CREATE TABLE ステートメントを参照してください。
3. (オプション) Amazon S3 ディレクトリバケットを使用するように Athena ワークグループのクエリ結果の場所を設定します。Amazon S3 ディレクトリバケットは一般的なバケットよりもパフォーマンスが高く、一貫した 1 桁ミリ秒のレイテンシーを必要とするワークロードまたはパフォーマンス重視のアプリケーション向けに設計されています。詳細については、「Amazon S3 ユーザーガイド」の「Directory buckets overview」を参照してください。

復元された Amazon S3 Glacier オブジェクトへのクエリ

Athena を使用して、S3 Glacier Flexible Retrieval (以前の Glacier) および S3 Glacier Deep Archive Amazon S3 ストレージクラスから復元されたオブジェクトをクエリできます。この機能はテーブルごとに有効にする必要があります。クエリを実行する前にテーブルでこの機能を有効にしない場合、Athena はクエリ実行中にテーブルの S3 Glacier Flexible Retrieval オブジェクトと S3 Glacier Deep Archive オブジェクトをすべてスキップします。

考慮事項と制約事項

- 復元された Amazon S3 Glacier オブジェクトのクエリは、Athena エンジンバージョン 3 でのみサポートされています。
- この機能は Apache Hive テーブルでのみサポートされています。
- Athena はオブジェクトを復元しないため、データをクエリする前にオブジェクトを復元する必要があります。

テーブルを設定して復元されたオブジェクトを使用する

復元されたオブジェクトをクエリに含めるように Athena テーブルを設定するには、`read_restored_glacier_objects` テーブルプロパティを `true` に設定する必要があります。これを行うには、Athena クエリエディタまたは AWS Glue コンソールを使用できます。また、AWS Glue CLI、AWS Glue API、または AWS Glue SDK も使用できます。

Athena クエリエディタの使用

Athena では、次の例のように、ALTER TABLE SET TBLPROPERTIES コマンドを使用してテーブルプロパティを設定できます。

```
ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'true')
```

AWS Glue コンソールを使用する場合

AWS Glue コンソールで、次の手順を実行して、read_restored_glacier_objects テーブルプロパティを追加します。

AWS Glue コンソールでテーブルプロパティを設定するには

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. 次のいずれかを行います。
   - [データカタログに移動] を選択します。
   - ナビゲーションペインで、[データカタログログテーブル] を選択します。
3. [テーブル] ページのテーブルのリストで、編集するテーブルのリンクを選択します。
4. [Actions] (アクション)、[Edit table] (テーブルの編集) の順に選択します。
5. [テーブルを編集] ページの [テーブルプロパティ] セクションで、以下のキーと値のペアを追加します。
   - [キー] に、read_restored_glacier_objects を追加します。
   - [Value (値)] に「true」と入力します。
6. [Save] を選択します。

AWS CLI の使用

AWS CLI では、テーブルを再定義するために AWS Glue update-table コマンドとその --table-input 引数を使用でき、そうすることで read_restored_glacier_objects プロパティが追加されます。--table-input 引数では、Parameters の構造を使用して read_restored_glacier_objects プロパティと true の値を指定します --table-input の引数にはスペースがあってはならず、二重引用符を エスケープするにはバックスラッシュを使用する必要があることに注意してください。以下の例では、*my_database* および *my_table* を独自のデータベース名とテーブル名に置き換えます。

```
aws glue update-table \
   --database-name my_database \
```

```
    --table-input={\"Name\":\"my_table\",\"Parameters\":{\"read_restored_glacier_objects
\":\"true\"}}
```

> **Important**
>
> AWS Glue update-table コマンドは上書きモードで動作します。これは、このコマンドが既存のテーブル定義を table-input パラメータ指定の新しい定義に置き換えることを意味します。このため、read_restored_glacier_objects プロパティを追加するときは、テーブルに含めるすべてのフィールドも table-input パラメータ内に指定するようにしてください。

スキーマ更新の処理

このセクションでは、さまざまなデータ形式に対するスキーマ更新の処理に関するガイダンスを提供します。Athena はスキーマオンリード (schema-on-read) のクエリエンジンです。これは、Athena でテーブルを作成する場合、Athena がデータの読み込み時にスキーマを適用することを意味します。基盤となるデータに変更または書き直しが行われることはありません。

テーブルスキーマの変更が予想される場合は、ニーズに適したデータ形式で作成することを検討します。目的は、進化するスキーマに対して既存の Athena クエリを再利用し、パーティションがあるテーブルをクエリするときのスキーマの不一致エラーを回避することです。

これらの目的を達成するには、次のトピックの表に基づいてテーブルのデータ形式を選択してください。

トピック

概要: Athena における更新とデータ形式

以下の表は、データストレージ形式と、それらがサポートするスキーマ操作をまとめたものです。この表を使用して、スキーマがいずれは変更されるとしても、Athena クエリを引き続き使用できるようにする形式を選択するために役立ててください。

この表では、Parquet と ORC の列形式が、列へのデフォルトのアクセス方法が異なるものであることに注意してください。デフォルトで、Parquet は名前で列にアクセスし、ORC はインデックス (序数値) で列にアクセスします。このため、Athena はテーブルの作成時に定義される SerDe プロパティを提供して、列へのデフォルトのアクセス方法を切り替えます。これによって、スキーマの進化における柔軟性が大きく向上します。

Parquet では、`parquet.column.index.access` プロパティを `true` に設定することができます。これは、列の序数を使用するように列へのアクセス方法を設定します。このプロパティを `false` に設定すると、列へのアクセス方法が列名を使用するように変更されます。同様に、ORC では `orc.column.index.access` プロパティを使用して列へのアクセス方法を制御します。詳細については、「Parquet および ORC におけるインデックスアクセス」を参照してください。

CSV と TSV を使用すると、列の並べ替えやテーブルの先頭に列を追加する以外のすべてのスキーマ操作を実行できます。例えば、スキーマの変更で列の名前を変更するだけで、列の名前を削除する必要がない場合は、CSV または TSV でテーブルを作成することができます。列を削除する必要がある場合は、CSV または TSV を使用せず、その他のサポートされている形式のいずれか (できれば、Parquet または ORC などの列形式) を使用します。

Athena におけるスキーマ更新とデータ形式

予想されるスキーマ更新のタイプ	概要	CSV (ヘッダー有りまたはヘッダーなし) および TSV	JSO	AVF	PARQUE 名前で読み込む (デフォルト)	PARQUE インデックスで読み込む	ORC: インデックスで読み込む (デフォルト)	ORC: 名前で読み込む
列の名前変更	データを CSV および TSV に保存するか、ORC および Parquet でインデックスに読み込んだ場合に保存します。	Y	N	N	N	Y	Y	N

予想されるスキーマ更新のタイプ	概要	CSV (ヘッダー有りまたはヘッダーなし) および TSV	JSO	AVF	PARQUE 名前で読み込む (デフォルト)	PARQUE インデックスで読み込む	ORC: インデックスで読み込む (デフォルト)	ORC: 名前で読み込む
テーブルの先頭または中間に列を追加する	データを JSON、AVRO に保存するか、Parquet および ORC で名前に読み込んだ場合に保存します。CSV や TSV を使用しないでください。	N	Y	Y	Y	N	N	Y
テーブルの末尾に列を追加する	CSV または TSV、JSON、AVRO、ORC、または Parquet 形式でデータを保存します。	Y	Y	Y	Y	Y	Y	Y
列の削除	データを JSON、AVRO に保存するか、Parquet および ORC で名前に読み込んだ場合に保存します。CSV や TSV を使用しないでください。	N	Y	Y	Y	N	N	Y

予想されるスキーマ更新のタイプ	概要	CSV (ヘッダー有りまたはヘッダーなし) および TSV	JSO	AVI	PARQUE 名前で読み込む (デフォルト)	PARQUE インデックスで読み込む	ORC: インデックスで読み込む (デフォルト)	ORC: 名前で読み込む
列の順序変更	データを AVRO、JSON に保存するか、ORC および Parquet で名前に読み込んだ場合に保存します。	N	Y	Y	Y	N	N	Y
列のデータ型の変更	任意の形式でデータを保存します。ただし、Athena でクエリをテストして、データ型に互換性があることを確認するようにしてください。Parquet および ORC のデータ型を変更した場合は、パーティションされたテーブルでのみ機能します。	Y	Y	Y	Y	Y	Y	Y

Parquet および ORC におけるインデックスアクセス

PARQUET および ORC は、インデックス引または名前で読み込み可能な列データストレージ形式です。これらの形式のいずれかでデータを保存すると、スキーマですべてのオペレーションを実行し、スキーマの不一致エラーを生じることなく Athena でクエリを実行できるようになります。

- Athena は、`SERDEPROPERTIES ( 'orc.column.index.access'='true')` で定義されているように、デフォルトでインデックスによる ORC の読み込みを行います。詳細については、「ORC: インデックスで読み込む」を参照してください。
- Athena は、`SERDEPROPERTIES ( 'parquet.column.index.access'='false')` で定義されているとおり、デフォルトで名前による Parquet の読み込みを行います。詳細については、「Parquet: 名前で読み込む」を参照してください。

これらはデフォルトであるため、これらの SerDe プロパティを `CREATE TABLE` クエリで使用するのはオプションで、暗黙的に使用されます。これらを使用すると、、他のオペレーションを防ぎながら、スキーマの更新オペレーションを実行することができます。これらのオペレーションを有効にするには、別の `CREATE TABLE` クエリを実行し、SerDe の設定を変更します。

> **Note**
>
> SerDe のプロパティは、各パーティションに自動的に伝播されません。各パーティションに SerDe プロパティを設定するには、`ALTER TABLE ADD PARTITION` ステートメントを使用します。このプロセスを自動化するには、`ALTER TABLE ADD PARTITION` ステートメントを実行するスクリプトを記述します。

以下のセクションで、これらのケースについて詳しく説明します。

ORC: インデックスで読み込む

ORC 内のテーブルは、デフォルトでインデックスで読み取られます。これは、次の構文で定義されます。

```
WITH SERDEPROPERTIES (
  'orc.column.index.access'='true')
```

インデックスで読み込むと、列の名前を変更できます。ただし、列の削除やテーブルの中間での追加ができなくなります。

ORC を名前で読み込み、テーブルの中間に列を追加したり、ORC の列を削除したりするには、`orc.column.index.access` ステートメントで、SerDe プロパティ `false` を `CREATE TABLE` に設定します。この設定では、列の名前を変更する機能が失われます。

> **Note**
>
> Athena エンジンバージョン 2 では、ORC テーブルが名前で読み込むように設定されている場合、Athena では ORC ファイル内のすべての列の名前を小文字にする必要があります。Apache Spark は ORC ファイルを生成するときにフィールド名を小文字にしないため、生成されたデータを Athena で読み込むことができない可能性があります。この問題を回避するためには、列の名前を小文字に変更するか、Athena エンジンバージョン 3 を使用します。

次の例は、ORC を変更して名前で読み込む方法を示しています。

```
CREATE EXTERNAL TABLE orders_orc_read_by_name (
   `o_comment` string,
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderpriority` string,
   `o_orderstatus` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_orderdate` string
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'orc.column.index.access'='false')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_orc/';
```

Parquet: 名前で読み込む

Parquet のテーブルはデフォルトで名前で読み取られます。これは、次の構文で定義されます。

```
WITH SERDEPROPERTIES (
```

```
  'parquet.column.index.access'='false')
```

名前で読み込むと、テーブルの中間に列を追加したり、列を削除したりできます。ただし、列の名前を変更することができなくなります。

Parquet にインデックスで読み取らせて列の名前を変更できるようにするには、`parquet.column.index.access` SerDe プロパティを `true` に設定してテーブルを作成する必要があります。

更新タイプ

このトピックでは、実際にデータを変更せずに CREATE TABLE ステートメントのスキーマに加えられるいくつかの変更について説明します。スキーマ更新の各タイプがレビューされ、Athena でこれらの使用が許可されるデータ形式を指定します。スキーマを更新するには、ALTER TABLE コマンドを使用できる場合もあれば、既存のテーブルを実際に変更しない場合もあります。代わりに、元の CREATE TABLE テートメントで使用したスキーマを変更する新しい名前のテーブルを作成します。

- テーブルの先頭または中間への列の追加
- テーブルの末尾への列の追加
- 列の削除
- 列名の変更
- 列の順序の変更
- 列のデータ型の変更

期待されるスキーマの進化方法に応じて、Athena クエリの使用を継続するために互換性のあるデータ形式を選択します。

CSV および Parquet の 2 つの形式で存在する `orders` テーブルからの注文情報を読み取るアプリケーションを考えます。

以下の例では、Parquet でテーブルを作成します。

```
CREATE EXTERNAL TABLE orders_parquet (
   `orderkey` int,
   `orderstatus` string,
   `totalprice` double,
   `orderdate` string,
```

```
   `orderpriority` string,
   `clerk` string,
   `shippriority` int
) STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_ parquet/';
```

以下の例では、CSV で同じテーブルを作成します。

```
CREATE EXTERNAL TABLE orders_csv (
   `orderkey` int,
   `orderstatus` string,
   `totalprice` double,
   `orderdate` string,
   `orderpriority` string,
   `clerk` string,
   `shippriority` int
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_csv/';
```

以下のセクションでは、これらのテーブルに対する更新がどのように Athena クエリに影響するかを検証します。

テーブルの先頭または中間への列の追加

列の追加は最も頻繁なスキーマの変更の 1 つです。たとえば、新しいデータでテーブルをエンリッチ化するために、新しい列を追加することがあります。または、既存の列のソースが変更された場合に新しい列を追加し、この列の以前のバージョンを保持してそれに依存するアプリケーションを調整することもあります。

テーブルの先頭または中間に列を追加し、既存のテーブルに対してクエリを実行するには、SerDe プロパティが名前で読み取るように設定されている場合は、AVRO、JSON、および Parquet と ORC を使用します。詳細については、Parquet および ORC におけるインデックスアクセス を参照してください。

これらの形式は順序に依存するため、CSV および TSV のテーブルの先頭または中間に列を追加しないでください。このような場合に列を追加すると、パーティションのスキーマが変更されたときにスキーマの不一致エラーが発生します。

次の例では、JSON データに基づいてテーブルの中央に `o_comment` 列を追加する新しいテーブルを作成します。

```
CREATE EXTERNAL TABLE orders_json_column_addition (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_comment` string,
   `o_totalprice` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_json/';
```

テーブルの末尾への列の追加

Parquet、ORC、Avro、JSON、CSV、および TSV などの Athena がサポートする形式のいずれかでテーブルを作成する場合は、ALTER TABLE ADD COLUMNS ステートメントを使用して、既存の列の後、かつパーティション列の前の位置に列を追加できます。

次の例では、パーティション comment 列の前の orders_parquet テーブルの末尾に列を追加します。

```
ALTER TABLE orders_parquet ADD COLUMNS (comment string)
```

> **Note**
>
> ALTER TABLE ADD COLUMNS を実行した後で Athena クエリエディタに新しいテーブル列を表示するには、エディタのテーブルリストを手動で更新してから、テーブルをもう一度展開します。

列の削除

列に含まれるデータがなくなった場合、あるいは列に含まれているデータへのアクセスを制限する場合に、これらの列を削除する必要があることがあります。

- 名前で読み込まれている場合、JSON、Avro、および Parquet、ORC でテーブルから列を削除することができます。詳細については、Parquet および ORC におけるインデックスアクセス を参照してください。

- 既に Athena で作成したテーブルを保持する場合は、CSV および TSV のテーブルから列を削除することは推奨されません。列を削除するとスキーマが壊れ、削除された列がない状態でテーブルを再作成しなければならなくなります。

この例では、Parquet のテーブルから `totalprice` 列を削除して、クエリを実行します。Athena では、デフォルトで Parquet の読み取りが名前で行われます。名前での読み取りを指定する SERDEPROPERTIES 設定が省略されているのはこのためです。スキーマを変更しても、次のクエリは成功することに注意してください。

```
CREATE EXTERNAL TABLE orders_parquet_column_removed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_comment` string
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_parquet/';
```

列名の変更

綴りを修正する、列名をより分かりやすくする、または列の順序変更を回避するために、テーブルで列の名前変更が必要となる場合があります。

データを CSV および TSV に保存する場合は列の名前を変更できます。また、インデックスで読み取るように設定されている場合は、Parquet および ORC に保存できます。詳細については、Parquet および ORC におけるインデックスアクセス を参照してください。

Athena は CSV と TSV のデータをスキーマの列順に読み取り、それらを同じ順序で返します。Athena は、列にデータをマップするために列名を使用しません。Athena クエリを破損することなく CSV または TSV で列名を変更できるのはこのためです。

列名を変更する戦略の 1 つは、同じ基盤データに基づく新しいテーブルを、新しい列名を使用して作成することです。以下の例は、orders_parquet_column_renamed という名前の新しい orders_parquet テーブルを作成します。この例は、列 `o_totalprice` の名前を `o_total_price` に変更してから、Athena でクエリを実行します。

```
CREATE EXTERNAL TABLE orders_parquet_column_renamed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_total_price` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_comment` string
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_parquet/';
```

Parquet テーブルの場合、次のクエリは実行されますが、列がインデックスではなく名前でアクセスされているため (Parquet のデフォルト)、名前が変更された列にはデータが表示されません。

```
SELECT *
FROM orders_parquet_column_renamed;
```

CSV のテーブルでのクエリも類似しています。

```
CREATE EXTERNAL TABLE orders_csv_column_renamed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_total_price` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_comment` string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_csv/';
```

CSV テーブルの場合、次のクエリを実行すると、名前が変更された列を含むすべての列でデータが表示されます。

```
SELECT *
FROM orders_csv_column_renamed;
```

列の順序の変更

デフォルトで名前で読み取る JSON や Parquet など、名前で読み取る形式のデータを含むテーブルの列の順序を変更できます。必要に応じて、ORC を名前で読み取ることもできます。詳細については、Parquet および ORC におけるインデックスアクセス を参照してください。

次の例では、列の順序が違う新しいテーブルを作成します。

```
CREATE EXTERNAL TABLE orders_parquet_columns_reordered (
   `o_comment` string,
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderpriority` string,
   `o_orderstatus` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_orderdate` string
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_parquet/';
```

列のデータ型の変更

既存のタイプでは必要な量の情報を保持できなくなった場合は、別の列タイプを使用することをお勧めします。例えば、ID 列の値が INT データ型のサイズを超えているため、その BIGINT データ型を使用する必要があります。

列に別のデータ型を使用するときは、以下の点を考慮してください。

- ほとんどの場合、列のデータ型を直接変更することはできません。代わりに、Athena テーブルを再作成し、新しいデータ型で列を定義します。
- 一部のデータ型のみを他のデータ型として読み取ることができます。扱えるデータ型については、このセクションの表を参照してください。
- Parquet および ORC のデータでは、テーブルがパーティション分割されていない場合、列に異なるデータ型を使用することはできません。
- Parquet および ORC のパーティション分割されたテーブルでは、パーティションの列タイプが別のパーティションの列タイプと異なる場合があり、可能な場合は、Athena が望ましいタイプに CAST します。詳細については、パーティションがあるテーブルに対するスキーマの不一致エラーの回避 を参照してください。

- LazySimpleSerDE のみを使用して作成されたテーブルでは、ALTER TABLE REPLACE COLUMNS ステートメントを使用して既存の列を別のデータ型に置き換えることができますが、保持したい既存の列もすべてステートメントに再定義する必要があります。これを行わないと、削除されます。詳細については、「ALTER TABLE REPLACE COLUMNS」を参照してください。
- Apache Iceberg テーブルの場合のみ、ALTER TABLE CHANGE COLUMN ステートメントを使用して列のデータ型を変更できます。 ALTER TABLE REPLACE COLUMNS は Iceberg テーブルではサポートされていません。詳細については、「Iceberg テーブルスキーマの進化」を参照してください。

> **Important**
>
> データ型の変換を実行する前に、クエリをテストして検証することを強くお勧めします。Athena がターゲットのデータ型を使用できない場合、CREATE TABLE クエリは失敗する可能性があります。

次の表は、他のデータ型として扱われるデータ型の一覧です。

互換性のあるデータ型

元のデータ型	使用可能なターゲットデータ型
STRING	BYTE, TINYINT, SMALLINT, INT, BIGINT
BYTE	TINYINT, SMALLINT, INT, BIGINT
TINYINT	SMALLINT, INT, BIGINT
SMALLINT	INT, BIGINT
INT	BIGINT
FLOAT	DOUBLE

次の例では、元の orders_json テーブルの CREATE TABLE ステートメントを使用して、orders_json_bigint という新しいテーブルを作成します。新しいテーブルでは、`o_shippriority` 列のデータ型として INT の代わりに BIGINT が使用されます。

```
CREATE EXTERNAL TABLE orders_json_bigint (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_totalprice` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` BIGINT
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_json';
```

次のクエリは、元の SELECT クエリと同様に、データ型が変更される前に正常に実行されます。

```
Select * from orders_json
LIMIT 10;
```

パーティションがあるテーブルを更新する

Athena では、テーブルとそのパーティションが同じデータ形式を使用する必要がありますが、スキーマは異なる場合があります。新しいパーティションを作成するとき、このパーティションは通常テーブルのスキーマを継承します。時間の経過とともに、このスキーマが変わり始める場合があります。その理由には次のようなものがあります。

- テーブルのスキーマの場合、パーティションのスキーマはテーブルのスキーマと同期するために更新されません。
- AWS Glue クローラでは、異なるスキーマのパーティションからデータを検出することができます。これは、AWS Glue を使って Athena でテーブルを作成する場合、クローラの処理が終了した後で、テーブルのスキーマとそのパーティションのスキーマが異なる場合があることを意味します。
- AWS API を使用して、直接パーティションを追加した場合。

Athena は、テーブルが以下の制約を満たす場合に、パーティションがあるテーブルを正常に処理します。これらの制約が満たされない場合、Athena は「HIVE_PARTITION_SCHEMA_MISMATCH」エラーを発行します。

- 各パーティションのスキーマがテーブルのスキーマと互換性があること。

- テーブルのデータ形式が実行する更新のタイプを許可すること (追加、削除、列の順序変更あるいは列のデータ型の変更)。

  たとえば、CSV および TSV 形式では列の名前変更、新しい列のテーブル末尾への追加および型に互換性がある場合の列のデータ型の変更のみを行うことができ、列を削除することはできません。その他の形式では、列の追加あるいは削除、型に互換性がある場合の列のデータ型の別への変更ができます。詳細については、「概要: Athena における更新とデータ形式」を参照してください。

パーティションがあるテーブルに対するスキーマの不一致エラーの回避

Athena は、クエリの実行開始時に、各列のデータ型にテーブルとパーティション間での互換性があることをチェックすることによってテーブルのスキーマを検証します。

- Parquet および ORC のデータストレージタイプの場合、Athena は列名に依存し、列名ベースのスキーマ検証のためにそれらを使用します。これにより、Parquet および ORC のパーティションがあるテーブルの `HIVE_PARTITION_SCHEMA_MISMATCH` エラーが解消されます。(ORC では、名前でインデックスにアクセスするように SerDe プロパティが設定されている (`orc.column.index.access=FALSE`) 場合にこれが機能します。Parquet はデフォルトで、名前によるインデックスの読み取りを行います)。
- CSV、JSON、および Avro の場合、Athena はインデックスベースのスキーマ検証を使用します。これは、スキーマの不一致エラーが発生した場合、スキーマの不一致の原因となっているパーティションをドロップして作成し直し、Athena で失敗なくクエリを実行できるようにする必要があることを意味します。

Athena は、テーブルのスキーマとパーティションのスキーマを比較します。Athena の AWS Glue クローラで CSV、JSON、および AVRO でテーブルを作成した場合、クローラの処理が終了すると、テーブルのスキーマとそのパーティションは異なる場合があります。テーブルのスキーマとパーティションのスキーマ間に不一致がある場合、Athena でのクエリは、「'crawler_test.click_avro' is declared as type 'string', but partition 'partition_0=2017-01-17' declared column 'col68' as type 'double'."」に似たスキーマ検証エラーが原因で失敗します。

このようなエラーを回避する一般的な対処法は、エラーの原因となるパーティションを削除して、再作成することです。詳細については、「ALTER TABLE DROP PARTITION」および「ALTER TABLE ADD PARTITION」を参照してください。

配列のクエリ

Amazon Athena では、配列の作成、連結、異なるデータ型への変換を実行して、その後それらをフィルタリング、フラット化、および並び替えることができます。

トピック

配列の作成

Athena で配列リテラルを構築するには、角括弧 [ ] が後に続く ARRAY キーワードを使用し、カンマで区切られた配列要素を含めます。

例

次のクエリでは、4 つの要素を持つ 1 つの配列を作成します。

```
SELECT ARRAY [1,2,3,4] AS items
```

返される結果は次のとおりです。

```
+------------+
```

```
| items     |
+-----------+
| [1,2,3,4] |
+-----------+
```

次のクエリでは 2 つの配列を作成します。

```
SELECT ARRAY[ ARRAY[1,2], ARRAY[3,4] ] AS items
```

返される結果は次のとおりです。

```
+--------------------+
| items              |
+--------------------+
| [[1, 2], [3, 4]]   |
+--------------------+
```

互換性のある複数のタイプ間で選択した列から配列を作成するには、次の例に示すように、クエリを使用します。

```
WITH
dataset AS (
  SELECT 1 AS x, 2 AS y, 3 AS z
)
SELECT ARRAY [x,y,z] AS items FROM dataset
```

このクエリは以下を返します。

```
+-----------+
| items     |
+-----------+
| [1,2,3]   |
+-----------+
```

次の例では、2 つの配列を選択し、ウェルカムメッセージとして返します。

```
WITH
dataset AS (
  SELECT
    ARRAY ['hello', 'amazon', 'athena'] AS words,
```

```
    ARRAY ['hi', 'alexa'] AS alexa
)
SELECT ARRAY[words, alexa] AS welcome_msg
FROM dataset
```

このクエリは以下を返します。

```
+----------------------------------------+
| welcome_msg                            |
+----------------------------------------+
| [[hello, amazon, athena], [hi, alexa]] |
+----------------------------------------+
```

キーと値のペアの配列を作成するには、次の例に示すように、MAP 演算子を使用してキーの配列に続けて値の配列を使用します。

```
SELECT ARRAY[
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Bob', 'Smith', '40']),
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Jane', 'Doe', '30']),
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Billy', 'Smith', '8'])
] AS people
```

このクエリは以下を返します。

```
+-----------------------------------------------------------------------------------------------------------------
+
| people
               |
+-----------------------------------------------------------------------------------------------------------------
+
| [{last=Smith, first=Bob, age=40}, {last=Doe, first=Jane, age=30}, {last=Smith,
 first=Billy, age=8}] |
+-----------------------------------------------------------------------------------------------------------------
+
```

文字列と配列の連結

文字列の連結

2 つの文字列を連結するには、以下の例にあるように、二重パイプ || 演算子を使用することができます。

```
SELECT 'This' || ' is' || ' a' || ' test.' AS Concatenated_String
```

このクエリは以下を返します。

#	Concatenated_String
1	This is a test.

concat() 関数を使って同じ結果を得ることが可能です。

```
SELECT concat('This', ' is', ' a', ' test.') AS Concatenated_String
```

このクエリは以下を返します。

#	Concatenated_String
1	This is a test.

concat_ws() 関数を使用して、最初の引数で指定された区切り文字で文字列を連結できます。

```
SELECT concat_ws(' ', 'This', 'is', 'a', 'test.') as Concatenated_String
```

このクエリは以下を返します。

#	Concatenated_String
1	This is a test.

文字列データ型の 2 つの列をドットで連結するには、二重引用符を使用して 2 つの列を参照し、そのドットをハードコーディングされた文字列として一重引用符で囲みます。列が文字列データ型ではない場合、CAST("*column_name*" as VARCHAR) を使用して最初に列をキャストできます。

```
SELECT "col1" || '.' || "col2" as Concatenated_String
FROM my_table
```

このクエリは以下を返します。

#	Concatenated_String
1	*col1_string_value .col2_string_value*

配列の連結

同じテクニックを使用して配列を連結できます。

複数の配列を連結するには、二重パイプ || 演算子を使用します。

```
SELECT ARRAY [4,5] || ARRAY[ ARRAY[1,2], ARRAY[3,4] ] AS items
```

このクエリは以下を返します。

#	項目
1	[[4, 5], [1, 2], [3, 4]]

複数の配列を単一の配列に統合するには、二重パイプ演算子、または concat() 関数を使用します。

```
WITH
dataset AS (
  SELECT
    ARRAY ['Hello', 'Amazon', 'Athena'] AS words,
    ARRAY ['Hi', 'Alexa'] AS alexa
)
SELECT concat(words, alexa) AS welcome_msg
FROM dataset
```

このクエリは以下を返します。

#	welcome_msg
1	[Hello, Amazon, Athena, Hi, Alexa]

concat() とその他の文字列関数の詳細については、Trino ドキュメントの「String Functions and Operators」(文字列関数と演算子) を参照してください。

配列のデータ型の変換

配列内のデータをサポートされているデータ型に変換するには、CAST 演算子を CAST(value AS type) のように使用します。Athena は、すべてのネイティブ Presto データ型をサポートします。

```
SELECT
   ARRAY [CAST(4 AS VARCHAR), CAST(5 AS VARCHAR)]
AS items
```

このクエリは以下を返します。

```
+-------+
| items |
+-------+
| [4,5] |
+-------+
```

次の例では、キーと値のペアの要素を持つ配列を 2 つ作成し、この 2 つの配列を JSON に変換して連結します。

```
SELECT
   ARRAY[CAST(MAP(ARRAY['a1', 'a2', 'a3'], ARRAY[1, 2, 3]) AS JSON)] ||
   ARRAY[CAST(MAP(ARRAY['b1', 'b2', 'b3'], ARRAY[4, 5, 6]) AS JSON)]
AS items
```

このクエリは以下を返します。

```
+-------------------------------------------------+
| items                                           |
+-------------------------------------------------+
| [{"a1":1,"a2":2,"a3":3}, {"b1":4,"b2":5,"b3":6}] |
+-------------------------------------------------+
```

長さの確認

cardinality 関数は、次の例に示すように、配列の長さを返します。

```
SELECT cardinality(ARRAY[1,2,3,4]) AS item_count
```

このクエリは以下を返します。

```
+------------+
| item_count |
+------------+
| 4          |
+------------+
```

配列要素へのアクセス

配列要素にアクセスするには、[] 演算子を使用します。次の例のように、1 に最初の要素、2 に 2 番目の要素を指定し、以下同様に指定します。

```
WITH dataset AS (
SELECT
   ARRAY[CAST(MAP(ARRAY['a1', 'a2', 'a3'], ARRAY[1, 2, 3]) AS JSON)] ||
   ARRAY[CAST(MAP(ARRAY['b1', 'b2', 'b3'], ARRAY[4, 5, 6]) AS JSON)]
AS items )
SELECT items[1] AS item FROM dataset
```

このクエリは以下を返します。

```
+------------------------+
| item                   |
+------------------------+
| {"a1":1,"a2":2,"a3":3} |
+------------------------+
```

配列の特定の位置 (インデックス位置) にある要素にアクセスするには、element_at() 関数を使用して配列名とインデックス位置を指定します。

- インデックスが 0 より大きい場合は、配列の先頭から末尾へカウントした位置の要素がelement_at() から返されます。これは [] 演算子の動作と同じです。
- インデックスが 0 より小さい場合は、配列の末尾から先頭へカウントした位置の要素が element_at() から返されます。

次のクエリでは、配列 words を作成し、この配列の最初の要素 hello を first_word、2 番目の要素 amazon (配列の末尾からカウント) を middle_word、3 番目の要素 athena を last_word として選択します。

```
WITH dataset AS (
  SELECT ARRAY ['hello', 'amazon', 'athena'] AS words
)
SELECT
  element_at(words, 1) AS first_word,
  element_at(words, -2) AS middle_word,
  element_at(words, cardinality(words)) AS last_word
FROM dataset
```

このクエリは以下を返します。

```
+----------------------------------------+
| first_word  | middle_word | last_word  |
+----------------------------------------+
| hello       | amazon      | athena     |
+----------------------------------------+
```

ネストされた配列のフラット化

ネストされた配列を使用する場合、必要に応じて、ネストされた配列の要素を単一の配列に展開したり、配列を複数の行に展開したりすることがあります。

例

ネストされた配列の要素を単一の値配列にフラット化するには、flatten 関数を使用します。次のクエリは、配列の要素ごとに行を返します。

```
SELECT flatten(ARRAY[ ARRAY[1,2], ARRAY[3,4] ]) AS items
```

このクエリは以下を返します。

```
+-----------+
| items     |
+-----------+
| [1,2,3,4] |
+-----------+
```

配列を複数の行にフラット化するには、次の例に示すように、CROSS JOIN 演算子を UNNEST 演算子と組み合わせて使用します。

```
WITH dataset AS (
  SELECT
    'engineering' as department,
    ARRAY['Sharon', 'John', 'Bob', 'Sally'] as users
)
SELECT department, names FROM dataset
CROSS JOIN UNNEST(users) as t(names)
```

このクエリは以下を返します。

```
+----------------------+
| department  | names  |
+----------------------+
| engineering | Sharon |
+----------------------|
| engineering | John   |
+----------------------|
| engineering | Bob    |
+----------------------|
| engineering | Sally  |
+----------------------+
```

キーと値のペアの配列をフラット化するには、次の例に示すように、選択したキーを列に入れ替えます。

```
WITH
dataset AS (
  SELECT
    'engineering' as department,
     ARRAY[
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Bob', 'Smith', '40']),
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Jane', 'Doe', '30']),
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Billy', 'Smith', '8'])
     ] AS people
  )
SELECT names['first'] AS
 first_name,
 names['last'] AS last_name,
 department FROM dataset
CROSS JOIN UNNEST(people) AS t(names)
```

このクエリは以下を返します。

```
+-------------------------------------+
| first_name | last_name | department  |
+-------------------------------------+
| Bob        | Smith     | engineering |
| Jane       | Doe       | engineering |
| Billy      | Smith     | engineering |
+-------------------------------------+
```

従業員の一覧から、総合スコアが最も高い従業員を選択します。UNNEST は、デフォルトの結合演算子で暗黙的に指定されるため、FROM 句において先行 CROSS JOIN なしで使用できます。

```
WITH
dataset AS (
  SELECT ARRAY[
    CAST(ROW('Sally', 'engineering', ARRAY[1,2,3,4]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER))),
    CAST(ROW('John', 'finance', ARRAY[7,8,9]) AS ROW(name VARCHAR, department VARCHAR,
 scores ARRAY(INTEGER))),
    CAST(ROW('Amy', 'devops', ARRAY[12,13,14,15]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER)))
  ] AS users
),
users AS (
 SELECT person, score
 FROM
   dataset,
   UNNEST(dataset.users) AS t(person),
   UNNEST(person.scores) AS t(score)
)
SELECT person.name, person.department, SUM(score) AS total_score FROM users
GROUP BY (person.name, person.department)
ORDER BY (total_score) DESC
LIMIT 1
```

このクエリは以下を返します。

```
+---------------------------------+
| name | department | total_score |
+---------------------------------+
| Amy  | devops     | 54          |
+---------------------------------+
```

従業員の一覧から、個別のスコアが最も高い従業員を選択します。

```
WITH
dataset AS (
 SELECT ARRAY[
   CAST(ROW('Sally', 'engineering', ARRAY[1,2,3,4]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER))),
   CAST(ROW('John', 'finance', ARRAY[7,8,9]) AS ROW(name VARCHAR, department VARCHAR,
 scores ARRAY(INTEGER))),
   CAST(ROW('Amy', 'devops', ARRAY[12,13,14,15]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER)))
 ] AS users
),
users AS (
 SELECT person, score
 FROM
   dataset,
   UNNEST(dataset.users) AS t(person),
   UNNEST(person.scores) AS t(score)
)
SELECT person.name, score FROM users
ORDER BY (score) DESC
LIMIT 1
```

このクエリは以下を返します。

```
+--------------+
| name | score |
+--------------+
| Amy  | 15    |
+--------------+
```

考慮事項と制約事項

UNNEST がクエリ内の 1 つ以上の配列で使用され、配列の 1 つが NULL である場合、クエリは行を返しません。空の文字列の配列に UNNEST を使用すると、空の文字列が返されます。

例えば、次のクエリでは、2 番目の配列が Null であるため、クエリは行を返しません。

```
SELECT
    col1,
    col2
```

```
FROM UNNEST (ARRAY ['apples','oranges','lemons']) AS t(col1)
CROSS JOIN UNNEST (ARRAY []) AS t(col2)
```

次の例では、2 番目の配列が空の文字列を含むように変更されています。クエリは各行について、col1 の値を返し、col2 の値では空の文字列を返します。1 番目の配列の値を返すには、2 番目の配列の空の文字列が必要です。

```
SELECT
    col1,
    col2
FROM UNNEST (ARRAY ['apples','oranges','lemons']) AS t(col1)
CROSS JOIN UNNEST (ARRAY ['']) AS t(col2)
```

サブクエリからの配列の作成

行のコレクションから配列を作成します。

```
WITH
dataset AS (
  SELECT ARRAY[1,2,3,4,5] AS items
)
SELECT array_agg(i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

このクエリは以下を返します。

```
+-----------------+
| array_items     |
+-----------------+
| [1, 2, 3, 4, 5] |
+-----------------+
```

行のセットから一意の値の配列を作成するには、distinct キーワードを使用します。

```
WITH
dataset AS (
  SELECT ARRAY [1,2,2,3,3,4,5] AS items
)
SELECT array_agg(distinct i) AS array_items
FROM dataset
```

```
CROSS JOIN UNNEST(items) AS t(i)
```

このクエリは次の結果を返します。順序は保証されないことに注意してください。

```
+-----------------+
| array_items     |
+-----------------+
| [1, 2, 3, 4, 5] |
+-----------------+
```

array_agg 関数の使用方法の詳細については「Trino ドキュメント」の「集計関数」を参照してください。

配列のフィルタ処理

フィルタ条件に一致する行のコレクションから配列を作成します。

```
WITH
dataset AS (
  SELECT ARRAY[1,2,3,4,5] AS items
)
SELECT array_agg(i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
WHERE i > 3
```

このクエリは以下を返します。

```
+-------------+
| array_items |
+-------------+
| [4, 5]      |
+-------------+
```

配列の各要素に特定の値 (2 など) が含まれているかどうかに基づいて配列をフィルタ処理します。次に例を示します。

```
WITH
dataset AS (
  SELECT ARRAY
  [
```

```
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
)
SELECT i AS array_items FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
WHERE contains(i, 2)
```

このクエリは以下を返します。

```
+--------------+
| array_items  |
+--------------+
| [1, 2, 3, 4] |
+--------------+
```

filter 関数

```
 filter(ARRAY [list_of_values], boolean_function)
```

ARRAY 式で filter 関数を使用して、*boolean_function* が true となっている *list_of_values* 内の項目のサブセットである新しい配列を作成できます。filter 関数は、*UNNEST* 関数を使用できない場合に役立ちます。

次の例では、配列 [1,0,5,-1] 内のゼロより大きい値をフィルタリングします。

```
SELECT filter(ARRAY [1,0,5,-1], x -> x>0)
```

結果

[1,5]

次の例では、配列 [-1, NULL, 10, NULL] 内の null 以外の値をフィルタリングします。

```
SELECT filter(ARRAY [-1, NULL, 10, NULL], q -> q IS NOT NULL)
```

結果

[-1,10]

配列の並び替え

行のセットから、一意の値の並び替えられた配列を作成するには、以下の例にあるように、array_sort 関数を使用することができます。

```
WITH
dataset AS (
  SELECT ARRAY[3,1,2,5,2,3,6,3,4,5] AS items
)
SELECT array_sort(array_agg(distinct i)) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

このクエリは以下を返します。

```
+--------------------+
| array_items        |
+--------------------+
| [1, 2, 3, 4, 5, 6] |
+--------------------+
```

配列を複数の行に拡張する方法については、「ネストされた配列のフラット化」(ネストされた配列を平坦化する) を参照してください。

配列での集計関数の使用

- 配列内の値を加算するには、次の例のように SUM を使用します。
- 配列内の複数の行を集計するには、array_agg を使用します。詳細については、サブクエリからの配列の作成 を参照してください。

> **Note**
>
> ORDER BY は、Athena エンジンバージョン 2 以降の集約関数でサポートされています。

```
WITH
dataset AS (
  SELECT ARRAY
```

```
  [
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
),
item AS (
  SELECT i AS array_items
  FROM dataset, UNNEST(items) AS t(i)
)
SELECT array_items, sum(val) AS total
FROM item, UNNEST(array_items) AS t(val)
GROUP BY array_items;
```

最後の SELECT ステートメントでは、sum( ) と UNNEST を使用する代わりに、reduce( ) を使用して処理時間とデータ転送を短縮できます。次に例を示します。

```
WITH
dataset AS (
  SELECT ARRAY
  [
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
),
item AS (
  SELECT i AS array_items
  FROM dataset, UNNEST(items) AS t(i)
)
SELECT array_items, reduce(array_items, 0 , (s, x) -> s + x, s -> s) AS total
FROM item;
```

いずれのクエリも次の結果を返します。返される結果の順序は保証されません。

```
+----------------------+
| array_items  | total |
+----------------------+
| [1, 2, 3, 4] | 10    |
| [5, 6, 7, 8] | 26    |
| [9, 0]       | 9     |
+----------------------+
```

配列から文字列への変換

配列を 1 つの文字列に変換するには、array_join 関数を使用します。以下のスタンドアロン例は、words という名前のエイリアス化された配列を含む、dataset という名前のテーブルを作成します。クエリは、words の配列要素を結合するために array_join を使用し、それらをスペースで区切って、結果として得られた文字列を welcome_msg という名前のエイリアス化された列内に返します。

```
WITH
dataset AS (
  SELECT ARRAY ['hello', 'amazon', 'athena'] AS words
)
SELECT array_join(words, ' ') AS welcome_msg
FROM dataset
```

このクエリは以下を返します。

```
+---------------------+
| welcome_msg         |
+---------------------+
| hello amazon athena |
+---------------------+
```

配列を使用したマップの作成

マップは、Athena で利用できるデータ型で構成されるキー/値ペアです。マップを作成するには、MAP 演算子を使用して 2 つの配列を渡します。最初の配列は列 (キー) 名、2 つ目の配列は値です。配列のすべての値は、同じ型にする必要があります。マップの値配列のいずれかの要素を別の型にする必要がある場合は、後で変換できます。

例

次の例では、データセットからユーザーを選択します。MAP 演算子を使用して 2 つの配列を渡します。最初の配列には、値として「first」、「last」、「age」などの列名が含まれています。2 番目の配列は、これらの各列の値として「Bob」、「Smith」、「35」などで構成されます。

```
WITH dataset AS (
  SELECT MAP(
    ARRAY['first', 'last', 'age'],
    ARRAY['Bob', 'Smith', '35']
  ) AS user
```

```
)
SELECT user FROM dataset
```

このクエリは以下を返します。

```
+---------------------------------+
| user                            |
+---------------------------------+
| {last=Smith, first=Bob, age=35} |
+---------------------------------+
```

Map 値を取得するには、次の例に示すように、フィールド名に続けて [key_name] を選択します。

```
WITH dataset AS (
 SELECT MAP(
   ARRAY['first', 'last', 'age'],
   ARRAY['Bob', 'Smith', '35']
 ) AS user
)
SELECT user['first'] AS first_name FROM dataset
```

このクエリは以下を返します。

```
+------------+
| first_name |
+------------+
| Bob        |
+------------+
```

複合型の配列とネストされた構造のクエリ

ソースデータには、複雑なデータ型とネスト構造を持つ配列が含まれている場合があります。このセクションの例は、Athena クエリを使用して要素のデータ型を変更し、配列内の要素を見つけて、キーワードを検索する方法を示しています。

- ROW の作成
- CAST を使用した配列内のフィールド名の変更
- . 表記を使用した配列のフィルタ処理
- ネスト値を含む配列のフィルタ処理
- UNNEST を使用した配列のフィルタ処理

- regexp_like を使用した配列内でのキーワードの検索

ROW の作成

> **Note**
>
> このセクションの例では、使用するサンプルデータの作成手段として ROW を使用します。Athena 内のテーブルをクエリするときは、ROW データ型がデータソースから既に作成されているため、改めて作成する必要はありません。CREATE_TABLE を使用すると、Athena によって STRUCT が定義され、データが入力されて、データセット内の行ごとに ROW データ型が作成されます。基となる ROW データ型は、サポートされている SQL データ型の名前付きフィールドで構成されます。

```
WITH dataset AS (
 SELECT
   ROW('Bob', 38) AS users
 )
SELECT * FROM dataset
```

このクエリは以下を返します。

```
+-------------------------+
| users                   |
+-------------------------+
| {field0=Bob, field1=38} |
+-------------------------+
```

CAST を使用した配列内のフィールド名の変更

ROW 値が含まれている配列のフィールド名を変更するには、CAST 宣言を ROW できます。

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('Bob', 38) AS ROW(name VARCHAR, age INTEGER)
    ) AS users
)
SELECT * FROM dataset
```

このクエリは以下を返します。

```
+--------------------+
| users              |
+--------------------+
| {NAME=Bob, AGE=38} |
+--------------------+
```

> **Note**
>
> 上の例では、name を VARCHAR として宣言しています。Presto での型に合わせるためです。この STRUCT を CREATE TABLE ステートメント内で宣言する場合は、String 型を使用します。Hive では、このデータ型を String として定義するためです。

. 表記を使用した配列のフィルタ処理

次の例では、ドット accountId 表記を使用して、userIdentity ログの AWS CloudTrail 列から . フィールドを選択します。詳細については、「AWS CloudTrail ログのクエリ」を参照してください。

```
SELECT
  CAST(useridentity.accountid AS bigint) as newid
FROM cloudtrail_logs
LIMIT 2;
```

このクエリは以下を返します。

```
+--------------+
| newid        |
+--------------+
| 112233445566 |
+--------------+
| 998877665544 |
+--------------+
```

値の配列にクエリを実行するには、次のクエリを発行します。

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(ROW('Bob', 38) AS ROW(name VARCHAR, age INTEGER)),
```

```
    CAST(ROW('Alice', 35) AS ROW(name VARCHAR, age INTEGER)),
    CAST(ROW('Jane', 27) AS ROW(name VARCHAR, age INTEGER))
  ] AS users
)
SELECT * FROM dataset
```

次の結果が返されます。

```
+-----------------------------------------------------------------------+
| users                                                                 |
+-----------------------------------------------------------------------+
| [{NAME=Bob, AGE=38}, {NAME=Alice, AGE=35}, {NAME=Jane, AGE=27}] |
+-----------------------------------------------------------------------+
```

ネスト値を含む配列のフィルタ処理

大きな配列にはネスト構造が含まれている場合があり、ネスト構造内の値をフィルタ処理または検索する必要があります。

ネストされた BOOLEAN 値を含む値配列のデータセットを定義するには、次のクエリを発行します。

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ) AS sites
)
SELECT * FROM dataset
```

次の結果が返されます。

```
+-------------------------------------------------------------+
| sites                                                       |
+-------------------------------------------------------------+
| {HOSTNAME=aws.amazon.com, FLAGGEDACTIVITY={ISNEW=true}}  |
+-------------------------------------------------------------+
```

次に、この要素の BOOLEAN 値のフィルタ処理およびアクセスには、引き続きドット . 表記を使用します。

```
WITH dataset AS (
```

```
  SELECT
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ) AS sites
)
SELECT sites.hostname, sites.flaggedactivity.isnew
FROM dataset
```

このクエリは、ネストされたフィールドを選択し、次の結果を返します。

```
+------------------------+
| hostname       | isnew |
+------------------------+
| aws.amazon.com | true  |
+------------------------+
```

UNNEST を使用した配列のフィルタ処理

ネスト構造を含む配列を、そのいずれかの子要素でフィルタ処理するには、UNNEST 演算子を使用してクエリを発行します。UNNEST の詳細については、「ネスト配列のフラット化」を参照してください。

例えば、このクエリは、データセット内のサイトのホスト名を見つけます。

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ),
    CAST(
      ROW('news.cnn.com', ROW(false)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ),
    CAST(
      ROW('netflix.com', ROW(false)) AS ROW(hostname VARCHAR, flaggedActivity ROW(isNew
 BOOLEAN))
    )
  ] as items
)
SELECT sites.hostname, sites.flaggedActivity.isNew
FROM dataset, UNNEST(items) t(sites)
```

```
WHERE sites.flaggedActivity.isNew = true
```

返される結果は次のとおりです。

```
+------------------------+
| hostname       | isnew |
+------------------------+
| aws.amazon.com | true  |
+------------------------+
```

regexp_like を使用した配列内でのキーワードの検索

次の例では、regexp_like 関数を使用して、配列内の要素内でキーワードのデータセットを検索する方法を示します。入力として評価対象の正規表現パターン、またはパイプ (|) で区切られた要素のリストを使用し、パターンを評価して、指定された文字列にそれが含まれるかどうかを判別します。

正規表現パターンは文字列内に含まれている必要がありますが、文字列と一致している必要はありません。文字列全体と一致させるには、冒頭に ^、末尾に $ を付けてパターンを囲みます (例: `'^pattern$'`)。

ホスト名と `flaggedActivity` 要素が含まれているサイトの配列について考えてみましょう。この要素内の ARRAY には、複数の MAP 要素が含まれており、要素ごとに異なる一般的なキーワードと人気度カウントを示しています。この配列の MAP 内で特定のキーワードを検索するとします。

このデータセットを検索して特定のキーワードを持つサイトを見つけるには、類似した SQL `regexp_like` 演算子の代わりに `LIKE` を使用します。多数のキーワードの検索には `regexp_like` の方が効率的です。

Example 例 1: **regexp_like** の使用

この例のクエリでは、`regexp_like` 関数を使用して用語 `'politics|bigdata'` を検索します。これは配列内の値で見つかります。

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(ARRAY[
          MAP(ARRAY['term', 'count'], ARRAY['bigdata', '10']),
          MAP(ARRAY['term', 'count'], ARRAY['serverless', '50']),
          MAP(ARRAY['term', 'count'], ARRAY['analytics', '82']),
          MAP(ARRAY['term', 'count'], ARRAY['iot', '74'])
      ])
```

```
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
   ),
   CAST(
     ROW('news.cnn.com', ROW(ARRAY[
       MAP(ARRAY['term', 'count'], ARRAY['politics', '241']),
       MAP(ARRAY['term', 'count'], ARRAY['technology', '211']),
       MAP(ARRAY['term', 'count'], ARRAY['serverless', '25']),
       MAP(ARRAY['term', 'count'], ARRAY['iot', '170'])
     ])
     ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
   ),
   CAST(
     ROW('netflix.com', ROW(ARRAY[
       MAP(ARRAY['term', 'count'], ARRAY['cartoons', '1020']),
       MAP(ARRAY['term', 'count'], ARRAY['house of cards', '112042']),
       MAP(ARRAY['term', 'count'], ARRAY['orange is the new black', '342']),
       MAP(ARRAY['term', 'count'], ARRAY['iot', '4'])
     ])
     ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
   )
 ] AS items
),
sites AS (
  SELECT sites.hostname, sites.flaggedactivity
  FROM dataset, UNNEST(items) t(sites)
)
SELECT hostname
FROM sites, UNNEST(sites.flaggedActivity.flags) t(flags)
WHERE regexp_like(flags['term'], 'politics|bigdata')
GROUP BY (hostname)
```

このクエリは 2 つのサイトを返します。

```
+----------------+
| hostname       |
+----------------+
| aws.amazon.com |
+----------------+
| news.cnn.com   |
+----------------+
```

Example 例 2: **regexp_like** の使用

次の例のクエリでは、regexp_like 関数を使用して検索語句に一致するサイトの人気度スコアを合計し、最高から最低の順に並べます。

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(ARRAY[
          MAP(ARRAY['term', 'count'], ARRAY['bigdata', '10']),
          MAP(ARRAY['term', 'count'], ARRAY['serverless', '50']),
          MAP(ARRAY['term', 'count'], ARRAY['analytics', '82']),
          MAP(ARRAY['term', 'count'], ARRAY['iot', '74'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    ),
    CAST(
      ROW('news.cnn.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['politics', '241']),
        MAP(ARRAY['term', 'count'], ARRAY['technology', '211']),
        MAP(ARRAY['term', 'count'], ARRAY['serverless', '25']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '170'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    ),
    CAST(
      ROW('netflix.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['cartoons', '1020']),
        MAP(ARRAY['term', 'count'], ARRAY['house of cards', '112042']),
        MAP(ARRAY['term', 'count'], ARRAY['orange is the new black', '342']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '4'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    )
  ] AS items
),
sites AS (
  SELECT sites.hostname, sites.flaggedactivity
  FROM dataset, UNNEST(items) t(sites)
)
```

```
SELECT hostname, array_agg(flags['term']) AS terms, SUM(CAST(flags['count'] AS
 INTEGER)) AS total
FROM sites, UNNEST(sites.flaggedActivity.flags) t(flags)
WHERE regexp_like(flags['term'], 'politics|bigdata')
GROUP BY (hostname)
ORDER BY total DESC
```

このクエリは 2 つのサイトを返します。

```
+-------------------------------------+
| hostname       | terms    | total  |
+----------------+------------------+
| news.cnn.com   | politics |  241   |
+----------------+------------------+
| aws.amazon.com | bigdata |  10    |
+----------------+------------------+
```

地理空間データのクエリ

地理空間データには、オブジェクトの地理的位置を指定する識別子が含まれています。このタイプのデータの例としては、天気予報、地図の案内、位置情報を含むツイート、店舗の場所、航空路線などがあります。地理空間データは、ビジネス分析、レポート、および予測で重要な役割を果たします。

緯度や経度などの地理空間識別子を使用すると、郵送先住所を地理座標系に変換できます。

トピック

- 地理空間クエリとは
- 入力データ形式とジオメトリデータ型
- サポートされる地理空間関数
- 例: 地理空間クエリ

地理空間クエリとは

地理空間クエリとは、Athena でサポートされている特殊な SQL クエリタイプです。非空間 SQL クエリとは以下の点が異なります。

- 特殊なジオメトリデータ型として `point`、`line`、`multiline`、`polygon`、および `multipolygon` を使用する。

- ジオメトリデータ型間の関係として `distance`、`equals`、`crosses`、`touches`、`overlaps`、`disjoint` などを表現する。

Athena で地理空間クエリを使用すると、以下のオペレーションと、その他の類似したオペレーションを実行できます。

- 2 点間の距離を確認する。
- あるエリア (ポリゴン) 内に別のエリアが含まれているかどうかを確認する。
- 1 つの行が別の行や多角形と交差または接触しているかどうかを確認する。

例えば、Athena でレーニア山の地理座標について `double` 型の値から `point` ジオメトリデータ型を得るには、以下の例にあるように `ST_Point (longitude, latitude)` 地理空間関数を使用します。

```
ST_Point(-121.7602, 46.8527)
```

入力データ形式とジオメトリデータ型

Athena で地理空間関数を使用するには、データを WKT 形式で入力するか、Hive JSON SerDe を使用します。Athena でサポートされているジオメトリデータ型を使用することもできます。

入力データ形式

地理空間クエリを処理するため、Athena では以下のデータ形式での入力データがサポートされています。

- WKT (Well-known Text)。Athena では、WKT が `varchar(x)` または `string` のデータ型として表されます。
- JSON エンコード形式の地理空間データ。地理空間データが含まれている JSON ファイルを解析してテーブルを作成するために、Athena では Hive JSON SerDe が使用されます。Athena でこの SerDe を使用する方法の詳細については、「JSON SerDe ライブラリ」を参照してください。

ジオメトリデータ型

地理空間クエリを処理するため、Athena では以下の特化されたジオメトリデータ型がサポートされています。

- `point`

- `line`
- `polygon`
- `multiline`
- `multipolygon`

サポートされる地理空間関数

Athena で使用できる地理空間関数は、使用するエンジンのバージョンに応じて異なります。

- Athena エンジンバージョン 3 の地理空間関数については、「Trino documentation」の「Geospatial functions」を参照してください。
- Athena エンジンバージョン 2 での関数名の変更と新しい関数のリストについては、「Athena エンジンバージョン 2 における地理空間関数名の変更と新しい関数」を参照してください。

Athena エンジンのバージョニングについては、「Athena エンジンのバージョニング」を参照してください。

トピック

- Athena エンジンバージョン 3 の地理空間関数
- Athena エンジンバージョン 2 の地理空間関数

Athena エンジンバージョン 3 の地理空間関数

Athena エンジンバージョン 3 の地理空間関数については、「Trino documentation」の「Geospatial functions」を参照してください。

Athena エンジンバージョン 2 の地理空間関数

このトピックでは、Athena エンジンバージョン 2 以降でサポートされる ESRI 地理空間関数のみを示しています。Athena のエンジンバージョンの詳細については、「Athena エンジンのバージョニング」を参照してください。

Athena エンジンバージョン 2 での変更

- 一部の関数の入力の型と出力の型が変更されました。最も注目すべき変更は、`VARBINARY` 型が入力に直接サポートされなくなったことです。詳細については、「地理空間関数への変更」(地理空間関数への変更) を参照してください。

- 一部の地理空間関数の名前が変更されています。詳細については、「Athena エンジンバージョン 2 における地理空間関数名の変更」(Athena エンジンバージョン 2 における地理空間関数の名前の変更) を参照してください。
- 新しい関数が追加されました。詳細については、「Athena エンジンバージョン 2 の新しい地理空間関数」を参照してください。

Athena は以下の種類の地理空間関数をサポートしています。

- コンストラクター関数
- 地理空間関係関数
- オペレーション関数
- アクセサ関数
- 集計関数
- Bing tile 関数

コンストラクター関数

コンストラクター関数では、point、line、または polygon ジオメトリデータ型のバイナリ表現を取得します。これらの関数を使用して、バイナリデータをテキストに変換し、Well-Known Text (WKT) として表現されるジオメトリデータのバイナリ値を取得することもできます。

ST_AsBinary(geometry)

指定されたジオメトリの WKB 表現を含む varbinary データ型を返します。例:

```
SELECT ST_AsBinary(ST_Point(-158.54, 61.56))
```

ST_AsText(geometry)

指定した各ジオメトリデータ型をテキストに変換します。値を varchar データ型で返します。これは、ジオメトリデータ型の WKT 表現です。例 :

```
SELECT ST_AsText(ST_Point(-158.54, 61.56))
```

ST_GeomAsLegacyBinary(geometry)

指定されたジオメトリから、従来の varbinary を返します。例 :

```
SELECT ST_GeomAsLegacyBinary(ST_Point(-158.54, 61.56)
```

ST_GeometryFromText(varchar)

WKT 形式のテキストをジオメトリデータ型に変換します。ジオメトリデータ型の値を返します。例:

```
SELECT ST_GeometryFromText(ST_AsText(ST_Point(1, 2)))
```

ST_GeomFromBinary(varbinary)

WKB 表現からジオメトリ型オブジェクトを返します。例 :

```
SELECT ST_GeomFromBinary(ST_AsBinary(ST_Point(-158.54, 61.56)))
```

ST_GeomFromLegacyBinary(varbinary)

従来の varbinary 型から、ジオメトリ型オブジェクトを返します。例 :

```
SELECT ST_GeomFromLegacyBinary(ST_GeomAsLegacyBinary(ST_Point(-158.54, 61.56)))
```

ST_LineFromText(varchar)

ジオメトリデータ型 `line` で値を返します。例:

```
SELECT ST_Line('linestring(1 1, 2 2, 3 3)')
```

ST_LineString(array(point))

Point ジオメトリ型の配列から形成される `LineString` のジオメトリ型を返します。指定された配列にある空ではないポイントが 2 個未満の場合、空の `LineString` が返されます。配列内の要素のいずれかが null、空、または以前のものと同じである場合は、例外がスローされます。返されるジオメトリは単純ではない場合があります。指定された入力に応じて、返されるジオメトリが自己交差している、または重複する頂点を含んでいる場合があります。例:

```
SELECT ST_LineString(ARRAY[ST_Point(-158.54, 61.56), ST_Point(-158.55, 61.56)])
```

ST_MultiPoint(array(point))

指定されたポイントから形成される `MultiPoint` のジオメトリオブジェクトを返します。指定された配列が空の場合は null を返します。配列内の要素のいずれかが null または空の場合は、例外がス

ローされます。返されるジオメトリは単純ではない場合があり、指定された配列に重複がある場合は、重複ポイントが含まれる可能性があります。例:

```
SELECT ST_MultiPoint(ARRAY[ST_Point(-158.54, 61.56), ST_Point(-158.55, 61.56)])
```

ST_Point(double, double)

ジオメトリ型が point のオブジェクトを返します。この関数への入力データ値には、ユニバーサル横メルカトル (UTM) 直交座標系の値などのジオメトリ値、または 10 進角で表した地図単位 (経度と緯度) を使用します。経度と緯度の値には、WGS 1984 または EPSG:4326 とも呼ばれる世界測地系が使用されます。WGS 1984 は、全地球測位システム (GPS) で使用されている座標系です。

たとえば、次の表記では、地図座標は経度と緯度で指定され、値 .072284 (バッファ距離) は 10 進角の角度単位で指定されます。

```
SELECT ST_Buffer(ST_Point(-74.006801, 40.705220), .072284)
```

構文:

```
SELECT ST_Point(longitude, latitude) FROM earthquakes LIMIT 1
```

以下の例では、特定の経度と緯度の座標を使用しています。

```
SELECT ST_Point(-158.54, 61.56)
FROM earthquakes
LIMIT 1
```

次の例では、特定の経度と緯度の座標を使用しています。

```
SELECT ST_Point(-74.006801, 40.705220)
```

以下の例では、WKT からジオメトリを取得するために ST_AsText 関数を使用しています。

```
SELECT ST_AsText(ST_Point(-74.006801, 40.705220)) AS WKT
```

ST_Polygon(varchar)

時計回り (左から右) に提供される縦座標のシーケンスを使用することで、ジオメトリデータ型 polygon が返されます。Athena エンジンバージョン 2 以降では、ポリゴンのみが入力として受け入れられます。例 :

```
SELECT ST_Polygon('polygon ((1 1, 1 4, 4 4, 4 1))')
```

to_geometry(sphericalGeography)

指定された球面ジオグラフィオブジェクトからジオメトリオブジェクトを返します。例:

```
SELECT to_geometry(to_spherical_geography(ST_Point(-158.54, 61.56)))
```

to_spherical_geography(geometry)

指定されたジオメトリから球面ジオグラフィオブジェクトを返します。この関数は、ジオメトリオブジェクトを地球半径の球面での球面ジオグラフィオブジェクトに変換するために使用します。この関数は、2D 空間で定義された POINT、MULTIPOINT、LINESTRING、MULTILINESTRING、POLYGON、および MULTIPOLYGON、またはこのようなジオメトリの GEOMETRYCOLLECTION のみで使用できます。この関数は、指定されたジオメトリの各ポイントについて、point.x が [-180.0, 180.0] 内であり、point.y が [-90.0, 90.0] 内であることを検証します。この関数は、これらのポイントを経度および緯度として使用して、sphericalGeography 結果の形状を構築します。

例:

```
SELECT to_spherical_geography(ST_Point(-158.54, 61.56))
```

地理空間関係関数

以下の関数は、入力として指定された 2 つの異なるジオメトリ間の関係を表現し、boolean 型の値を返します。ジオメトリのペアを指定する順序が重要になります。最初のジオメトリ値は左ジオメトリ、2 番目のジオメトリ値は右ジオメトリと呼ばれます。

これら関数は以下を返します。

- TRUE。関数が記述する関係が満たされた場合に限ります。
- FALSE。関数が記述する関係が満たされない場合に限ります。

ST_Contains(geometry, geometry)

左ジオメトリに右ジオメトリが含まれている場合に限り、TRUE を返します。例:

```
SELECT ST_Contains('POLYGON((0 2,1 1,0 -1,0 2))', 'POLYGON((-1 3,2 1,0 -3,-1 3))')
```

```
SELECT ST_Contains('POLYGON((0 2,1 1,0 -1,0 2))', ST_Point(0, 0))
```

```
SELECT ST_Contains(ST_GeometryFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeometryFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'))
```

ST_Crosses(geometry, geometry)

左ジオメトリが右ジオメトリとクロスする場合に限り、TRUE を返します。例:

```
SELECT ST_Crosses(ST_Line('linestring(1 1, 2 2 )'), ST_Line('linestring(0 1, 2 2)'))
```

ST_Disjoint(geometry, geometry)

左ジオメトリと右ジオメトリの共通部分が空の場合に限り、TRUE を返します。例:

```
SELECT ST_Disjoint(ST_Line('linestring(0 0, 0 1)'), ST_Line('linestring(1 1, 1 0)'))
```

ST_Equals(geometry, geometry)

左ジオメトリに右ジオメトリと等しい場合に限り、TRUE を返します。例:

```
SELECT ST_Equals(ST_Line('linestring( 0 0, 1 1)'), ST_Line('linestring(1 3, 2 2)'))
```

ST_Intersects(geometry, geometry)

左ジオメトリが右ジオメトリとインターセクトする場合に限り、TRUE を返します。例:

```
SELECT ST_Intersects(ST_Line('linestring(8 7, 7 8)'), ST_Polygon('polygon((1 1, 4 1, 4
 4, 1 4))'))
```

ST_Overlaps(geometry, geometry)

左ジオメトリが右ジオメトリにオーバーラップする場合に限り、TRUE を返します。例:

```
SELECT ST_Overlaps(ST_Polygon('polygon((2 0, 2 1, 3 1))'), ST_Polygon('polygon((1 1, 1
 4, 4 4, 4 1))'))
```

ST_Relate(geometry, geometry, varchar)

左ジオメトリに、右ジオメトリとの DE-9IM (Dimensionally Extended nine-Intersection Model) 関係が指定されている場合に限り、TRUE を返します。3 番目の (varchar) 入力は、この関係を使用します。例:

```
SELECT ST_Relate(ST_Line('linestring(0 0, 3 3)'), ST_Line('linestring(1 1, 4 4)'),
 'T********')
```

ST_Touches(geometry, geometry)

左ジオメトリが右ジオメトリに接する場合に限り、TRUE を返します。

例:

```
SELECT ST_Touches(ST_Point(8, 8), ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

ST_Within(geometry, geometry)

左ジオメトリが右ジオメトリ内にある場合に限り、TRUE を返します。

例:

```
SELECT ST_Within(ST_Point(8, 8), ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

オペレーション関数

オペレーション関数では、ジオメトリデータ型の値に対してオペレーションを実行します。たとえば、単一のジオメトリデータ型の境界、2 つのジオメトリデータ型の共通部分、左ジオメトリと右ジオメトリ間の差異 (両方が同じデータ型の場合)、および特定のジオメトリデータ型の外部バッファや外部リングを取得できます。

geometry_union(array(geometry))

指定されたジオメトリのポイントセットの和集合を表すジオメトリを返します。例:

```
SELECT geometry_union(ARRAY[ST_Point(-158.54, 61.56), ST_Point(-158.55, 61.56)])
```

ST_Boundary(geometry)

ジオメトリデータ型の 1 つを入力として使用し、boundary ジオメトリデータ型を返します。

例:

```
SELECT ST_Boundary(ST_Line('linestring(0 1, 1 0)')))
```

```
SELECT ST_Boundary(ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

ST_Buffer(geometry, double)

いずれかのジオメトリデータ型 (point、line、polygon、multiline、multipolygon、および double 型として distance) を入力として受け取ります。指定した距離 (または半径) でバッファリングされたジオメトリデータ型を返します。例:

```
SELECT ST_Buffer(ST_Point(1, 2), 2.0)
```

次の例では、地図座標は経度と緯度で指定され、値 .072284 (バッファ距離) は 10 進角の角度単位で指定されます。

```
SELECT ST_Buffer(ST_Point(-74.006801, 40.705220), .072284)
```

ST_Difference(geometry, geometry)

左ジオメトリと右ジオメトリ間の差のジオメトリを返します。例:

```
SELECT ST_AsText(ST_Difference(ST_Polygon('polygon((0 0, 0 10, 10 10, 10 0))'),
 ST_Polygon('polygon((0 0, 0 5, 5 5, 5 0))')))
```

ST_Envelope(geometry)

line、polygon、multiline および multipolygon ジオメトリデータ型を入力として受け取ります。point ジオメトリデータ型はサポートされていません。エンベロープをジオメトリとして返します。エンベロープは指定されたジオメトリデータ型を囲む矩形です。例:

```
SELECT ST_Envelope(ST_Line('linestring(0 1, 1 0)'))
```

```
SELECT ST_Envelope(ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

ST_EnvelopeAsPts(geometry)

ジオメトリの境界矩形ポリゴンの左下隅と右上隅を表す 2 つのポイントの配列を返します。指定したジオメトリが空の場合は、null を返します。例:

```
SELECT ST_EnvelopeAsPts(ST_Point(-158.54, 61.56))
```

ST_ExteriorRing(geometry)

入力タイプ polygon の外部リングのジオメトリを返します。Athena エンジンバージョン 2 以降では、ポリゴン以外のジオメトリは入力として受け入れられません。例:

```
SELECT ST_ExteriorRing(ST_Polygon(1,1, 1,4, 4,1))
```

```
SELECT ST_ExteriorRing(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1
 1))'))
```

ST_Intersection(geometry, geometry)

左ジオメトリと右ジオメトリの共通部分のジオメトリを返します。例:

```
SELECT ST_Intersection(ST_Point(1,1), ST_Point(1,1))
```

```
SELECT ST_Intersection(ST_Line('linestring(0 1, 1 0)'), ST_Polygon('polygon((1  1, 1
 4, 4  4, 4 1))'))
```

```
SELECT ST_AsText(ST_Intersection(ST_Polygon('polygon((2 0, 2 3, 3 0))'),
 ST_Polygon('polygon((1 1, 4 1, 4 4, 1 4))')))
```

ST_SymDifference(geometry, geometry)

左ジオメトリと右ジオメトリの間における幾何学的な対称差のジオメトリを返します。例:

```
SELECT ST_AsText(ST_SymDifference(ST_Line('linestring(0 2, 2 2)'),
 ST_Line('linestring(1 2, 3 2)')))
```

ST_Union(geometry, geometry)

指定されたジオメトリのポイントセットの和集合を表すジオメトリデータ型を返します。例:

```
SELECT ST_Union(ST_Point(-158.54, 61.56),ST_LineString(array[ST_Point(1,2),
 ST_Point(3,4)]))
```

アクセサ関数

アクセサ関数は、さまざまな geometry データ型から varchar 型、bigint 型、または double 型の値を取得するために役立ちます。geometry は、Athena でサポートされているジオメトリデータ型 (point、line、polygon、multiline、および multipolygon) のいずれかです。たとえば、polygon ジオメトリデータ型の面積、指定したジオメトリデータ型の最大と最小の X 値と Y 値、line の長さ、または指定したジオメトリデータ型のポイント数を取得できます。

geometry_invalid_reason(geometry)

指定されたジオメトリが有効ではない、または単純ではない理由を varchar データ型で返します。指定されたジオメトリが有効でも単純でもない場合は、有効ではない理由を返します。指定されたジオメトリが有効で単純な場合は、null が返されます。例 :

```
SELECT geometry_invalid_reason(ST_Point(-158.54, 61.56))
```

great_circle_distance(latitude1, longitude1, latitude2, longitude2)

地球表面上の 2 つのポイント間のキロ単位での大円距離を double として返します。例 :

```
SELECT great_circle_distance(36.12, -86.67, 33.94, -118.40)
```

line_locate_point(lineString, point)

指定されたポイントに対する指定されたラインストリング上の最も近いポイントの位置を 2D ラインの合計長の割合として表す 0 から 1 の double を返します。

指定されたラインストリングまたはポイントが空または null の場合は、null が返されます。例:

```
SELECT line_locate_point(ST_GeometryFromText('LINESTRING (0 0, 0 1)'), ST_Point(0,
 0.2))
```

simplify_geometry(geometry, double)

Ramer-Douglas-Peucker アルゴリズムを使用して、指定されたジオメトリの簡略化バージョンであるジオメトリデータ型を返します。無効な派生ジオメトリ (特にポリゴン) の作成を回避します。例:

```
SELECT simplify_geometry(ST_GeometryFromText('POLYGON ((1 0, 2 1, 3 1, 3 1, 4 1, 1
 0))'), 1.5)
```

ST_Area(geometry)

ジオメトリデータ型を入力として受け取り、面積を double 型で返します。例:

```
SELECT ST_Area(ST_Polygon('polygon((1 1, 4 1, 4 4, 1 4))'))
```

ST_Centroid(geometry)

ジオメトリデータ型 polygon を入力として使用し、ポリゴンのエンベロープの中心点である point ジオメトリデータ型を返します。例:

```
SELECT ST_Centroid(ST_GeometryFromText('polygon ((0 0, 3 6, 6 0, 0 0))'))
```

```
SELECT ST_AsText(ST_Centroid(ST_Envelope(ST_GeometryFromText('POINT (53 27)'))))
```

ST_ConvexHull(geometry)

指定された入力内のすべてのジオメトリを囲む最小の凸型ジオメトリであるジオメトリデータ型を返します。例:

```
SELECT ST_ConvexHull(ST_Point(-158.54, 61.56))
```

ST_CoordDim(geometry)

サポートされているジオメトリデータ型のいずれかを入力として使用し、座標コンポーネントの数を tinyint 型で返します。例:

```
SELECT ST_CoordDim(ST_Point(1.5,2.5))
```

ST_Dimension(geometry)

サポートされているいずれかのジオメトリデータ型を入力として受け取り、ジオメトリの空間ディメンションを tinyint 型で返します。例:

```
SELECT ST_Dimension(ST_Polygon('polygon((1 1, 4 1, 4 4, 1 4))'))
```

ST_Distance(geometry, geometry)

空間参照に基づいて、2 つのジオメトリ間の投影単位での最短 2 次元デカルト距離が含まれる double を返します。Athena エンジンバージョン 2 以降では、入力の 1 つが空のジオメトリである場合、null を返します。例 :

```
SELECT ST_Distance(ST_Point(0.0,0.0), ST_Point(3.0,4.0))
```

ST_Distance(sphericalGeography, sphericalGeography)

2 つの球面ジオグラフィポイント間のメートル単位での大円距離を double として返します。例:

```
SELECT ST_Distance(to_spherical_geography(ST_Point(61.56,
 -86.67)),to_spherical_geography(ST_Point(61.56, -86.68)))
```

ST_EndPoint(geometry)

line ジオメトリデータ型の最後のポイントを point ジオメトリデータ型で返します。例:

```
SELECT ST_EndPoint(ST_Line('linestring(0 2, 2 2)'))
```

ST_Geometries(geometry)

指定されたコレクション内のジオメトリの配列を返します。指定されたジオメトリがマルチジオメトリではない場合は、1 要素の配列を返します。指定されたジオメトリが空の場合は null を返します。

例えば、MultiLineString オブジェクトが与えられると、ST_Geometries は LineString オブジェクトの配列を作成します。GeometryCollection オブジェクトが与えられると、ST_Geometries は、その構成要素のフラット化されていない配列を返します。例:

```
SELECT ST_Geometries(GEOMETRYCOLLECTION(MULTIPOINT(0 0, 1 1),
 GEOMETRYCOLLECTION(MULTILINESTRING((2 2, 3 3)))))
```

結果:

```
array[MULTIPOINT(0 0, 1 1),GEOMETRYCOLLECTION(MULTILINESTRING((2 2, 3 3)))]
```

ST_GeometryN(geometry, index)

指定された整数インデックスにあるジオメトリ要素をジオメトリデータ型として返します。インデックスは 1 から始まります。指定されたジオメトリがジオメトリのコレクション (例えば

GEOMETRYCOLLECTION または MULTI* オブジェクト) である場合は、指定されたインデックスのジオメトリが返されます。指定されたインデックスが 1 より小さい、またはコレクション内の要素の合計数より大きい場合は、null が返されます。要素の合計数を確認するには、ST_NumGeometries を使用します。特異ジオメトリ (例えば POINT、LINESTRING、または POLYGON) は、1 つの要素のコレクションとして扱われます。空のジオメトリは空のコレクションとして扱われます。例:

```
SELECT ST_GeometryN(ST_Point(-158.54, 61.56),1)
```

ST_GeometryType(geometry)

ジオメトリの型を varchar として返します。例:

```
SELECT ST_GeometryType(ST_Point(-158.54, 61.56))
```

ST_InteriorRingN(geometry, index)

指定されたインデックスの内部リンク要素を返します (インデックスは 1 から始まります)。指定されたインデックスが 1 より小さい、または指定されたジオメトリ内の内部リングの合計数より大きい場合は、null が返されます。指定されたジオメトリがポリゴンではない場合は、エラーがスローされます。要素の合計数を確認するには、ST_NumInteriorRing を使用します。例:

```
SELECT ST_InteriorRingN(st_polygon('polygon ((0 0, 1 0, 1 1, 0 1, 0 0))'),1)
```

ST_InteriorRings(geometry)

指定されたジオメトリで見つかったすべての内部リングのジオメトリ配列を返します。または、ポリゴンに内部リングがない場合は、空の配列を返します。指定されたジオメトリが空の場合は null を返します。指定されたジオメトリがポリゴンではない場合は、エラーがスローされます。例:

```
SELECT ST_InteriorRings(st_polygon('polygon ((0 0, 1 0, 1 1, 0 1, 0 0))'))
```

ST_IsClosed(geometry)

line および multiline ジオメトリデータ型のみを入力として受け取ります。ラインが閉じている場合に限り、TRUE (boolean 型) を返します。例:

```
SELECT ST_IsClosed(ST_Line('linestring(0 2, 2 2)'))
```

ST_IsEmpty(geometry)

line および multiline ジオメトリデータ型のみを入力として受け取ります。指定されたジオメトリが空、つまり line の開始値と終了値が一致する場合に限り、TRUE (boolean 型) を返します。例:

```
SELECT ST_IsEmpty(ST_Point(1.5, 2.5))
```

ST_IsRing(geometry)

TRUE 型が閉じていてシンプルである場合に限り、boolean (line 型) を返します。例:

```
SELECT ST_IsRing(ST_Line('linestring(0 2, 2 2)'))
```

ST_IsSimple(geometry)

指定されたジオメトリに変則的なジオメトリポイント (自己交差や自己接触など) がない場合、true を返します。ジオメトリが単純ではない理由を判断するには、geometry_invalid_reason() を使用します。例:

```
SELECT ST_IsSimple(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]))
```

ST_IsValid(geometry)

指定したジオメトリが適切に形成されている場合に限り、true を返します。ジオメトリが適切に形成されていない理由を判断するには、geometry_invalid_reason() を使用します。例:

```
SELECT ST_IsValid(ST_Point(61.56, -86.68))
```

ST_Length(geometry)

line の長さを double 型で返します。例:

```
SELECT ST_Length(ST_Line('linestring(0 2, 2 2)'))
```

ST_NumGeometries(geometry)

コレクション内のジオメトリの数を整数で返します。ジオメトリがジオメトリのコレクション (例えば GEOMETRYCOLLECTION または MULTI* オブジェクト) である場合、ジオメトリの数を返しま

す。単一のジオメトリは 1 を返し、空のジオメトリは 0 を返します。GEOMETRYCOLLECTION オブジェクト内の空のジオメトリは、1 つのジオメトリとして計上されます。例えば、以下の式は 1 と評価されます。

```
ST_NumGeometries(ST_GeometryFromText('GEOMETRYCOLLECTION(MULTIPOINT EMPTY)'))
```

ST_NumInteriorRing(geometry)

polygon ジオメトリの内部リング数を bigint 型で返します。例:

```
SELECT ST_NumInteriorRing(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1
 1))'))
```

ST_NumPoints(geometry)

ジオメトリのポイント数を bigint 型で返します。例:

```
SELECT ST_NumPoints(ST_Point(1.5, 2.5))
```

ST_PointN(lineString, index)

指定された整数インデックスでの指定されたラインストリングの頂点を point ジオメトリデータ型として返します。インデックスは 1 から始まります。指定されたインデックスが 1 より小さい、またはコレクション内の要素の合計数より大きい場合は、null が返されます。要素の合計数を確認するには、ST_NumPoints を使用します。例:

```
SELECT ST_PointN(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]),1)
```

ST_Points(geometry)

指定されたラインストリングジオメトリオブジェクトからのポイントの配列を返します。例:

```
SELECT ST_Points(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]))
```

ST_StartPoint(geometry)

line ジオメトリデータ型の最初のポイントを point ジオメトリデータ型で返します。例:

```
SELECT ST_StartPoint(ST_Line('linestring(0 2, 2 2)'))
```

ST_X(point)

ポイントの X 座標を double 型で返します。例:

```
SELECT ST_X(ST_Point(1.5, 2.5))
```

ST_XMax(geometry)

ジオメトリの X 座標の最大値を double 型で返します。例:

```
SELECT ST_XMax(ST_Line('linestring(0 2, 2 2)'))
```

ST_XMin(geometry)

ジオメトリの X 座標の最小値を double 型で返します。例:

```
SELECT ST_XMin(ST_Line('linestring(0 2, 2 2)'))
```

ST_Y(point)

ポイントの Y 座標を double 型で返します。例:

```
SELECT ST_Y(ST_Point(1.5, 2.5))
```

ST_YMax(geometry)

ジオメトリの Y 座標の最大値を double 型で返します。例:

```
SELECT ST_YMax(ST_Line('linestring(0 2, 2 2)'))
```

ST_YMin(geometry)

ジオメトリの Y 座標の最小値を double 型で返します。例:

```
SELECT ST_YMin(ST_Line('linestring(0 2, 2 2)'))
```

集計関数

convex_hull_agg(geometry)

入力として渡されたすべてのジオメトリを囲む最小凸型ジオメトリを返します。

geometry_union_agg(geometry)

入力として渡されたすべてのジオメトリのポイントセットの和集合を表すジオメトリを返します。

Bing tile 関数

以下の関数は、ジオメトリと Microsoft Bing Maps Tile System のタイル間の変換を実行します。

bing_tile(x, y, zoom_level)

整数座標 x および y と、指定されたズームレベルから Bing tile オブジェクトを返します。ズームレベルは、1 から 23 の整数である必要があります。例:

```
SELECT bing_tile(10, 20, 12)
```

bing_tile(quadKey)

quadkey から Bing tile オブジェクトを返します。例:

```
SELECT bing_tile(bing_tile_quadkey(bing_tile(10, 20, 12)))
```

bing_tile_at(latitude, longitude, zoom_level)

指定された緯度、経度、およびズームレベルの Bing tile オブジェクトを返します。緯度は、-85.05112878 から 85.05112878 までのものにする必要があります。経度は -180 から 180 までのものにする必要があります。`latitude` および `longitude` の値は、整数の `double` および `zoom_level` にする必要があります。例:

```
SELECT bing_tile_at(37.431944, -122.166111, 12)
```

bing_tiles_around(latitude, longitude, zoom_level)

指定されたズームレベルで、指定された緯度と経度のポイントを取り囲む Bing tile の配列を返します。例:

```
SELECT bing_tiles_around(47.265511, -122.465691, 14)
```

bing_tiles_around(latitude, longitude, zoom_level, radius_in_km)

指定されたズームレベルでの Bing tile の配列を返します。この配列には、指定された緯度と経度を囲む、キロメートル単位で指定された半径の円をカバーする Bing tile の最小セットが含まれます。`latitude`、`longitude`、および `radius_in_km` の値は `double` で、ズームレベルは `integer` です。例:

```
SELECT bing_tiles_around(37.8475, 112.596667, 10, .5)
```

bing_tile_coordinates(tile)

指定された Bing tile の x および y 座標を返します。例:

```
SELECT bing_tile_coordinates(bing_tile_at(37.431944, -122.166111, 12))
```

bing_tile_polygon(tile)

指定された Bing tile のポリゴン表現を返します。例:

```
SELECT bing_tile_polygon(bing_tile_at(47.265511, -122.465691, 4))
```

bing_tile_quadkey(tile)

指定された Bing tile の quadkey を返します。例:

```
SELECT bing_tile_quadkey(bing_tile(52, 143, 10))
```

bing_tile_zoom_level(tile)

指定された Bing tile のズームレベルを整数として返します。例:

```
SELECT bing_tile_zoom_level(bing_tile(52, 143, 10))
```

geometry_to_bing_tiles(geometry, zoom_level)

指定されたズームレベルで指定されたジオメトリを完全にカバーする Bing tile の最小セットを返します。1 から 23 のズームレベルがサポートされています。例 :

```
SELECT geometry_to_bing_tiles(ST_Point(61.56, 58.54), 10)
```

Athena エンジンバージョン 2 における地理空間関数名の変更と新しい関数

このセクションでは、Athena エンジンバージョン 2 での地理空間関数名の変更と、新しく追加された地理空間関数をリストします。Athena エンジンのバージョン 2 でのその他の変更については、「Athena エンジンバージョン 2」を参照してください。

Athena エンジンのバージョニングについては、「Athena エンジンのバージョニング」を参照してください。

Athena エンジンバージョン 2 における地理空間関数名の変更

以下の関数の名称が変更されました。場合によっては、入力の型と出力の型も変更されています。詳細については、対応するリンク先をご覧ください。

以前の関数名	Athena エンジンバージョン 2 以降での関数名
st_coordinate_dimension	ST_CoordDim
st_end_point	ST_EndPoint
st_exterior_ring	ST_ExteriorRing
st_interior_ring_number	ST_NumInteriorRing
st_geometry_from_text	ST_GeometryFromText
st_is_closed	ST_IsClosed
st_is_empty	ST_IsEmpty
st_is_ring	ST_IsRing
st_max_x	ST_XMax
st_max_y	ST_YMax
st_min_x	ST_XMin
st_min_y	ST_YMin

以前の関数名	Athena エンジンバージョン 2 以降での関数名
st_point_number	ST_NumPoints
st_start_point	ST_StartPoint
st_symmetric_difference	ST_SymDifference

Athena エンジンバージョン 2 の新しい地理空間関数

以下の地理空間関数は、Athena エンジンバージョン 2 で新しく追加されたものです。詳細については、対応するリンク先をご覧ください。

コンストラクター関数

- ST_AsBinary
- ST_GeomAsLegacyBinary
- ST_GeomFromBinary
- ST_GeomFromLegacyBinary
- ST_LineString
- ST_MultiPoint
- to_geometry
- to_spherical_geography

オペレーション関数

- geometry_union
- ST_EnvelopeAsPts
- ST_Union

アクセサ関数

- geometry_invalid_reason
- great_circle_distance
- line_locate_point

- simplify_geometry
- ST_ConvexHull
- ST_Distance (球面ジオグラフィ)
- ST_Geometries
- ST_GeometryN
- ST_GeometryType
- ST_InteriorRingN
- ST_InteriorRings
- ST_IsSimple
- ST_IsValid
- ST_NumGeometries
- ST_PointN
- ST_Points

集計関数

- convex_hull_agg
- geometry_union_agg

Bing tile 関数

- bing_tile
- bing_tile (quadkey)
- bing_tile_at
- bing_tiles_around
- bing_tiles_around (半径)
- bing_tile_coordinates
- bing_tile_polygon
- bing_tile_quadkey
- bing_tile_zoom_level
- geometry_to_bing_tiles

例: 地理空間クエリ

このトピックの例では、GitHub で利用可能なサンプルデータから 2 つのテーブルを作成し、そのデータに基づいてテーブルをクエリします。サンプルデータは、あくまで例に過ぎず、正確であることは保証されていません。以下のファイルにあります。

- **earthquakes.csv** – カリフォルニアで発生した地震をリスト化します。この earthquakes テーブルの例では、このデータのフィールドを使用しています。
- **california-counties.json** – ESRI 準拠の GeoJSON 形式でカリフォルニア州の郡データをリスト化します。データには AREA、PERIMETER、STATE、COUNTY、および NAME など多くのフィールドが含まれていますが 、サンプルの counties テーブルでは Name (文字列) と BoundaryShape (バイナリ) の 2 つしか使用していません。

> **Note**
>
> Athena は、com.esri.json.hadoop.EnclosedEsriJsonInputFormat を使用して JSON データを地理空間バイナリ形式に変換します。

以下のコード例は、earthquakes という名前のテーブルを作成します。

```
CREATE external TABLE earthquakes
(
 earthquake_date string,
 latitude double,
 longitude double,
 depth double,
 magnitude double,
 magtype string,
 mbstations string,
 gap string,
 distance string,
 rms string,
 source string,
 eventid string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION 's3://DOC-EXAMPLE-BUCKET/my-query-log/csv/';
```

以下のコード例は、counties という名前のテーブルを作成します。

```
CREATE external TABLE IF NOT EXISTS counties
 (
 Name string,
 BoundaryShape binary
 )
ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.EsriJsonSerDe'
STORED AS INPUTFORMAT 'com.esri.json.hadoop.EnclosedEsriJsonInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/my-query-log/json/';
```

以下のサンプルクエリは、CROSS JOIN 関数を counties および earthquake テーブルで使用します。この例では、境界に ST_POINT で指定される地震位置が含まれる郡のクエリに ST_CONTAINS を使用します。クエリはこれらの郡を名前でグループ化し、発生数順に並べて、それらを降順で返します。

> **Note**
>
> Athena エンジンバージョン 2 以降では、ST_CONTAINS のような関数が VARBINARY 型を入力としてサポートしなくなっています。このため、この例では ST_GeomFromLegacyBinary(varbinary) 関数を使用して boundaryshape バイナリ値をジオメトリに変換します。詳細については、Athena エンジンバージョン 2 リファレンスの「地理空間関数への変更」を参照してください。

```
SELECT counties.name,
        COUNT(*) cnt
FROM counties
CROSS JOIN earthquakes
WHERE ST_CONTAINS (ST_GeomFromLegacyBinary(counties.boundaryshape),
 ST_POINT(earthquakes.longitude, earthquakes.latitude))
GROUP BY  counties.name
ORDER BY  cnt DESC
```

このクエリは以下を返します。

```
+-------------------------+
| name             | cnt |
+-------------------------+
| Kern             | 36  |
```

```
+------------------------+
| San Bernardino   | 35  |
+------------------------+
| Imperial         | 28  |
+------------------------+
| Inyo             | 20  |
+------------------------+
| Los Angeles      | 18  |
+------------------------+
| Riverside        | 14  |
+------------------------+
| Monterey         | 14  |
+------------------------+
| Santa Clara      | 12  |
+------------------------+
| San Benito       | 11  |
+------------------------+
| Fresno           | 11  |
+------------------------+
| San Diego        | 7   |
+------------------------+
| Santa Cruz       | 5   |
+------------------------+
| Ventura          | 3   |
+------------------------+
| San Luis Obispo  | 3   |
+------------------------+
| Orange           | 2   |
+------------------------+
| San Mateo        | 1   |
+------------------------+
```

追加リソース

地理空間クエリのその他の例については、以下のブログ記事を参照してください。

- Extend geospatial queries in Amazon Athena with UDFs and AWS Lambda
- Visualize over 200 years of global climate data using Amazon Athena and Amazon QuickSight
- Querying OpenStreetMap with Amazon Athena

JSON のクエリ

Amazon Athena では、JSON でエンコードされた値の解析、JSON からのデータの抽出、値の検索、および JSON 配列の長さとサイズの確認を行うことができます。

トピック

JSON データ読み取りのベストプラクティス

JavaScript Object Notation (JSON) は、データ構造をテキストとしてエンコードするための一般的な方法です。多くのアプリケーションやツールは、JSON エンコード形式のデータを出力します。

Amazon Athena では、外部データからテーブルを作成し、それらに JSON でエンコードされたデータを含めることができます。このようなタイプのソースデータには、Athena を JSON SerDe ライブラリ と共に使用します。

JSON でエンコードされたデータを読み取るために以下のヒントを使用してください。

- 適切な SerDe、ネイティブ JSON SerDe、`org.apache.hive.hcatalog.data.JsonSerDe`、または OpenX SerDe、`org.openx.data.jsonserde.JsonSerDe` を選択します。詳細については、「JSON SerDe ライブラリ」を参照してください。
- 各 JSON エンコード方式のレコードが、プリティプリントではなく、個別の行に入力されていることを確認します。

> **Note**
>
> SerDe では、各 JSON ドキュメントが、レコード内のフィールドを区切る行終端文字なしの、1 行のテキストに収まっていることを想定しています。JSON テキストがプリティプリント形式の場合、テーブルを作成した後にクエリを実行しようとすると、以下のようなエラーメッセージが表示される場合があります。「HIVE_CURSOR_ERROR: Row is not a valid JSON Object」、または「HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT」。詳細については、GitHub

の OpenX SerDe のドキュメントで「JSON Data Files」(JSON データファイル) を参照してください。

- JSON でエンコードされたデータを、大文字と小文字が区別されない列内に生成します。
- 例に示すように、誤った形式のレコードを無視するオプションを指定します。

```
CREATE EXTERNAL TABLE json_table (
  column_a string,
  column_b int
 )
 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 WITH SERDEPROPERTIES ('ignore.malformed.json' = 'true')
 LOCATION 's3://DOC-EXAMPLE-BUCKET/path/';
```

- Athena で、未確定のスキーマを持つソースデータのフィールドを JSON でエンコードされた文字列に変換します。

Athena が JSON データに基づくテーブルを作成するとき、Athena は既存の事前定義されたスキーマに基づいてデータを解析します。しかし、データには事前定義されたスキーマがないものもあります。このような場合におけるスキーマ管理を簡素化するには、多くの場合、不確定のスキーマがあるソースデータのフィールドを Athena で JSON 文字列に変換してから、JSON SerDe ライブラリ を使用することが有用です。

たとえば、さまざまなセンサーからのイベントを一般的なフィールドで発行する IoT アプリケーションについて考えます。これらのフィールドの 1 つに、イベントを送信するセンサー独自のカスタムペイロードを保存する必要があるとします。この場合、スキーマがわからないため、情報を JSON エンコード形式の文字列として保存することをお勧めします。これを実行するには、以下の例にあるように、Athena テーブル内のデータを JSON に変換します。JSON でエンコードされたデータを Athena データ型に変換することも可能です。

- Athena データ型から JSON への変換
- JSON から Athena データ型への変換

Athena データ型から JSON への変換

Athena データ型を JSON データに変換するには、CAST を使用します。

```
WITH dataset AS (
  SELECT
```

```
    CAST('HELLO ATHENA' AS JSON) AS hello_msg,
    CAST(12345 AS JSON) AS some_int,
    CAST(MAP(ARRAY['a', 'b'], ARRAY[1,2]) AS JSON) AS some_map
)
SELECT * FROM dataset
```

このクエリは以下を返します。

```
+-----------------------------------------+
| hello_msg      | some_int | some_map      |
+-----------------------------------------+
| "HELLO ATHENA" | 12345    | {"a":1,"b":2} |
+-----------------------------------------+
```

JSON から Athena データ型への変換

JSON データを Athena データ型に変換するには、CAST を使用します。

> **Note**
>
> 以下の例では、文字列を JSON エンコード形式にするために、JSON キーワードで開始し、文字列を単一引用符で囲んでいます (例: JSON '12345')。

```
WITH dataset AS (
  SELECT
    CAST(JSON '"HELLO ATHENA"' AS VARCHAR) AS hello_msg,
    CAST(JSON '12345' AS INTEGER) AS some_int,
    CAST(JSON '{"a":1,"b":2}' AS MAP(VARCHAR, INTEGER)) AS some_map
)
SELECT * FROM dataset
```

このクエリは以下を返します。

```
+-------------------------------------+
| hello_msg    | some_int | some_map  |
+-------------------------------------+
| HELLO ATHENA | 12345    | {a:1,b:2} |
```

```
+-----------------------------------+
```

JSON からのデータの抽出

Athena のテーブルにデシリアライズしたくない、JSON でエンコードされた文字列が含まれるデータソースがある場合があります。このような場合でも、Presto の JSON 関数を使用し、このデータに対して SQL オペレーションを実行できます。

次の JSON 文字列をデータセット例として考えます。

```
{"name": "Susan Smith",
"org": "engineering",
"projects":
    [
     {"name":"project1", "completed":false},
     {"name":"project2", "completed":true}
    ]
}
```

例: プロパティの抽出

この JSON 文字列から `name` プロパティと `projects` プロパティを抽出するには、次の例に示すように、`json_extract` 関数を使用します。`json_extract` 関数は、JSON 文字列を含む列を対象にして、`JSONPath` (ドット `.` 表記を含む式など) を使用して JSON 文字列を検索します。

> **Note**
>
> `JSONPath` は、単純なツリー トラバーサルを行います。JSON ドキュメントのルートを示す `$` 記号に続けて、ルートの直下にネストされたピリオドと要素を使用します (例: `$.name`)。

```
WITH dataset AS (
  SELECT '{"name": "Susan Smith",
           "org": "engineering",
           "projects": [{"name":"project1", "completed":false},
           {"name":"project2", "completed":true}]}'
    AS myblob
)
SELECT
  json_extract(myblob, '$.name') AS name,
  json_extract(myblob, '$.projects') AS projects
```

```
FROM dataset
```

返される値は JSON でエンコードされた文字列で、ネイティブ Athena データ型ではありません。

```
+-----------------------------------------------------------------------------------------------
+
| name           | projects
         |
+-----------------------------------------------------------------------------------------------
+
| "Susan Smith"  | [{"name":"project1","completed":false},
{"name":"project2","completed":true}] |
+-----------------------------------------------------------------------------------------------
+
```

JSON 文字列からスカラー値を抽出するには、json_extract_scalar 関数を使用します。これは json_extract に似ていますが、スカラー値 (ブール値、数値、文字列) のみを返します。

> **Note**
>
> 配列、マップ、または構造体に対して json_extract_scalar 関数は使用しないでください。

```
WITH dataset AS (
  SELECT '{"name": "Susan Smith",
           "org": "engineering",
           "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
    AS myblob
)
SELECT
  json_extract_scalar(myblob, '$.name') AS name,
  json_extract_scalar(myblob, '$.projects') AS projects
FROM dataset
```

このクエリは以下を返します。

```
+---------------------------+
| name           | projects |
+---------------------------+
```

```
| Susan Smith    |          |
+---------------------------+
```

配列の例で projects プロパティの最初の要素を取得するには、json_array_get 関数を使用してインデックス位置を指定します。

```
WITH dataset AS (
  SELECT '{"name": "Bob Smith",
           "org": "engineering",
           "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
    AS myblob
)
SELECT json_array_get(json_extract(myblob, '$.projects'), 0) AS item
FROM dataset
```

JSON エンコード形式の配列で、指定したインデックス位置にある値が返されます。

```
+---------------------------------------+
| item                                  |
+---------------------------------------+
| {"name":"project1","completed":false} |
+---------------------------------------+
```

Athena 文字列型を返すには、JSONPath 式内で [] 演算子を使用し、次に json_extract_scalar 関数を使用します。[] の詳細については、「配列要素へのアクセス」を参照してください。

```
WITH dataset AS (
   SELECT '{"name": "Bob Smith",
            "org": "engineering",
            "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
     AS myblob
)
SELECT json_extract_scalar(myblob, '$.projects[0].name') AS project_name
FROM dataset
```

次の結果が返されます。

```
+--------------+
| project_name |
```

```
+--------------+
| project1     |
+--------------+
```

JSON 配列内の値の検索

JSON エンコード形式の配列内に特定の値が存在するかどうかを確認するには、json_array_contains 関数を使用します。

次のクエリでは「project2」に参加しているユーザーの名前を一覧表示します。

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name": "Bob Smith", "org": "legal", "projects": ["project1"]}'),
    (JSON '{"name": "Susan Smith", "org": "engineering", "projects": ["project1",
 "project2", "project3"]}'),
    (JSON '{"name": "Jane Smith", "org": "finance", "projects": ["project1",
 "project2"]}')
  ) AS t (users)
)
SELECT json_extract_scalar(users, '$.name') AS user
FROM dataset
WHERE json_array_contains(json_extract(users, '$.projects'), 'project2')
```

このクエリはユーザーのリストを返します。

```
+-------------+
| user        |
+-------------+
| Susan Smith |
+-------------+
| Jane Smith  |
+-------------+
```

次のクエリ例では、プロジェクトを完了したユーザーの名前とユーザー別の完了したプロジェクト数を一覧表示します。実行するアクションは以下のとおりです。

- SELECT ステートメントをネストして見やすくします。
- プロジェクトの配列を抽出します。
- CAST で配列をキー/値ペアのネイティブ配列に変換します。
- UNNEST 演算子を使用して配列の各要素を抽出します。

- 取得した値をフィルタ処理して完了済みプロジェクトに絞り込み、カウントします。

> **Note**
>
> CAST を使用して MAP する場合、キー要素を VARCHAR (Presto のネイティブ文字列) として指定できますが、値は JSON のままにします。MAP の値は、最初のキー/値ペアが文字列型で、2 番目のペアがブール型であるなど、さまざまな型になるためです。

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name": "Bob Smith",
             "org": "legal",
             "projects": [{"name":"project1", "completed":false}]}'),
    (JSON '{"name": "Susan Smith",
             "org": "engineering",
             "projects": [{"name":"project2", "completed":true},
                          {"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith",
             "org": "finance",
             "projects": [{"name":"project2", "completed":true}]}')
  ) AS t (users)
),
employees AS (
  SELECT users, CAST(json_extract(users, '$.projects') AS
    ARRAY(MAP(VARCHAR, JSON))) AS projects_array
  FROM dataset
),
names AS (
  SELECT json_extract_scalar(users, '$.name') AS name, projects
  FROM employees, UNNEST (projects_array) AS t(projects)
)
SELECT name, count(projects) AS completed_projects FROM names
WHERE cast(element_at(projects, 'completed') AS BOOLEAN) = true
GROUP BY name
```

このクエリは次の結果を返します。

```
+----------------------------------+
| name        | completed_projects |
+----------------------------------+
```

```
| Susan Smith | 2                  |
+---------------------------------+
| Jane Smith  | 1                  |
+---------------------------------+
```

JSON 配列の長さとサイズの取得

例: **json_array_length**

JSON エンコード形式の配列の長さを取得するには、json_array_length 関数を使用します。

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name":
            "Bob Smith",
            "org":
            "legal",
            "projects": [{"name":"project1", "completed":false}]}'),
    (JSON '{"name": "Susan Smith",
            "org": "engineering",
            "projects": [{"name":"project2", "completed":true},
                         {"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith",
             "org": "finance",
             "projects": [{"name":"project2", "completed":true}]}')
  ) AS t (users)
)
SELECT
  json_extract_scalar(users, '$.name') as name,
  json_array_length(json_extract(users, '$.projects')) as count
FROM dataset
ORDER BY count DESC
```

このクエリは次の結果を返します。

```
+---------------------+
| name        | count |
+---------------------+
| Susan Smith | 2     |
+---------------------+
| Bob Smith   | 1     |
+---------------------+
| Jane Smith  | 1     |
```

```
+----------------------+
```

例: **json_size**

JSON エンコード形式の配列やオブジェクトのサイズを取得するには、json_size 関数を使用し、JSON 文字列を含む列とその配列やオブジェクトへの JSONPath 式を指定します。

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name": "Bob Smith", "org": "legal", "projects": [{"name":"project1",
 "completed":false}]}'),
    (JSON '{"name": "Susan Smith", "org": "engineering", "projects":
 [{"name":"project2", "completed":true},{"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith", "org": "finance", "projects": [{"name":"project2",
 "completed":true}]}')
  ) AS t (users)
)
SELECT
  json_extract_scalar(users, '$.name') as name,
  json_size(users, '$.projects') as count
FROM dataset
ORDER BY count DESC
```

このクエリは次の結果を返します。

```
+----------------------+
| name        | count |
+----------------------+
| Susan Smith | 2     |
+----------------------+
| Bob Smith   | 1     |
+----------------------+
| Jane Smith  | 1     |
+----------------------+
```

JSON クエリのトラブルシューティング

JSON 関連のクエリに関する問題のトラブルシューティングについては、「JSON 関連のエラー」、または以下のリソースを参照してください。

- Amazon Athena で JSON データを読み込もうとすると、エラーが発生します。

- Athena で AWS Config からファイルを読み取る際に発生する、「HIVE_CURSOR_ERROR: Row is not a valid JSON Object – JSONException: Duplicate key (HIVE_CURSOR_ERROR: 行は有効な JSON オブジェクトではありません – JSONException: キーの重複)」を解決する方法を教えてください。
- JSON ファイルに複数のレコードがあるにも関わらず、Amazon Athena の SELECT COUNT クエリがレコードを 1 件しか返しません。
- Athena テーブルの行の Amazon S3 ソースファイルを表示するにはどうすればよいですか?

「Amazon Athena での SQL クエリに関する考慮事項と制約事項」も参照してください。

Machine Learning (ML) with Amazon Athena の使用

Machine Learning (ML) with Amazon Athena では、Amazon SageMaker を使用して Machine Learning (ML) 推論を実行する SQL ステートメントの記述に Athena を使用することができます。この機能により、データ分析用の ML モデルへのアクセスが簡単になり、複合型のプログラミング方法を使用して推論を実行する必要がなくなります。

ML with Athena を使用するには、`USING EXTERNAL FUNCTION` 句で ML with Athena 関数を定義します。この関数は、使用する SageMaker モデルのエンドポイントをポイントし、モデルに渡す変数名とデータ型を指定します。クエリの後続の句は、関数を参照してモデルに値を渡します。モデルは、クエリが渡す値に基づいて推論を実行し、推論結果を返します。SageMaker と SageMaker エンドポイントの仕組みに関する詳細については、Amazon SageMaker デベロッパーガイドを参照してください。

ML with Athena と SageMaker 推論を使用して結果セット内の異常な値を検出する例については、AWS ビッグデータブログの記事「Amazon Athena 機械学習推論機能を起動して異常値を検出する」を参照してください。

考慮事項と制約事項

- 利用可能なリージョン - Athena ML は、Athena エンジンバージョン 2 以降がサポートされている AWS リージョン の機能です。
- SageMaker モデルのエンドポイントは **`text/csv`** を受け入れて返す必要があります – データ形式の詳細については、「Amazon SageMaker デベロッパーガイド」の「推論の共通データ形式」を参照してください。
- Athena が CSV ヘッダーを送信しない – SageMaker エンドポイントが `text/csv` である場合、入力ハンドラーは入力の最初の行が CSV ヘッダーであると想定しません。Athena は CSV ヘッダー

を送信しないため、Athena に返される出力に含まれる結果は Athena が想定する行よりも 1 行少なくなり、エラーが発生します。

- SageMaker エンドポイントのスケーリング – 参照された SageMaker モデルのエンドポイントが、エンドポイントへの Athena コールのために十分にスケールアップされていることを確認してください。詳細については、「Amazon SageMaker デベロッパーガイド」の「Amazon SageMaker モデルの自動スケーリング」と、「Amazon SageMaker API リファレンス」の「CreateEndpointConfig」を参照してください。
- IAM 許可 – ML with Athena 関数を指定するクエリを実行するには、クエリを実行する IAM プリンシパルに、参照される SageMaker モデルのエンドポイントへの `sagemaker:InvokeEndpoint` アクションの実行が許可されている必要があります。詳細については、「Athena での ML へのアクセスの許可」を参照してください。
- **`GROUP BY`** 句で ML with Athena 関数を直接使用することはできません

ML with Athena 構文

`USING EXTERNAL FUNCTION` 句は、クエリ内の後続の `SELECT` ステートメントで参照できる 1 つの ML with Athena 関数または複数の関数を指定します。関数名、変数名、および変数と戻り値のデータ型を定義します。

概要

以下の構文は、ML with Athena 関数を指定する `USING EXTERNAL FUNCTION` 句を示しています。

```
USING EXTERNAL FUNCTION ml_function_name (variable1 data_type[, variable2 data_type]
[,...])
RETURNS data_type
SAGEMAKER 'sagemaker_endpoint'
SELECT ml_function_name()
```

パラメータ

USING EXTERNAL FUNCTION ***ml_function_name*** (***variable1data_type***[, ***variable2data_type***][,...])

ml_function_name は、後続のクエリ句で使用できる関数名を定義します。各 *variable data_type* は、SageMaker モデルが入力として受け入れる名前付き変数とそれに対応するデータ型を指定します。指定されるデータ型は、サポートされている Athena データ型である必要があります。

RETURNS ***data_type***

data_type は、SageMaker モデルからの出力として *ml_function_name* がクエリに返す SQL データ型を指定します。

SAGEMAKER '***sagemaker_endpoint***'

sagemaker_endpoint は、SageMaker モデルのエンドポイントを指定します。

SELECT [...]***ml_function_name***(***expression***) [...]

結果を返すために関数変数と SageMaker モデルに値を渡す SELECT クエリです。*ml_function_name* は以前にクエリで定義された関数を指定し、値を渡すために評価される *expression* がその後に続きます。渡される値および返される値は、USING EXTERNAL FUNCTION 句内の関数に指定された対応するデータ型と一致する必要があります。

例

以下の例は、ML with Athena を使用したクエリを示しています。

Example

```
USING EXTERNAL FUNCTION predict_customer_registration(age INTEGER)
    RETURNS DOUBLE
    SAGEMAKER 'xgboost-2019-09-20-04-49-29-303'
SELECT predict_customer_registration(age) AS probability_of_enrolling, customer_id
     FROM "sampledb"."ml_test_dataset"
     WHERE predict_customer_registration(age) < 0.5;
```

お客様の使用例

Machine Learning (ML) with Amazon Athena のプレビュー版を使用する以下の動画では、Athena と SageMaker を併用できる方法が紹介されています。

カスタマーチャーンの予測

以下の動画は、Athena と Amazon SageMaker の機械学習機能を組み合わせてカスタマーチャーンを予測する方法を紹介しています。

Predict customer churn using Amazon Athena and Amazon SageMaker

ボットネットの検出

以下の動画は、ある企業が Amazon Athena と Amazon SageMaker を使用してボットネットを検出する方法を紹介しています。

Detect botnets using Amazon Athena and Amazon SageMaker

ユーザー定義関数を使用したクエリ

Amazon Athena のユーザー定義関数 (UDF) を使用すると、レコードまたはレコードのグループを処理するためのカスタム関数を作成できます。UDF は、パラメータを受け入れ、作業を実行し、結果を返します。

Athena で UDF を使用するには、SQL クエリの `USING EXTERNAL FUNCTION` ステートメントの前に `SELECT` 句を記述します。この `SELECT` ステートメントは UDF を参照し、クエリの実行時に、UDF に渡される変数を定義します。SQL クエリは UDF を呼び出すときに、Java ランタイムを使用して Lambda 関数を呼び出します。UDF は、Java デプロイパッケージのメソッドとして Lambda 関数内で定義されます。Lambda 関数の同じ Java デプロイパッケージで複数の UDF を定義できます。また、`USING EXTERNAL FUNCTION` 句で Lambda 関数の名前も指定します。

Athena UDF 用の Lambda 関数のデプロイには 2 つのオプションがあり、Lambda を使用して直接関数をデプロイすることも、AWS Serverless Application Repository を使用することもできます。UDF の既存の Lambda 関数を検索するには、パブリック AWS Serverless Application Repository またはプライベートリポジトリを検索し、Lambda にデプロイします。また、Java ソースコードを作成または変更して、JAR ファイルにパッケージ化し、Lambda または AWS Serverless Application Repository を使用してデプロイすることもできます。これを開始するための Java ソースコードとパッケージの例については、「Lambda 使用した UDF の作成とデプロイ」を参照してください。Lambda の詳細については、AWS Lambda デベロッパーガイドを参照してください。AWS Serverless Application Repository の詳細については、AWS Serverless Application Repository 開発者ガイドを参照してください。

Athena で UDF を使用してテキストを翻訳および分析する例については、AWS Machine Learning ブログ記事「Amazon Athena、Amazon Translate、および Amazon Comprehend で SQL 関数を使用してテキストを翻訳および分析する」を参照、または video をご覧ください。

UDF を使用して Amazon Athena で地理空間クエリを拡張する例については、AWS Big Data Blog の「Extend geospatial queries in Amazon Athena with UDFs and AWS Lambda」を参照してください。

考慮事項と制約事項

- 組み込み Athena 関数 – Athena の組み込み関数は、高い性能を発揮するように設計されています。可能な場合は、UDF よりも組み込み関数を使用することをお勧めします。組み込み関数の詳細については、「Amazon Athena の関数」を参照してください。
- スカラー UDF 限定 – Athena は、一度に 1 行を処理し、単一の列値を返すスカラー UDF のみをサポートします。Athena は、Lambda を呼び出すたびに行のバッチを UDF に渡しますが、これは並行的に行われる可能性があります。UDF とクエリを設計するときは、この処理がネットワークトラフィックに及ぼす可能性がある影響に注意する必要があります。
- UDF ハンドラ関数は省略形式を使用 – UDF 関数には、省略形式 (フルフォーマットではない) を使用します (例:`package.Class::method` の代わりに `package.Class`)。
- UDF メソッドは小文字を使用する必要がある – UDF メソッドは小文字にする必要があります。キャメルケースは使用できません。
- パラメータを必要とする UDF メソッド — UDF メソッドには少なくとも 1 つの入力パラメータが必要です。入力パラメータなしで定義された UDF を呼び出そうとすると、ランタイム例外が発生する原因となります。UDF はデータレコードに対して関数を実行するためのものですが、引数のない UDF はデータを取り込まないため、例外が発生します。
- Java ランタイムサポート – 現在、Athena UDF は Lambda について Java 8 と Java 11 ランタイムをサポートしています。詳細については、AWS Lambda デベロッパーガイドの「Java による Lambda 関数のビルド」を参照してください。
- IAM 許可 – Athena で UDF クエリステートメントを作成して実行するには、クエリを実行する IAM プリンシパルに、Athena 関数以外のアクションを実行することが許可されている必要があります。詳細については、「Amazon Athena ユーザー定義関数 (UDF) を許可する IAM 許可ポリシーの例」を参照してください。
- Lambda のクォータ – UDF には Lambda のクォータが適用されます。詳細については、AWS Lambda デベロッパーガイドの「Lambda のクォータ」を参照してください。
- 行レベルのフィルタリング – UDF では、Lake Formation の行レベルのフィルタリングはサポートされていません。
- ビュー – UDF でビューを使用することはできません。
- 既知の問題 – 既知の問題に関する最新のリストについては、GitHub の awslabs/aws-athena-query-federation セクションにある「Limitations and Issues」(制限と問題) を参照してください。

動画

Athena での UDF の使用に関する詳細については、以下の動画をご覧ください。

動画: Amazon Athena のユーザー定義関数 (UDF) の概要

以下の動画は、Amazon Athena で UDF を使用して、機密情報のリダクションを行う方法を紹介しています。

> **Note**
>
> このビデオの構文はプレリリースですが、コンセプトは同じです。`AmazonAthenaPreviewFunctionality` ワークグループなしで Athena を使用してください。

Introducing user defined functions (UDFs) in Amazon Athena

動画:Amazon Athena で SQL クエリを使用したテキストフィールドの翻訳、分析、およびリダクション

以下の動画は、その他の AWS のサービス と共に Amazon Athena の UDF を使用して、テキストを翻訳し、分析する方法を紹介しています。

> **Note**
>
> この動画で示されている構文はリリース前のものですが、概念は同じです。正しい構文については、AWS Machine Learning ブログの関連するブログ記事「Amazon Athena、Amazon Translate、および Amazon Comprehend による SQL 関数を使用したテキストの翻訳、修正、分析」を参照してください。

Translate, analyze, and redact text fields using SQL queries in Amazon Athena

UDF クエリ構文

`USING EXTERNAL FUNCTION` 句は、クエリで後続の `SELECT` ステートメントで参照できる UDF または複数の UDF を指定します。UDF のメソッド名と UDF をホストする Lambda 関数の名前が必要です。Lambda 関数名の代わりに、Lambda ARN を使用できます。クロスアカウントのシナリオでは、Lambda ARN が必要です。

概要

```
USING EXTERNAL FUNCTION UDF_name(variable1 data_type[, variable2 data_type][,...])
```

```
RETURNS data_type
LAMBDA 'lambda_function_name_or_ARN'
[, EXTERNAL FUNCTION UDF_name2(variable1 data_type[, variable2 data_type][,...])
RETURNS data_type
LAMBDA 'lambda_function_name_or_ARN'[,...]]
SELECT  [...] UDF_name(expression) [, UDF_name2(expression)] [...]
```

パラメータ

USING EXTERNAL FUNCTION ***UDF_name***(***variable1 data_type***[, ***variable2 data_type***][,...])

UDF_name は UDF の名前を指定します。この名前は、参照される Lambda 関数内の Java メソッドに対応している必要があります。各 *variable data_type* は、UDF が入力として受け入れる名前付きの変数とそれに対応するデータ型を指定します。*data_type* は、以下の表に記載されているサポート対象の Athena データ型の 1 つであり、対応する Java データ型にマップする必要があります。

Athena データ型	Java データ型
TIMESTAMP	java.time.LocalDateTime (UTC)
DATE	java.time.LocalDate (UTC)
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
REAL	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL (「RETURNS ノート」を参照)	java.math.BigDecimal
BIGINT	java.lang.Long
INTEGER	java.lang.Int
VARCHAR	java.lang.String

Athena データ型	Java データ型
VARBINARY	byte[]
BOOLEAN	java.lang.Boolean
ARRAY	java.util.List
ROW	java.util.Map<文字列,オブジェクト>

RETURNS ***data_type***

`data_type` は、UDF が出力として返す SQL データ型を指定します。上記の表にリストされている Athena データ型がサポートされます。`DECIMAL` データ型の場合、この構文 `RETURNS DECIMAL(`*`precision`*`, `*`scale`*`)` を使います。*`##`*と*`####`*は整数です。

LAMBDA '***lambda_function***'

`lambda_function` は、UDF の実行時に呼び出される Lambda 関数の名前を指定します。

SELECT [...] ***UDF_name***(***expression***) [...]

UDF に値を渡し、結果を返す `SELECT` クエリです。*`UDF_Name`* は使用する UDF を指定し、値を渡すために評価される *`expression`* がその後に続きます。渡される値と返される値は、`USING EXTERNAL FUNCTION` 句で UDF に指定された対応するデータ型と一致する必要があります。

例

GitHub にある AthenaUDFHandler.java コードに基づいたクエリの例については、GitHub の「Amazon Athena UDF connector」(Amazon Athena UDF コネクタ) ページを参照してください。

Lambda 使用した UDF の作成とデプロイ

カスタム UDF を作成するには、`UserDefinedFunctionHandler` クラスを拡張して新しい Java クラスを作成します。SDK の UserDefinedFunctionHandler.java のソースコードは、GitHub の awslabs/aws-athena-query-federation/athena-federation-sdk リポジトリで入手できます。このコードには、検証および変更して UDF を作成するために使用できる UDF 実装例が付属しています。

このセクションの手順では、コマンドラインとデプロイから Apache Maven を使用してカスタム UDF Jar ファイルを記述し、構築する方法を示します。

Maven を使用して Athena にカスタム UDF を作成する手順

- SDK のクローン作成と開発環境の準備
- Maven プロジェクトの作成
- Maven プロジェクトへの依存関係とプラグインの追加
- UDF に Java コードを記述
- JAR ファイルのビルド
- JAR を AWS Lambda にデプロイする

SDK のクローン作成と開発環境の準備

始める前に、`sudo yum install git -y` を使用して git がシステムにインストールされていることを確認してください。

AWS クエリフェデレーション SDK をインストールするには

- コマンドラインで次のコマンドを入力して、SDK リポジトリのクローンを作成します。このリポジトリには、SDK、例、データソースコネクタのスイートが含まれています。データソースコネクタの詳細については、「Amazon Athena 横串検索の使用」を参照してください。

```
git clone https://github.com/awslabs/aws-athena-query-federation.git
```

この手順の前提条件をインストールする

Apache Maven、AWS CLI、および AWS Serverless Application Model ビルドツールが既にインストールされている開発マシンで作業している場合は、このステップをスキップできます。

1. クローン作成時に作成した `aws-athena-query-federation` ディレクトリのルートから、開発環境を準備する prepare_dev_env.sh スクリプトを実行します。
2. インストールプロセスによって作成された新しい変数を調達するようにシェルを更新するか、ターミナルセッションを再起動します。

```
source ~/.profile
```

> **Important**
>
> このステップをスキップすると、後で AWS CLI または AWS SAM ビルドツールが Lambda 関数を公開できないというエラーが表示されます。

Maven プロジェクトの作成

次のコマンドを実行して、Maven プロジェクトを作成します。*groupId* を組織の一意の ID に置き換え、*my-athena-udf* をアプリケーション名に置き換えます。詳細については、Apache Maven ドキュメントの「How do I make my first Maven project?」を参照してください。

```
mvn -B archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DgroupId=groupId \
-DartifactId=my-athena-udfs
```

Maven プロジェクトへの依存関係とプラグインの追加

Maven プロジェクト pom.xml ファイルに次の設定を追加します。例については、GitHub の pom.xml ファイルを参照してください。

```
<properties>
    <aws-athena-federation-sdk.version>2022.47.1</aws-athena-federation-sdk.version>
</properties>

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-athena-federation-sdk</artifactId>
        <version>${aws-athena-federation-sdk.version}</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.1</version>
            <configuration>
```

```
                <createDependencyReducedPom>false</createDependencyReducedPom>
                <filters>
                    <filter>
                        <artifact>*:*</artifact>
                        <excludes>
                            <exclude>META-INF/*.SF</exclude>
                            <exclude>META-INF/*.DSA</exclude>
                            <exclude>META-INF/*.RSA</exclude>
                        </excludes>
                    </filter>
                </filters>
            </configuration>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>shade</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

UDF に Java コードを記述

UserDefinedFunctionHandler.java を拡張して、新しいクラスを作成します。クラス内に UDF を記述します。

次の例では、UDF の 2 つの Java メソッド、`compress()` および `decompress()` がクラス `MyUserDefinedFunctions` 内に作成されます。

```
*package *com.mycompany.athena.udfs;

public class MyUserDefinedFunctions
        extends UserDefinedFunctionHandler
{
    private static final String SOURCE_TYPE = "MyCompany";

    public MyUserDefinedFunctions()
    {
        super(SOURCE_TYPE);
    }
```

```
    /**
     * Compresses a valid UTF-8 String using the zlib compression library.
     * Encodes bytes with Base64 encoding scheme.
     *
     * @param input the String to be compressed
     * @return the compressed String
     */
    public String compress(String input)
    {
        byte[] inputBytes = input.getBytes(StandardCharsets.UTF_8);

        // create compressor
        Deflater compressor = new Deflater();
        compressor.setInput(inputBytes);
        compressor.finish();

        // compress bytes to output stream
        byte[] buffer = new byte[4096];
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream(inputBytes.length);
        while (!compressor.finished()) {
            int bytes = compressor.deflate(buffer);
            byteArrayOutputStream.write(buffer, 0, bytes);
        }

        try {
            byteArrayOutputStream.close();
        }
        catch (IOException e) {
            throw new RuntimeException("Failed to close ByteArrayOutputStream", e);
        }

        // return encoded string
        byte[] compressedBytes = byteArrayOutputStream.toByteArray();
        return Base64.getEncoder().encodeToString(compressedBytes);
    }

    /**
     * Decompresses a valid String that has been compressed using the zlib compression
library.
     * Decodes bytes with Base64 decoding scheme.
     *
     * @param input the String to be decompressed
```

```
     * @return the decompressed String
     */
    public String decompress(String input)
    {
        byte[] inputBytes = Base64.getDecoder().decode((input));

        // create decompressor
        Inflater decompressor = new Inflater();
        decompressor.setInput(inputBytes, 0, inputBytes.length);

        // decompress bytes to output stream
        byte[] buffer = new byte[4096];
        ByteArrayOutputStream byteArrayOutputStream = new
 ByteArrayOutputStream(inputBytes.length);
        try {
            while (!decompressor.finished()) {
                int bytes = decompressor.inflate(buffer);
                if (bytes == 0 && decompressor.needsInput()) {
                    throw new DataFormatException("Input is truncated");
                }
                byteArrayOutputStream.write(buffer, 0, bytes);
            }
        }
        catch (DataFormatException e) {
            throw new RuntimeException("Failed to decompress string", e);
        }

        try {
            byteArrayOutputStream.close();
        }
        catch (IOException e) {
            throw new RuntimeException("Failed to close ByteArrayOutputStream", e);
        }

        // return decoded string
        byte[] decompressedBytes = byteArrayOutputStream.toByteArray();
        return new String(decompressedBytes, StandardCharsets.UTF_8);
    }
}
```

JAR ファイルのビルド

`mvn clean install` を実行して、プロジェクトをビルドします。正常にビルドされると、JAR ファイルが `target` という名前のプロジェクトの *`artifactId`*`-`*`version`*`.jar` フォルダに作成されます。ここで、*`artifactId`* は Maven プロジェクトで指定した名前（例: `my-athena-udfs`）です。

JAR を AWS Lambda にデプロイする

コードを Lambda にデプロイする、次の 2 つの方法があります。

- AWS Serverless Application Repository を使用してデプロイする（推奨）
- JAR ファイルから Lambda 関数を作成する

オプション 1: AWS Serverless Application Repository にデプロイする

JAR ファイルを AWS Serverless Application Repository にデプロイするときは、アプリケーションのアーキテクチャを表す AWS SAM テンプレート YAML ファイルを作成します。次に、この YAML ファイルと、アプリケーションのアーティファクトがアップロードされて AWS Serverless Application Repository に対して利用可能になる Simple Storage Service (Amazon S3) バケットを指定します。以下の手順では、先ほどクローンした Athena Query Federation SDK の `athena-query-federation/tools` ディレクトリにある publish.sh スクリプトを使用します。

詳細と要件については、「AWS Serverless Application Repository デベロッパーガイド」の「アプリケーションの公開」、「AWS Serverless Application Model デベロッパーガイド」の「AWS SAM テンプレートの概念」、および「AWS SAM CLI を使用してサーバーレスアプリケーションを公開する」を参照してください。

次の例は、YAML ファイルのパラメータを示しています。同様のパラメータを YAML ファイルに追加し、プロジェクトディレクトリに保存します。完全な例については、GitHub の athena-udf.yaml を参照してください。

```
Transform: 'AWS::Serverless-2016-10-31'
Metadata:
  'AWS::ServerlessRepo::Application':
    Name: MyApplicationName
    Description: 'The description I write for my application'
    Author: 'Author Name'
    Labels:
      - athena-federation
    SemanticVersion: 1.0.0
```

```
Parameters:
  LambdaFunctionName:
    Description: 'The name of the Lambda function that will contain your UDFs.'
    Type: String
  LambdaTimeout:
    Description: 'Maximum Lambda invocation runtime in seconds. (min 1 - 900 max)'
    Default: 900
    Type: Number
  LambdaMemory:
    Description: 'Lambda memory in MB (min 128 - 3008 max).'
    Default: 3008
    Type: Number
Resources:
  ConnectorConfig:
    Type: 'AWS::Serverless::Function'
    Properties:
      FunctionName: !Ref LambdaFunctionName
      Handler: "full.path.to.your.handler. For example,
 com.amazonaws.athena.connectors.udfs.MyUDFHandler"
      CodeUri: "Relative path to your JAR file. For example, ./target/athena-
udfs-1.0.jar"
      Description: "My description of the UDFs that this Lambda function enables."
      Runtime: java8
      Timeout: !Ref LambdaTimeout
      MemorySize: !Ref LambdaMemory
```

YAML ファイルを保存したプロジェクトディレクトリに publish.sh スクリプトをコピーし、次のコマンドを実行します。

```
./publish.sh MyS3Location MyYamlFile
```

たとえば、バケットの場所が s3://DOC-EXAMPLE-BUCKET/mysarapps/athenaudf で、YAML ファイルが my-athena-udfs.yaml として保存された場合:

```
./publish.sh DOC-EXAMPLE-BUCKET/mysarapps/athenaudf my-athena-udfs
```

Lambda 関数を作成するには

1. https://console.aws.amazon.com/lambda/ で Lambda コンソールを開いて [Create function] (関数の作成) をクリックし、[Browse serverless app repository] (Serverless Application Repository の参照) を選択します。

2. [プライベートアプリケーション] を選択し、リストでアプリケーションを見つけるか、キーワードを使用してアプリケーションを検索して選択します。
3. アプリケーションの詳細を確認して指定し、[デプロイ] を選択します。

   これで、Lambda 関数の JAR ファイルで定義されたメソッド名を Athena で UDF として使用できるようになりました。

オプション 2: Lambda 関数を直接作成する

コンソールまたは AWS CLI を使用して、Lambda 関数を直接作成することもできます。以下の例は、Lambda `create-function` CLI コマンドを使用する方法を示しています。

```
aws lambda create-function \
 --function-name MyLambdaFunctionName \
 --runtime java8 \
 --role arn:aws:iam::1234567890123:role/my_lambda_role \
 --handler com.mycompany.athena.udfs.MyUserDefinedFunctions \
 --timeout 900 \
 --zip-file fileb://./target/my-athena-udfs-1.0-SNAPSHOT.jar
```

リージョン間のクエリ

Athena は、Athena を使用しているリージョンとは異なる AWS リージョン で Simple Storage Service (Amazon S3) データをクエリする機能をサポートしています。データを移動するのが実用的でない場合や許可されない場合、あるいは複数のリージョンにわたってデータをクエリする場合には、必要に応じてリージョン間のクエリを実行できます。特定のリージョンで Athena を使用できない場合でも、Athena を使用できる別のリージョンからそのリージョンのデータをクエリできます。

Simple Storage Service (Amazon S3) データが自分のアカウントに属していない場合でも、リージョン内のデータをクエリするためには、そのリージョンで自分のアカウントを有効にする必要があります。米国東部 (オハイオ) など一部のリージョンでは、アカウントの作成時にリージョンへのアクセスが自動的に有効になります。そうでないリージョンでは、リージョンを使用する前に、アカウントをリージョンに「オプトイン」する必要があります。オプトインを必要とするリージョンのリストについては、「Linux インスタンス用の Amazon EC2 ユーザーガイド」の「リージョンとアベイラビリティーゾーン」を参照してください。リージョンへのオプトインに関する具体的な手順については、「Amazon Web Services 全般のリファレンス」の「AWS リージョンの管理」を参照してください。

考慮事項と制約事項

- データアクセス許可 – リージョン間で Athena から Simple Storage Service (Amazon S3) データを正常にクエリするには、アカウントにデータを読み取る許可が必要です。クエリするデータが別のアカウントに属している場合は、目的のデータが含まれている Simple Storage Service (Amazon S3) ロケーションへのアクセス権をそのアカウントから付与してもらう必要があります。
- データ転送料金 – リージョン間のクエリには、Amazon S3 データ転送料金が適用されます。クエリを実行すると、データセットのサイズよりも多くのデータが転送される可能性があります。データのサブセットに対してクエリをテストし、AWS Cost Explorer でコストを確認することから始めることをお勧めします。
- AWS Glue – リージョン間で AWS Glue を使用できます。リージョン間の AWS Glue トラフィックには、追加料金が適用される場合があります。詳細については、「AWS ビッグデータブログ」の「クロスアカウントおよびリージョン間の AWS Glue 接続の作成」を参照してください。
- Simple Storage Service (Amazon S3) 暗号化オプション – リージョン間のクエリでは、SSE-S3 と SSE-KMS の暗号化オプションはサポートされていますが、CSE-KMS はサポートされていません。詳細については、「サポートされる Amazon S3 の暗号化オプション」を参照してください。
- [フェデレーティッドクエリ] — 複数 AWS リージョン にわたる統合クエリの使用はサポートされていません。
- 中国リージョン – クロスリージョンクエリは、中国リージョンではサポートされていません。

上記の条件が満たされていれば、指定した `LOCATION` 値を指す Athena テーブルを作成し、データを透過的にクエリできます。特別な構文は不要です。Athena テーブル作成の詳細については、「Athena でのテーブルの作成」を参照してください。

AWS Glue Data Catalog のクエリ

AWS Glue Data Catalog は多くの AWS のサービス で中央メタデータリポジトリとして使用されるため、データカタログメタデータのクエリが必要になる場合があります。これを実行するには、Athena で SQL クエリを使用できます。Athena を使用して、データベース、テーブル、パーティション、および列などの AWS Glue カタログメタデータをクエリできます。

AWS Glue カタログのメタデータを取得するには、Athena バックエンドで `information_schema` データベースをクエリします。このトピックのクエリ例は、一般的なユースケースのために Athena を使用して AWS Glue カタログメタデータをクエリする方法を示しています。

トピック

考慮事項と制約事項

- `information_schema` データベースをクエリする代わりに、個別の Apache Hive DDL コマンドを使用して、Athena から特定のデータベース、テーブル、ビュー、パーティション、および列のメタデータ情報を抽出することができます。ただし、出力は表形式になりません。
- `information_schema` のクエリのパフォーマンスは、AWS Glue メタデータの量が少ない、または中程度である場合に最も高くなります。大量のメタデータがある場合は、エラーが発生する可能性があります。
- `CREATE VIEW` を使用して `information_schema` データベース上にビューを作成することはできません。

データベースのリスト化と指定したデータベースの検索

このセクションの例では、メタデータ内のデータベースをスキーマ名別にリストする方法を示します。

Example – データベースのリスト化

次のクエリ例は、`information_schema.schemata` テーブルのデータベースを一覧表示します。

```
SELECT schema_name
FROM   information_schema.schemata
LIMIT  10;
```

次の表に項目の例を示します。

6	alb-databas1

7	alb_original_cust
8	alblogsdatabase
9	athena_db_test
10	athena_ddl_db

Example – 指定したデータベースの検索

次のクエリの例では、rdspostgresql はサンプルデータベースです。

```
SELECT schema_name
FROM   information_schema.schemata
WHERE  schema_name = 'rdspostgresql'
```

次の表に項目の例を示します。

	schema_name
1	rdspostgresql

指定したデータベース内のテーブルのリスト化と名前によるテーブルの検索

テーブルのメタデータを一覧表示するには、テーブルスキーマまたはテーブル名でクエリを実行します。

Example – テーブルのスキーマ別のリスト化

次のクエリは、rdspostgresql テーブルスキーマを使用するテーブルを一覧表示します。

```
SELECT table_schema,
       table_name,
       table_type
FROM   information_schema.tables
WHERE  table_schema = 'rdspostgresql'
```

次の表は、サンプル結果を示しています。

	table_schema	table_name	table_type
1	rdspostgresql	rdspostgresqldb1_public_account	BASE TABLE

Example – 名前によるテーブルの検索

次のクエリは、テーブル athena1 のメタデータ情報を取得します。

```
SELECT table_schema,
       table_name,
       table_type
FROM   information_schema.tables
WHERE  table_name = 'athena1'
```

次の表は、サンプル結果を示しています。

	table_schema	table_name	table_type
1	デフォルト	athena1	BASE TABLE

特定のテーブルのパーティションのリスト化

以下の例のように、SHOW PARTITIONS *table_name* を使用して、指定したテーブルのパーティションをリスト化できます。

```
SHOW PARTITIONS cloudtrail_logs_test2
```

$partitions メタデータクエリを使用して、特定のテーブルのパーティション番号とパーティション値を一覧表示することもできます。

Example — $partitions 構文を使用したテーブルのパーティションのクエリ

以下のクエリの例では、$partitions 構文を使用して、テーブル cloudtrail_logs_test2 のパーティションを一覧表示します。

```
SELECT * FROM default."cloudtrail_logs_test2$partitions" ORDER BY partition_number
```

次の表に項目の例を示します。

	table_catalog	table_schema	table_name	年	月	日
1	awsdatacatalog	デフォルト	cloudtrail_logs_test2	2020	08	10
2	awsdatacatalog	デフォルト	cloudtrail_logs_test2	2020	08	11
3	awsdatacatalog	デフォルト	cloudtrail_logs_test2	2020	08	12

全テーブルでのすべての列の一覧表示

AwsDataCatalog にある全テーブル、あるいは AwsDataCatalog にある特定のデータベース内の全テーブルで、すべての列を一覧表示できます。

- AwsDataCatalog にある全データベースのすべての列を一覧表示するには、クエリ SELECT * FROM information_schema.columns を使用します。
- 結果を特定のデータベースに制限するには、WHERE 句で table_schema='*database_name*' を使用します。

Example — 特定のデータベース内の全テーブルですべての列を一覧表示する

次のクエリ例では、データベース webdata にある全テーブルのすべての列を一覧表示します。

```
SELECT * FROM information_schema.columns WHERE table_schema = 'webdata'
```

特定のテーブルに共通する列を一覧表示する

データベース内の特定のテーブルに共通する列を一覧表示できます。

- SELECT column_name FROM information_schema.columns 構文を使用してください。
- WHERE 句には、以下の構文 WHERE table_name IN ('table1', 'table2') を使用します。

Example — 同じデータベース内の 2 つのテーブルに共通する列を一覧表示する

次のクエリ例では、テーブル table1 と table2 に共通する列を一覧表示しています。

```
SELECT column_name
FROM information_schema.columns
WHERE table_name IN ('table1', 'table2')
GROUP BY column_name
HAVING COUNT(*) > 1;
```

指定したテーブルまたはビューの列のリスト化または検索

指定したデータベースとテーブルで、テーブルのすべての列、ビューのすべての列を一覧表示したり、名前で列を検索したりできます。

列を一覧表示するには、SELECT * クエリを使用します。FROM 句で、information_schema.columns を指定します。WHERE 句で、table_schema='*database_name*' を使用してデータベースを指定し、table_name = '*table_name*' を使用して一覧表示する列を含むテーブルまたはビューを指定します。

Example – 指定したテーブルのすべての列のリスト化

次のクエリ例は、テーブル rdspostgresqldb1_public_account のすべての列を一覧表示します。

```
SELECT *
FROM   information_schema.columns
WHERE  table_schema = 'rdspostgresql'
       AND table_name = 'rdspostgresqldb1_public_account'
```

次の表に項目の例を示します。

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	コメント	extra_info
1	awsdatacatalog	rdspostgresql	rdspostgresqldb1_p	passwor	1		はい	varcha		

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_t	コメント	extra_info
			ublic_account							
2	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	user_id	2		はい	intege		
3	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	created_n	3		はい	timest		
4	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	last_login	4		はい	timest		
5	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	email	5		はい	varcha		
6	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	usernam	6		はい	varcha		

Example – 指定したビューの列のリスト化

次のクエリ例は、ビュー `default` の `arrayview` データベース内のすべての列を一覧表示します。

```
SELECT *
```

```
FROM   information_schema.columns
WHERE  table_schema = 'default'
       AND table_name = 'arrayview'
```

次の表に項目の例を示します。

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	コメント	extra_info
1	awsdatacatalog	デフォルト	arrayview	searchdate	1		はい	varchar		
2	awsdatacatalog	デフォルト	arrayview	sid	2		はい	varchar		
3	awsdatacatalog	デフォルト	arrayview	btid	3		はい	varchar		
4	awsdatacatalog	デフォルト	arrayview	p	4		はい	varchar		
5	awsdatacatalog	デフォルト	arrayview	infantprice	5		はい	varchar		
6	awsdatacatalog	デフォルト	arrayview	sump	6		はい	varchar		
7	awsdatacatalog	デフォルト	arrayview	journeymaparray	7		はい	array(varchar)		

Example – 指定したデータベースとテーブルでの名前による列の検索

次のクエリ例は、`sid` データベースの `arrayview` ビューで `default` 列のメタデータを検索します。

```
SELECT *
FROM   information_schema.columns
```

```
WHERE  table_schema = 'default'
       AND table_name = 'arrayview'
       AND column_name='sid'
```

次の表は、サンプル結果を示しています。

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_t	コメント	extra_info
1	awsdatacatalog	デフォルト	arrayview	sid	2		はい	varcha		

AWS のサービス ログのクエリ

このセクションでは、Amazon Athena を使用して一般的なデータセット (AWS CloudTrail ログ、Amazon CloudFront ログ、Classic Load Balancer ログ、Application Load Balancer ログ、Amazon VPC フローログ、Network Load Balancer ログなど) をクエリするための手順をいくつか紹介します。

このセクションのタスクでは Athena コンソールを使用しますが、Athena JDBC ドライバー、AWS CLI、または Amazon Athena API リファレンスなどの他のツールを使うこともできます。

Athena での AWS CloudFormation を使用した AWS のサービス のログのテーブル、パーティション、およびクエリ例の自動作成については、AWS Big Dataブログの「Amazon Athena でのAWS のサービス ログのテーブル作成とクエリの自動化」を参照してください。AWS Glue 用の Python ライブラリを使用して、AWS のサービス のログを処理し、Athena でクエリを実行するための共通のフレームワークを作成する方法については、「Amazon Athena を使用して AWS のサービス のログに対するクエリを簡単に実行する」を参照してください。

このセクションのトピックでは、Athena と、クエリするデータがが格納される Amazon S3 バケットにアクセスするための適切なアクセス許可が設定済みであることを前提としています。詳細については、セットアップおよび開始を参照してください。

トピック

- Application Load Balancer ログのクエリ

- Classic Load Balancer ログのクエリ
- Amazon CloudFront ログのクエリ
- AWS CloudTrail ログのクエリ
- Amazon EMR ログのクエリ
- AWS Global Accelerator フローログのクエリ
- Amazon GuardDuty の結果のクエリ
- AWS Network Firewall ログのクエリ
- Network Load Balancer のログのクエリ
- Amazon Route 53 Resolver クエリログのクエリ
- Amazon SES イベントログのクエリ
- Amazon VPC フローログのクエリ
- AWS WAF ログのクエリ

Application Load Balancer ログのクエリ

Application Load Balancer は、Elastic Load Balancing の負荷分散オプションであり、コンテナを使用したマイクロサービスのデプロイメントでのトラフィックの分散を可能にします。Application Load Balancer のログをクエリすることで、トラフィックの送信元、レイテンシー、Elastic Load Balancing インスタンスとバックエンドアプリケーションとの間で転送されるバイト数を確認できます。詳細については、「Application Load Balancer ユーザーガイド」の「Access logs for your Application Load Balancer」と「Connection logs for your Application Load Balancer」を参照してください。

トピック

- 前提条件
- ALB アクセスログ用のテーブルの作成
- パーティション射影を使用した Athena での ALB アクセスログ用テーブルの作成
- ALB アクセスログのクエリ例
- ALB 接続ログ用テーブルの作成
- パーティション射影を使用した Athena での ALB 接続ログ用テーブルの作成
- ALB 接続ログのクエリ例

- 追加リソース

前提条件

- アクセスログまたは接続ログ を有効にして、Application Load Balancer ログを Amazon S3 バケットに保存できるようにします。
- Athena 用に作成するテーブルを保存するデータベース。データベースを作成するには、Athena または AWS Glue コンソールを使用します。詳細については、本ガイドの「Athena でのデータベースの作成」または「AWS Glue デベロッパーガイド」の「AWS Glue コンソールでのデータベースの操作」を参照してください。

ALB アクセスログ用のテーブルの作成

1. 次の CREATE TABLE ステートメントをコピーして Athena コンソールのクエリエディタに貼り付けます。Athena コンソールを開始する方法については、「開始」を参照してください。LOCATION 句内のパスを Amazon S3 アクセスログフォルダの場所に置き換えます。アクセスログファイルの場所に関する詳細については、「Application Load Balancer ユーザーガイド」の「アクセスログファイル」を参照してください。ログファイルの各フィールドに関する詳細については、「アクセスログのエントリ」を参照してください。

> **Note**
>
> 次の CREATE TABLE ステートメントには、最近追加された classification 列、classification_reason 列、および traceability_id 列が含まれています。これらのエントリを含まないApplication Load Balancer アクセスログのテーブルを作成するには、CREATE TABLE ステートメントから対応する列を削除し、それに応じて正規表現を変更します。

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_access_logs (
             type string,
             time string,
             elb string,
             client_ip string,
             client_port int,
             target_ip string,
             target_port int,
             request_processing_time double,
```

```
            target_processing_time double,
            response_processing_time double,
            elb_status_code int,
            target_status_code string,
            received_bytes bigint,
            sent_bytes bigint,
            request_verb string,
            request_url string,
            request_proto string,
            user_agent string,
            ssl_cipher string,
            ssl_protocol string,
            target_group_arn string,
            trace_id string,
            domain_name string,
            chosen_cert_arn string,
            matched_rule_priority string,
            request_creation_time string,
            actions_executed string,
            redirect_url string,
            lambda_error_reason string,
            target_port_list string,
            target_status_code_list string,
            classification string,
            classification_reason string,
            traceability_id string
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \"([^ ]*) (.*) (-
 |[^ ]*)\" \"([^\"]*)\" ([A-Z0-9-_]+) ([A-Za-z0-9.-]*) ([^ ]*) \"([^\"]*)\" \"([^
\"]*)\" \"([^\"]*)\" ([-.0-9]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\" \"([^ ]*)\" \"([^
\s]+?)\" \"([^\s]+)\" \"([^ ]*)\" \"([^ ]*)\" ?([^ ]*)?( .*)?')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/access-log-folder-path/'
```

2. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が `alb_access_logs` テーブルを登録し、その中のデータに対してクエリを発行できるように準備します。

パーティション射影を使用した Athena での ALB アクセスログ用テーブルの作成

ALB アクセスログにはパーティションスキームを事前に指定できる既知の構造があるため、Athena のパーティション射影機能を使用することで、クエリの実行時間を短縮し、パーティション管理を自動化することが可能です。新しいデータが追加されると、パーティション射影は新しいパーティションを自動で追加します。このため、ALTER TABLE ADD PARTITION を使用してパーティションを手動で追加する必要がなくなります。

以下の CREATE TABLE ステートメント例は、単一の AWS リージョンの指定された日付から現在までの ALB アクセスログに、パーティション射影を自動的に使用します。ステートメントは、前のセクションの例に基づいていますが、PARTITIONED BY 句および TBLPROPERTIES 句を追加して、パーティション射影を有効にしています。LOCATION 句および storage.location.template 句では、プレースホルダーを ALB アクセスログの Amazon S3 バケットの場所を特定する値に置き換えます。アクセスログファイルの場所に関する詳細については、「Application Load Balancer ユーザーガイド」の「アクセスログファイル」を参照してください。projection.day.range では、*2022*/*01*/*01* を使用を開始する日に置き換えます。クエリが正常に実行されると、テーブルをクエリできます。パーティションをロードするのに、ALTER TABLE ADD PARTITION を実行する必要はありません。ログファイルの各フィールドに関する詳細については、「アクセスログのエントリ」を参照してください。

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_access_logs (
             type string,
             time string,
             elb string,
             client_ip string,
             client_port int,
             target_ip string,
             target_port int,
             request_processing_time double,
             target_processing_time double,
             response_processing_time double,
             elb_status_code int,
             target_status_code string,
             received_bytes bigint,
             sent_bytes bigint,
             request_verb string,
             request_url string,
             request_proto string,
             user_agent string,
             ssl_cipher string,
             ssl_protocol string,
```

```
            target_group_arn string,
            trace_id string,
            domain_name string,
            chosen_cert_arn string,
            matched_rule_priority string,
            request_creation_time string,
            actions_executed string,
            redirect_url string,
            lambda_error_reason string,
            target_port_list string,
            target_status_code_list string,
            classification string,
            classification_reason string,
            traceability_id string
            )
            PARTITIONED BY
            (
             day STRING
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \"([^ ]*) (.*) (- |
[^ ]*)\" \"([^\"]*)\" ([A-Z0-9-_]+) ([A-Za-z0-9.-]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\"
 \"([^\"]*)\" ([-.0-9]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\" \"([^ ]*)\" \"([^\s]+?)\"
 \"([^\s]+)\" \"([^ ]*)\" \"([^ ]*)\" ?([^ ]*)?( .*)?')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-NUMBER>/
elasticloadbalancing/<REGION>/'
            TBLPROPERTIES
            (
             "projection.enabled" = "true",
             "projection.day.type" = "date",
             "projection.day.range" = "2022/01/01,NOW",
             "projection.day.format" = "yyyy/MM/dd",
             "projection.day.interval" = "1",
             "projection.day.interval.unit" = "DAYS",
             "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-
NUMBER>/elasticloadbalancing/<REGION>/${day}"
            )
```

パーティション射影の詳細については、「Amazon Athena でのパーティション射影」を参照してください。

ALB アクセスログのクエリ例

次のクエリでは、ロードバランサーで受信し、クライアント IP アドレス別にグループ分けした HTTP GET リクエストの数をカウントします。

```
SELECT COUNT(request_verb) AS
 count,
 request_verb,
 client_ip
FROM alb_logs
GROUP BY request_verb, client_ip
LIMIT 100;
```

別のクエリでは、Safari ブラウザのユーザーがアクセスした URL を表示します。

```
SELECT request_url
FROM alb_logs
WHERE user_agent LIKE '%Safari%'
LIMIT 10;
```

次のクエリでは、ELB ステータスコードの値が 500 以上であるレコードが表示されます。

```
SELECT * FROM alb_logs
WHERE elb_status_code >= 500
```

次の例は、datetime でログを解析する方法を示しています。

```
SELECT client_ip, sum(received_bytes)
FROM alb_logs
WHERE parse_datetime(time,'yyyy-MM-dd''T''HH:mm:ss.SSSSSS''Z')
     BETWEEN parse_datetime('2018-05-30-12:00:00','yyyy-MM-dd-HH:mm:ss')
     AND parse_datetime('2018-05-31-00:00:00','yyyy-MM-dd-HH:mm:ss')
GROUP BY client_ip;
```

次のクエリは、指定された日からのすべての ALB ログに対してパーティション射影を使用するテーブルをクエリします。

```
SELECT *
FROM alb_logs
WHERE day = '2022/02/12'
```

ALB 接続ログ用テーブルの作成

1. 次の CREATE TABLE ステートメントをコピーして Athena コンソールのクエリエディタに貼り付けます。Athena コンソールを開始する方法については、「開始」を参照してください。LOCATION 句内のパスを Amazon S3 接続ログフォルダの場所に置き換えます。接続ログファイルの場所に関する詳細については、「Application Load Balancer ユーザーガイド」の「接続ログファイル」を参照してください。ログファイルの各フィールドに関する詳細については、「接続ログエントリ」を参照してください。

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS alb_connection_logs (
            time string,
            client_ip string,
            client_port int,
            listener_port int,
            tls_protocol string,
            tls_cipher string,
            tls_handshake_latency double,
            leaf_client_cert_subject string,
            leaf_client_cert_validity string,
            leaf_client_cert_serial_number string,
            tls_verify_status string,
            traceability_id string
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
             '([^ ]*) ([^ ]*) ([0-9]*) ([0-9]*) ([A-Za-z0-9.-]*) ([^ ]*) ([-.0-9]*)
    \"([^\"]*)\" ([^ ]*) ([^ ]*) ([^ ]*) ?([^ ]*)?( .*)?')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/connection-log-folder-path/'
   ```

2. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が alb_connection_logs テーブルを登録し、その中のデータに対してクエリを発行できるように準備します。

パーティション射影を使用した Athena での ALB 接続ログ用テーブルの作成

ALB 接続ログにはパーティションスキームを事前に指定できる既知の構造があるため、Athena のパーティション射影機能を使用することで、クエリの実行時間を短縮し、パーティション管理を自動化することが可能です。新しいデータが追加されると、パーティション射影は新しいパーティションを自動で追加します。このため、ALTER TABLE ADD PARTITION を使用してパーティションを手動で追加する必要がなくなります。

以下の CREATE TABLE ステートメント例は、単一の AWS リージョンの指定された日付から現在までの ALB 接続ログに、パーティション射影を自動的に使用します。ステートメントは、前のセクションの例に基づいていますが、PARTITIONED BY 句および TBLPROPERTIES 句を追加して、パーティション射影を有効にしています。LOCATION 句および storage.location.template 句では、プレースホルダーを ALB 接続ログの Amazon S3 バケットの場所を特定する値に置き換えます。接続ログファイルの場所に関する詳細については、「Application Load Balancer ユーザーガイド」の「接続ログファイル」を参照してください。projection.day.range では、*2023*/*01*/*01* を使用したい開始日に置き換えます。クエリが正常に実行されると、テーブルをクエリできます。パーティションをロードするのに、ALTER TABLE ADD PARTITION を実行する必要はありません。ログファイルの各フィールドに関する詳細については、「接続ログエントリ」を参照してください。

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_connection_logs (
         time string,
         client_ip string,
         client_port int,
         listener_port int,
         tls_protocol string,
         tls_cipher string,
         tls_handshake_latency double,
         leaf_client_cert_subject string,
         leaf_client_cert_validity string,
         leaf_client_cert_serial_number string,
         tls_verify_status string,
         traceability_id string
         )
            PARTITIONED BY
            (
             day STRING
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
```

```
             '([^ ]*) ([^ ]*) ([0-9]*) ([0-9]*) ([A-Za-z0-9.-]*) ([^ ]*) ([-.0-9]*)
 \"([^\"]*)\" ([^ ]*) ([^ ]*) ([^ ]*) ?([^ ]*)?( .*)?')
           LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-NUMBER>/
elasticloadbalancing/<REGION>/'
           TBLPROPERTIES
           (
            "projection.enabled" = "true",
            "projection.day.type" = "date",
            "projection.day.range" = "2023/01/01,NOW",
            "projection.day.format" = "yyyy/MM/dd",
            "projection.day.interval" = "1",
            "projection.day.interval.unit" = "DAYS",
            "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-
NUMBER>/elasticloadbalancing/<REGION>/${day}"
           )
```

パーティション射影の詳細については、「Amazon Athena でのパーティション射影」を参照してください。

ALB 接続ログのクエリ例

以下のクエリは、tls_verify_status の値が 'Success' ではなかったオカレンスの数をカウントし、クライアント IP アドレスごとにグループ化します。

```
SELECT DISTINCT client_ip, count() AS count FROM alb_connection_logs
WHERE tls_verify_status != 'Success'
GROUP BY client_ip
ORDER BY count() DESC;
```

以下のクエリは、指定された時間範囲内で tls_handshake_latency の値が 2 秒を超えたオカレンスを検索します。

```
SELECT * FROM alb_connection_logs
WHERE
  (
    parse_datetime(time, 'yyyy-MM-dd''T''HH:mm:ss.SSSSSS''Z')
    BETWEEN
    parse_datetime('2024-01-01-00:00:00', 'yyyy-MM-dd-HH:mm:ss')
    AND
    parse_datetime('2024-03-20-00:00:00', 'yyyy-MM-dd-HH:mm:ss')
  )
  AND
```

```
    (tls_handshake_latency >= 2.0);
```

追加リソース

- AWS ナレッジセンターの「Amazon Athena で Application Load Balancer のアクセスログを分析する方法を教えてください」。
- Elastic Load Balancing における HTTP ステータスコードについては、「Application Load Balancer ユーザーガイド」の「Application Load Balancer のトラブルシューティング」を参照してください。
- AWS Big Data Blog の「Catalog and analyze Application Load Balancer logs more efficiently with AWS Glue custom classifiers and Amazon Athena」。

Classic Load Balancer ログのクエリ

Classic Load Balancer ログを使用して、Elastic Load Balancing インスタンスおよびバックエンドアプリケーションとの間で送受信されるトラフィックパターンを分析して理解します。トラフィックの送信元、レイテンシー、および転送されたバイト数を確認できます。

Elastic Load Balancing ログを分析する前に、宛先の Amazon S3 バケットでの保存用にそれらを設定します。詳細については、「Classic Load Balancer のアクセスログの有効化」を参照してください。

- Elastic Load Balancing ログのテーブルを作成する
- Elastic Load Balancing のクエリ例

Elastic Load Balancing ログのテーブルを作成する

1. 以下の DDL ステートメントをコピーして Athena コンソール内に貼り付けます。Elastic Load Balancing ログレコードの構文をチェックします。以下のクエリを更新して、最新バージョンのレコードのための列と Regex 構文を含めることが必要になる場合があります。

```
CREATE EXTERNAL TABLE IF NOT EXISTS elb_logs (

 timestamp string,
 elb_name string,
 request_ip string,
 request_port int,
 backend_ip string,
```

```
 backend_port int,
 request_processing_time double,
 backend_processing_time double,
 client_response_time double,
 elb_response_code string,
 backend_response_code string,
 received_bytes bigint,
 sent_bytes bigint,
 request_verb string,
 url string,
 protocol string,
 user_agent string,
 ssl_cipher string,
 ssl_protocol string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
 'serialization.format' = '1',
 'input.regex' = '([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \\\"([^ ]*)
 ([^ ]*) (- |[^ ]*)\\\" (\"[^\"]*\") ([A-Z0-9-]+) ([A-Za-z0-9.-]*)$'
)
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/AWS_account_ID/elasticloadbalancing/';
```

2. LOCATION Amazon S3 バケットを変更して、Elastic Load Balancing ログの宛先を指定します。
3. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が elb_logs テーブルを登録して、その中のデータをクエリ向けに準備します。詳細については、「Elastic Load Balancing のクエリ例」を参照してください。

Elastic Load Balancing のクエリ例

以下の例のようなクエリを使用します。この例では、4XX または 5XX エラーレスポンスコードを返したバックエンドアプリケーションサーバーを示しています。LIMIT 演算子を使用すると、一度にクエリするログの数を制限できます。

```
SELECT
 timestamp,
 elb_name,
 backend_ip,
 backend_response_code
FROM elb_logs
WHERE backend_response_code LIKE '4%' OR
```

```
      backend_response_code LIKE '5%'
LIMIT 100;
```

後続のクエリを使用して、バックエンド IP アドレスと Elastic Load Balancing インスタンス名でグループ分けされたすべてのトランザクションの応答時間を合算します。

```
SELECT sum(backend_processing_time) AS
 total_ms,
 elb_name,
 backend_ip
FROM elb_logs WHERE backend_ip <> ''
GROUP BY backend_ip, elb_name
LIMIT 100;
```

詳細については、「Athena を使用した S3 のデータの分析」を参照してください。

Amazon CloudFront ログのクエリ

Amazon CloudFront CDN を設定して、ウェブディストリビューションのアクセスログを Amazon Simple Storage Service にエクスポートできます。これらのログを使用して、CloudFront によって提供されるウェブプロパティ全体でのユーザーのサーフィンパターンを調べます。

ログのクエリを開始する前に、使用する CloudFront ディストリビューションでウェブディストリビューションのアクセスログを有効にします。詳細については、「Amazon CloudFront デベロッパーガイド」の「アクセスログ」を参照してください。ログを保存する Amazon S3 バケットをメモしておきます。

- CloudFront 標準ログ用テーブルの作成
- CloudFront リアルタイムログ用テーブルの作成
- 標準 CloudFront ログのクエリ例

CloudFront 標準ログ用テーブルの作成

> **Note**
>
> この手順は、CloudFront にあるウェブディストリビューションのアクセスログで機能します。RTMP ディストリビューションのストリーミングログには該当しません。

CloudFront 標準ログファイルフィールド用のテーブルを作成する

1. 次の DDL ステートメントの例をコピーして Athena コンソールのクエリエディタに貼り付けます。例のステートメントでは、「Amazon CloudFront デベロッパーガイド」の「標準ログファイルフィールド」セクションに記載されているログファイルフィールドを使用しています。LOCATION をログを保存する Simple Storage Service (Amazon S3) バケットに変更します。クエリエディタの使用については、「開始」を参照してください。

   このクエリは、フィールドがタブ文字で区切られることを示すために ROW FORMAT DELIMITED と FIELDS TERMINATED BY '\t' を指定します。ROW FORMAT DELIMITED には、Athena がデフォルトで LazySimpleSerDe を使用します。列 date は Athena の予約語であるため、バックティック (`) を使用してエスケープされています。詳細については、予約済みキーワード を参照してください。

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_standard_logs (
  `date` DATE,
  time STRING,
  x_edge_location STRING,
  sc_bytes BIGINT,
  c_ip STRING,
  cs_method STRING,
  cs_host STRING,
  cs_uri_stem STRING,
  sc_status INT,
  cs_referrer STRING,
  cs_user_agent STRING,
  cs_uri_query STRING,
  cs_cookie STRING,
  x_edge_result_type STRING,
  x_edge_request_id STRING,
  x_host_header STRING,
  cs_protocol STRING,
  cs_bytes BIGINT,
  time_taken FLOAT,
  x_forwarded_for STRING,
  ssl_protocol STRING,
  ssl_cipher STRING,
  x_edge_response_result_type STRING,
  cs_protocol_version STRING,
  fle_status STRING,
  fle_encrypted_fields INT,
  c_port INT,
```

```
  time_to_first_byte FLOAT,
  x_edge_detailed_result_type STRING,
  sc_content_type STRING,
  sc_content_len BIGINT,
  sc_range_start BIGINT,
  sc_range_end BIGINT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
TBLPROPERTIES ( 'skip.header.line.count'='2' )
```

2. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が `cloudfront_standard_logs` テーブルを登録し、その中のデータに対してクエリを発行できるように準備します。

CloudFront リアルタイムログ用テーブルの作成

CloudFront リアルタイムログファイルフィールド用のテーブルを作成する

1. 次の DDL ステートメントの例をコピーして Athena コンソールのクエリエディタに貼り付けます。ステートメント例では、「Amazon CloudFront 開発者ガイド」の「Real-time logs」セクションに記載されているログファイルフィールドを使用しています。`LOCATION` をログを保存する Simple Storage Service (Amazon S3) バケットに変更します。クエリエディタの使用については、「開始」を参照してください。

   このクエリは、フィールドがタブ文字で区切られることを示すために `ROW FORMAT DELIMITED` と `FIELDS TERMINATED BY '\t'` を指定します。`ROW FORMAT DELIMITED` には、Athena がデフォルトで LazySimpleSerDe を使用します。列 `timestamp` は Athena の予約語であるため、バックティック (`) を使用してエスケープされています。詳細については、予約済みキーワードを参照してください。

   以下の例には、利用可能なフィールドのすべてが含まれています。不要なフィールドは、コメントアウトまたは削除することができます。

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_real_time_logs (
`timestamp` STRING,
c_ip STRING,
time_to_first_byte BIGINT,
sc_status BIGINT,
sc_bytes BIGINT,
```

```
cs_method STRING,
cs_protocol STRING,
cs_host STRING,
cs_uri_stem STRING,
cs_bytes BIGINT,
x_edge_location STRING,
x_edge_request_id STRING,
x_host_header STRING,
time_taken BIGINT,
cs_protocol_version STRING,
c_ip_version STRING,
cs_user_agent STRING,
cs_referer STRING,
cs_cookie STRING,
cs_uri_query STRING,
x_edge_response_result_type STRING,
x_forwarded_for STRING,
ssl_protocol STRING,
ssl_cipher STRING,
x_edge_result_type STRING,
fle_encrypted_fields STRING,
fle_status STRING,
sc_content_type STRING,
sc_content_len BIGINT,
sc_range_start STRING,
sc_range_end STRING,
c_port BIGINT,
x_edge_detailed_result_type STRING,
c_country STRING,
cs_accept_encoding STRING,
cs_accept STRING,
cache_behavior_path_pattern STRING,
cs_headers STRING,
cs_header_names STRING,
cs_headers_count BIGINT,
primary_distribution_id STRING,
primary_distribution_dns_name STRING,
origin_fbl STRING,
origin_lbl STRING,
asn STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

```
TBLPROPERTIES ( 'skip.header.line.count'='2' )
```

2. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が `cloudfront_real_time_logs` テーブルを登録し、その中のデータに対してクエリを発行できるように準備します。

標準 CloudFront ログのクエリ例

以下のクエリは、2018 年 6 月 9 日から 6 月 11 日の間に CloudFront によって提供されたバイト数を Cに で処理されたバイト数を集計します。date 列名は予約語であるため、二重引用符で囲みます。

```
SELECT SUM(bytes) AS total_bytes
FROM cloudfront_standard_logs
WHERE "date" BETWEEN DATE '2018-06-09' AND DATE '2018-06-11'
LIMIT 100;
```

クエリ結果から重複する行 (重複する空の行など) を削除するには、次の例のように `SELECT DISTINCT` ステートメントを使用します。

```
SELECT DISTINCT *
FROM cloudfront_standard_logs
LIMIT 10;
```

追加リソース

Athena を使用した CloudFront ログのクエリについての詳細は、「AWS ビッグデータブログ」の以下の記事を参照してください。

Amazon Athena を使用して AWS のサービス ログを簡単にクエリする (2019 年 5 月 29 日)。

Amazon CloudFront のアクセスログを大規模に分析する (December 21, 2018 年 12 月 21 日)。

AWS Lambda、Amazon Athena、Amazon Managed Service for Apache Flink を使用して Amazon CloudFront アクセスログを分析するためのサーバーレスアーキテクチャを構築する (2017 年 5 月 26 日)。

AWS CloudTrail ログのクエリ

AWS CloudTrail は、Amazon Web Services アカウントの AWS API コールとイベントを記録するサービスです。

CloudTrail ログには、コンソールを含めた AWS のサービス に対して発行された、あらゆる API コールの詳細が記載されます。CloudTrail は暗号化されたログファイルを生成して、それらを Amazon S3 に保存します。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。

> **Note**
>
> アカウント、リージョン、および日付にわたって CloudTrail イベント情報に対して SQL クエリを実行する場合は、CloudTrail Lake の使用を検討してください。CloudTrail Lakeは、企業からの情報を単一の検索可能なイベントデータストアに集約するトレイルを作成する AWS の代替です。Amazon S3 バケットストレージを使用する代わりに、イベントはデータレイクに保存されるため、より豊富で高速なクエリが可能になります。これを使用して、組織、リージョン間、およびカスタム時間範囲内でイベントを検索する SQL クエリを作成できます。CloudTrail Lake クエリは CloudTrail コンソール自体で実行するため、CloudTrail Lake を使用すると Athena は必要ありません。詳細については、CloudTrail Lake のドキュメントを参照してください。

Athena での CloudTrail ログ の使用は、AWS のサービス のアクティビティに関する分析を強化する手段として優れています。たとえば、クエリを使用して傾向を識別したり、アクティビティを属性 (ソース IP アドレスやユーザーなど) でさらに分離したりできます。

一般的な用途は、セキュリティとコンプライアンスのための運用上のアクティビティの分析に CloudTrail ログを使用することです。詳細な例については、「AWS ビッグデータブログ」の記事「AWS CloudTrail と Amazon Athena を使用してセキュリティ、コンプライアンス、運用上のアクティビティを分析する方法」を参照してください。

Athena は、ログファイルの `LOCATION` を指定して、Amazon S3 からこれらのログファイルを直接クエリするために使用できます。以下の 2 つの方法のいずれかを実行できます。

- CloudTrail コンソールから直接 CloudTrailログファイルのテーブルを作成する。
- Athena コンソールで CloudTrail ログファイルのテーブルを手動で作成する。

トピック

- パーティション射影を使用した Athena での CloudTrail ログ用テーブルの作成
- ネストされたフィールドのクエリ
- クエリの例
- CloudTrail ログのクエリに関するヒント

CloudTrail ログと Athena テーブルについて

テーブルの作成を開始する前に、CloudTrail、そしてそれがどのようにデータを保存するかについての理解を深めておく必要があります。これは、テーブルを CloudTrail コンソールから作成するか Athena から作成するかにかかわらず、必要なテーブルを作成するために役立ちます。

CloudTrail は、ログを JSON テキストファイルとして、圧縮された gzip 形式 (*.json.gzip) に保存します。ログファイルの保存先は、証跡のセットアップ方法、AWS リージョン あるいはログを記録する対象のリージョン、その他の要因によって異なります。

ログの保存先、JSON 構造、およびレコードファイルの内容の詳細については、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。

- CloudTrail ログファイルの検索
- CloudTrail ログファイルの例
- CloudTrail レコードの内容
- CloudTrail イベントリファレンス

ログを収集して Amazon S3 に保存し、AWS Management Console で CloudTrail を有効にします。詳細については、「AWS CloudTrail ユーザーガイド」の「証跡の作成」を参照してください。

ログの保存先の Amazon S3 バケットをメモしておきます。LOCATION 句を、CloudTrail ログの場所と使用するオブジェクトのセットへのパスに置き換えます。例では、特定のアカウントに関するログの LOCATION 値を使用していますが、目的に応じた保存先を指定できます。

以下に例を示します。

- 複数のアカウントのデータを分析するには、LOCATION を元の設定に戻し、AWSLogs を使用してすべての LOCATION `'s3://DOC-EXAMPLE-BUCKET/AWSLogs/'` を分析対称にします。
- 特定の日付、アカウント、リージョンのデータを分析するには、`LOCATION 's3://DOC-EXAMPLE-BUCKET/123456789012/CloudTrail/us-east-1/2016/03/14/'.` を使用します。

オブジェクト階層の最上位レベルを使用することで、Athena を使用したクエリの実行時に最大の柔軟性が提供されます。

CloudTrail ログ用の Athena テーブルを作成するための CloudTrail コンソールの使用

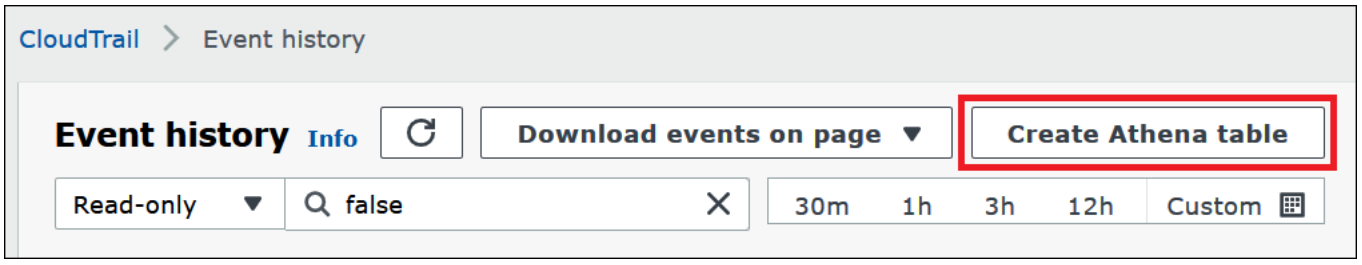
CloudTrail コンソールから直接 CloudTrail ログをクエリするために、パーティションされていない Athena テーブルを作成できます。CloudTrail コンソールからの Athena テーブルの作成には、Athena でテーブルを作成するために十分なアクセス許可を持つロールでログインする必要があります。

> **Note**
>
> CloudTrail コンソールを使用して、組織の証跡ログ用の Athena テーブルを作成することはできません。代わりに、Athena コンソールを使用してテーブルを手動で作成し、正しいストレージ場所を指定できるようにします。組織の証跡については、AWS CloudTrail ユーザーガイドの「組織の証跡の作成」を参照してください。

- Athena のためのアクセス許可のセットアップについては、「セットアップ」を参照してください。
- パーティションを含むテーブルの作成については、「Athena で手動パーティショニングを使用して CloudTrail ログ用のテーブルを作成する」を参照してください。

CloudTrail コンソールを使用して CloudTrail 証跡用の Athena テーブルを作成する

1. CloudTrail コンソール (https://console.aws.amazon.com/cloudtrail/) を開きます。
2. ナビゲーションペインで [Event history (イベント履歴)] を選択します。
3. [Athena テーブルを作成] をクリックします。



4. [Storage location] (ストレージの場所) の下矢印を使用して、クエリする証跡のためにログファイルが保存される Amazon S3 バケットを選択します。

> **Note**
>
> 証跡に関連付けられているバケットの名前を探すには、CloudTrail のナビゲーションペインで [Trails] (証跡) を選択して、証跡の [S3 bucket] (S3 バケット) 列を表示します。バケットの Amazon S3 の場所を確認するには、[S3 bucket] (S3 バケット) 列でバケットのリンクを選択します。これにより、Amazon S3 コンソールで CloudTrail バケットの場所が開きます。

5. [Create table (テーブルの作成)] を選択します。Amazon S3 バケットの名前が含まれたデフォルトの名前でテーブルが作成されます。

Athena で手動パーティショニングを使用して CloudTrail ログ用のテーブルを作成する

Athena コンソールで、CloudTrail ログファイル用のテーブルを手動で作成してから、Athena でクエリを実行することができます。

Athena コンソールを使用して CloudTrail 証跡のための Athena テーブルを作成する

1. 以下の DDL ステートメントを Athena コンソールクエリエディタにコピーして貼り付けます。

```
CREATE EXTERNAL TABLE cloudtrail_logs (
eventversion STRING,
useridentity STRUCT<
               type:STRING,
               principalid:STRING,
               arn:STRING,
               accountid:STRING,
               invokedby:STRING,
               accesskeyid:STRING,
               userName:STRING,
  sessioncontext:STRUCT<
    attributes:STRUCT<
               mfaauthenticated:STRING,
               creationdate:STRING>,
    sessionissuer:STRUCT<
               type:STRING,
               principalId:STRING,
               arn:STRING,
               accountId:STRING,
               userName:STRING>,
```

```
    ec2RoleDelivery:string,
    webIdFederationData: STRUCT<
                federatedProvider: STRING,
                attributes: map<string,string>
    >
  >
>,
eventtime STRING,
eventsource STRING,
eventname STRING,
awsregion STRING,
sourceipaddress STRING,
useragent STRING,
errorcode STRING,
errormessage STRING,
requestparameters STRING,
responseelements STRING,
additionaleventdata STRING,
requestid STRING,
eventid STRING,
resources ARRAY<STRUCT<
                arn:STRING,
                accountid:STRING,
                type:STRING>>,
eventtype STRING,
apiversion STRING,
readonly STRING,
recipientaccountid STRING,
serviceeventdetails STRING,
sharedeventid STRING,
vpcendpointid STRING,
eventCategory STRING,
tlsDetails struct<
  tlsVersion:string,
  cipherSuite:string,
  clientProvidedHostHeader:string>
)
PARTITIONED BY (region string, year string, month string, day string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/Account_ID/CloudTrail/';
```

> **Note**
>
> 例に示されている org.apache.hive.hcatalog.data.JsonSerDe の使用をお勧めします。com.amazon.emr.hive.serde.CloudTrailSerde は存在するものの、現在新しい CloudTrail フィールドの一部を処理しません。

2. (オプション) テーブルに必要のないフィールドをすべて削除します。特定の列のセットのみを読み込む必要がある場合は、テーブル定義で他の列を除外できます。
3. ログデータが含まれる保存先の Amazon S3 バケットをポイントするように s3://DOC-EXAMPLE-BUCKET/AWSLogs/*Account_ID*/CloudTrail/ を変更します。
4. フィールドが正しく表示されていることを確認します。CloudTrail レコード内のフィールドの完全なリストに関する詳細については、「CloudTrail レコードの内容」を参照してください。

   次の例では Hive JSON SerDe を使用しています。この例では、フィールド requestparameters、responseelements、および additionaleventdata は、クエリでタイプ STRING としてリストされていますが、JSON で使用される STRUCT データ型です。そのため、これらのフィールドからデータを取得するには、JSON_EXTRACT 関数を使用します。詳細については、「the section called “JSON からのデータの抽出”」を参照してください。パフォーマンスを向上させるために、この例では、AWS リージョン、年、月、日に基づいてデータをパーティションしています。
5. Athena コンソールで CREATE TABLE ステートメントを実行します。
6. 以下の例のように、ALTER TABLE ADD PARTITION コマンドを使用してパーティションをロードすることで、データをクエリできるようにします。

```
ALTER TABLE table_name ADD
   PARTITION (region='us-east-1',
              year='2019',
              month='02',
              day='01')
   LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/Account_ID/CloudTrail/us-
east-1/2019/02/01/'
```

手動パーティショニングを使用して組織全体の証跡用のテーブルを作成する

Athena で組織全体の CloudTrail ログファイル用のテーブルを作成するには、「Athena で手動パーティショニングを使用して CloudTrail ログ用のテーブルを作成する」(Athena で手動パーティショニ

ングを使用して CloudTrail ログ用のテーブルを作成する) の手順に従います。ただし、次の手順に記載されている変更を加えてください。

組織全体の CloudTrail ログ用の Athena テーブルを作成するには

1. 次の例のように、CREATE TABLE ステートメントで、組織 ID が含まれるように LOCATION 句を変更します。

```
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/Account_ID/CloudTrail/'
```

2. 次の例のように、PARTITIONED BY 句で、アカウント ID のエントリを文字列として追加します。

```
PARTITIONED BY (account string, region string, year string, month string, day
 string)
```

次の例は、結合された結果を示しています。

```
...

PARTITIONED BY (account string, region string, year string, month string, day
 string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/Account_ID/CloudTrail/'
```

3. 次の例のように、ALTER TABLE ステートメントの ADD PARTITION 句には、アカウント ID を含めます。

```
ALTER TABLE table_name ADD
PARTITION (account='111122223333',
region='us-east-1',
year='2022',
month='08',
day='08')
```

4. 次の例のように、ALTER TABLE ステートメントの LOCATION 句には、追加する組織 ID、アカウント ID、パーティションを含めます。

```
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/Account_ID/CloudTrail/us-
east-1/2022/08/08/'
```

次の例では、ALTER TABLE ステートメントは結合された結果を示します。

```
ALTER TABLE table_name ADD
PARTITION (account='111122223333',
region='us-east-1',
year='2022',
month='08',
day='08')
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/111122223333/CloudTrail/
us-east-1/2022/08/08/'
```

パーティション射影を使用した Athena での CloudTrail ログ用テーブルの作成

CloudTrail ログには事前に指定できるパーティションスキームを持つ既知の構造があるため、Athena のパーティション射影機能を使用することで、クエリランタイムを短縮し、パーティション管理を自動化できます。新しいデータが追加されると、パーティション射影は新しいパーティションを自動で追加します。このため、ALTER TABLE ADD PARTITION を使用してパーティションを手動で追加する必要がなくなります。

次の CREATE TABLE ステートメント例では、指定した日付から現在までの単一の AWS リージョン リージョンの CloudTrail ログで、パーティション射影を自動的に使用しています。LOCATION 句と storage.location.template 句では、*bucket*、*account-id*、および *aws-region* の各プレースホルダーを、対応する同等の値に置き換えます。projection.timestamp.range では、*2020*/*01*/*01* を使用を開始する日に置き換えます。クエリが正常に実行されると、テーブルをクエリできます。パーティションをロードするのに、ALTER TABLE ADD PARTITION を実行する必要はありません。

```
CREATE EXTERNAL TABLE cloudtrail_logs_pp(
    eventVersion STRING,
    userIdentity STRUCT<
        type: STRING,
        principalId: STRING,
        arn: STRING,
        accountId: STRING,
        invokedBy: STRING,
        accessKeyId: STRING,
```

```
        userName: STRING,
        sessionContext: STRUCT<
            attributes: STRUCT<
                mfaAuthenticated: STRING,
                creationDate: STRING>,
            sessionIssuer: STRUCT<
                type: STRING,
                principalId: STRING,
                arn: STRING,
                accountId: STRING,
                userName: STRING>,
            ec2RoleDelivery:string,
            webIdFederationData: STRUCT<
                federatedProvider: STRING,
                attributes: map<string,string>
            >
        >
    >,
    eventTime STRING,
    eventSource STRING,
    eventName STRING,
    awsRegion STRING,
    sourceIpAddress STRING,
    userAgent STRING,
    errorCode STRING,
    errorMessage STRING,
    requestparameters STRING,
    responseelements STRING,
    additionaleventdata STRING,
    requestId STRING,
    eventId STRING,
    readOnly STRING,
    resources ARRAY<STRUCT<
        arn: STRING,
        accountId: STRING,
        type: STRING>>,
    eventType STRING,
    apiVersion STRING,
    recipientAccountId STRING,
    serviceEventDetails STRING,
    sharedEventID STRING,
    vpcendpointid STRING,
    eventCategory STRING,
    tlsDetails struct<
```

```
        tlsVersion:string,
        cipherSuite:string,
        clientProvidedHostHeader:string>
  )
PARTITIONED BY (
   `timestamp` string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/CloudTrail/aws-region'
TBLPROPERTIES (
  'projection.enabled'='true',
  'projection.timestamp.format'='yyyy/MM/dd',
  'projection.timestamp.interval'='1',
  'projection.timestamp.interval.unit'='DAYS',
  'projection.timestamp.range'='2020/01/01,NOW',
  'projection.timestamp.type'='date',
  'storage.location.template'='s3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/
CloudTrail/aws-region/${timestamp}')
```

パーティション射影の詳細については、「Amazon Athena でのパーティション射影」を参照してください。

ネストされたフィールドのクエリ

userIdentity および resources フィールドはネストされたデータ型であるため、それらのクエリには特別な処理が必要です。

userIdentity オブジェクトは、ネストされた STRUCT 型で構成されています。以下の例にあるように、これらは、フィールドを区切るためにドットを使用してクエリすることができます。

```
SELECT
    eventsource,
    eventname,
    useridentity.sessioncontext.attributes.creationdate,
    useridentity.sessioncontext.sessionissuer.arn
FROM cloudtrail_logs
WHERE useridentity.sessioncontext.sessionissuer.arn IS NOT NULL
ORDER BY eventsource, eventname
LIMIT 10
```

resources フィールドは STRUCT オブジェクトの配列です。これらの配列では、CROSS JOIN UNNEST を使用して配列のネストを解除し、そのオブジェクトをクエリできるようにします。

以下の例は、リソース ARN が example/datafile.txt で終わるすべての行を返します。読みやすさのために、replace 関数が ARN から初期の arn:aws:s3::: サブストリングを削除します。

```
SELECT
    awsregion,
    replace(unnested.resources_entry.ARN,'arn:aws:s3:::') as s3_resource,
    eventname,
    eventtime,
    useragent
FROM cloudtrail_logs t
CROSS JOIN UNNEST(t.resources) unnested (resources_entry)
WHERE unnested.resources_entry.ARN LIKE '%example/datafile.txt'
ORDER BY eventtime
```

以下の例は、DeleteBucket のイベントをクエリします。このクエリは、バケットの名前とバケットが属するアカウント ID を resources オブジェクトから抽出します。

```
SELECT
    awsregion,
    replace(unnested.resources_entry.ARN,'arn:aws:s3:::') as deleted_bucket,
    eventtime AS time_deleted,
    useridentity.username,
    unnested.resources_entry.accountid as bucket_acct_id
FROM cloudtrail_logs t
CROSS JOIN UNNEST(t.resources) unnested (resources_entry)
WHERE eventname = 'DeleteBucket'
ORDER BY eventtime
```

ネスト解除の詳細については、「配列のフィルタ処理」を参照してください。

クエリの例

以下の例は、CloudTrail イベントログ向けに作成されたテーブルから、匿名化された (署名されていない) すべてのリクエストを返すクエリの一部を示しています。このクエリは、useridentity.accountid が匿名で useridentity.arn が指定されていないリクエストを選択します。

```
SELECT *
```

```
FROM cloudtrail_logs
WHERE
    eventsource = 's3.amazonaws.com' AND
    eventname in ('GetObject') AND
    useridentity.accountid = 'anonymous' AND
    useridentity.arn IS NULL AND
    requestparameters LIKE '%[your bucket name ]%';
```

詳細については、「AWS Big·Data·Blog」(ビッグデータブログ) の記事「AWS CloudTrail と Amazon Athena を使用して、セキュリティ、コンプライアンス、運用上のアクティビティを分析する」を参照してください。

CloudTrail ログのクエリに関するヒント

CloudTrail ログデータを探索するには、以下のヒントを参考にしてください。

- ログをクエリする前に、ログテーブルが「the section called “Athena で手動パーティショニングを使用して CloudTrail ログ用のテーブルを作成する”」に示している内容と同じになっていることを確認します。これが最初のテーブルではない場合は、コマンド `DROP TABLE cloudtrail_logs` を使用して既存のテーブルを削除します。
- 既存のテーブルを削除した後、再作成します。詳細については、「Athena で手動パーティショニングを使用して CloudTrail ログ用のテーブルを作成する」を参照してください。

  Athena クエリのフィールドが正しく表示されていることを確認します。CloudTrail レコード内のフィールドの完全なリストについては、「CloudTrail レコードの内容」を参照してください。

  クエリ内のフィールドが JSON 形式 (STRUCT など) である場合は、JSON からデータを抽出します。詳細については、「JSON からのデータの抽出」を参照してください。

  CloudTrail テーブルに対してクエリを発行するためのいくつかの提案事項:

- 最初に、どの ユーザーが、どの送信元 IP アドレスから API オペレーションを呼び出したかを確認します。
- 次の基本的な SQL クエリをテンプレートとして使用します。このクエリを Athena コンソールに貼り付けて実行します。

```
SELECT
 useridentity.arn,
 eventname,
 sourceipaddress,
 eventtime
```

```
FROM cloudtrail_logs
LIMIT 100;
```

- クエリを修正して、データをさらに詳しく調べます。
- パフォーマンスを強化するには、LIMIT 句を追加して、指定したサブセットの行のみが返るようにします。

Amazon EMR ログのクエリ

Amazon EMR と、Amazon EMR で実行されるビッグデータアプリケーションは、ログファイルを生成します。ログファイルはマスターノードに書き込まれます。ログファイルを自動的に Amazon S3 にアーカイブするように Amazon EMR を設定することもできます。Amazon Athena を使用してこれらのログをクエリし、アプリケーションとクラスターのイベントと傾向を特定することができます。Amazon EMR のログファイルのタイプと、それらの Amazon S3 への保存に関する詳細については、「Amazon EMR マネジメントガイド」の「ログファイルを表示する」を参照してください。

Amazon EMR ログファイルに基づく基本的なテーブルの作成とクエリ

以下の例は、s3://aws-logs-*123456789012*-*us-west-2*/elasticmapreduce/*j-2ABCDE34F5GH6*/elasticmapreduce/ に保存されたログファイルに基づいて、基本的なテーブル myemrlogs を作成します。以下の例で使用されている Amazon S3 の場所は、リージョン *us-west-2* で Amazon Web Services アカウント *123456789012* によって作成された EMR クラスターのデフォルトでのログの場所のパターンを反映しています。カスタムロケーションを使用する場合、パターンは s3://DOC-EXAMPLE-BUCKET/*ClusterID* になります。

パーティション化されたテーブルを作成してクエリのパフォーマンスを向上させ、データ転送を削減できる可能性のある方法については、「Amazon EMR ログに基づくパーティションテーブルの作成とクエリ」を参照してください。

```
CREATE EXTERNAL TABLE `myemrlogs`(
  `data` string COMMENT 'from deserializer')
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
```

```
  's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6'
```

次の例のクエリは、前の例で作成した myemrlogs テーブルに対して実行できます。

Example – ERROR、WARN、INFO、EXCEPTION、FATAL、または DEBUG の発生に関するステップログのクエリ

```
SELECT data,
        "$PATH"
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'s-86URH188Z6B1')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – ERROR、WARN、INFO、EXCEPTION、FATAL、または DEBUG に関する特定のインスタンスログ i-00b3c0a839ece0a9c のクエリ

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'i-00b3c0a839ece0a9c')
        AND regexp_like("$PATH",'state')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – ERROR、WARN、INFO、EXCEPTION、FATAL、または DEBUG に関する Presto アプリケーションログのクエリ

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'presto')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – ERROR、WARN、INFO、EXCEPTION、FATAL、または DEBUG に関する Namenode アプリケーションログのクエリ

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'namenode')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example – ERROR、WARN、INFO、EXCEPTION、FATAL、または DEBUG に関する日付と時間でのすべてのログのクエリ

```
SELECT distinct("$PATH") AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'2019-07-23-10')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Amazon EMR ログに基づくパーティションテーブルの作成とクエリ

これらの例は、Athena テーブルを作成するためのものと同じログの場所を使用しますが、テーブルがパーティション分割された後、パーティションがログの場所ごとに作成されます。詳細については、「Athena でのデータのパーティション化」を参照してください。

次のクエリは、mypartitionedemrlogs という名前のパーティション化されたテーブルを作成します。

```
CREATE EXTERNAL TABLE `mypartitionedemrlogs`(
  `data` string COMMENT 'from deserializer')
 partitioned by (logtype string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6'
```

以下のクエリステートメントは、次に Amazon S3 で Amazon EMR が作成する異なるログタイプのサブディレクトリに基づいて、テーブルパーティションを作成します。

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='containers')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
containers/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='hadoop-mapreduce')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
hadoop-mapreduce/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='hadoop-state-pusher')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
hadoop-state-pusher/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='node')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
node/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='steps')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
steps/'
```

パーティションを作成したら、テーブルで SHOW PARTITIONS クエリを実行して、以下を確認します。

```
SHOW PARTITIONS mypartitionedemrlogs;
```

以下の例は、上記の例で作成されたテーブルとパーティションを使用する特定のログエントリに対するクエリを示しています。

Example – ERROR または WARN に関するコンテナパーティション内のアプリケーション application_1561661818238_0002 ログのクエリ

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='containers'
        AND regexp_like("$PATH",'application_1561661818238_0002')
        AND regexp_like(data, 'ERROR|WARN') limit 100;
```

Example – ジョブ job_1561661818238_0004 と Failed Reduces に関する Hadoop-Mapreduce パーティションのクエリ

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='hadoop-mapreduce'
```

```
        AND regexp_like(data,'job_1561661818238_0004|Failed Reduces') limit 100;
```

Example – クエリ ID 056e0609-33e1-4611-956c-7a31b42d2663 に関するノードパーティションでの Hive ログのクエリ

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='node'
        AND regexp_like("$PATH",'hive')
        AND regexp_like(data,'056e0609-33e1-4611-956c-7a31b42d2663') limit 100;
```

Example – アプリケーション 1567660019320_0001_01_000001 に関するノードパーティションでの Resourcemanager ログのクエリ

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='node'
        AND regexp_like(data,'resourcemanager')
        AND regexp_like(data,'1567660019320_0001_01_000001') limit 100
```

AWS Global Accelerator フローログのクエリ

AWS Global Accelerator を使用して、ネットワークトラフィックを AWS グローバルネットワーク経由で最適なエンドポイントに転送するアクセラレーターを作成できます。Global Accelerator の詳細については、「AWS Global Accelerator とは?」を参照してください。

Global Accelerator フローログは、アクセラレーター内のネットワークインターフェイス間で送受信される IP アドレストラフィックに関する情報の取得を可能にします。フローログデータは Amazon S3 にパブリッシュされ、そこでデータを取得して表示できます。詳細については、「AWS Global Accelerator のフローログ」を参照してください。

Athena を使用して、Amazon S3 内の Global Accelerator フローログの場所を指定するテーブルを作成することによって、フローログをクエリすることができます。

Global Accelerator フローログのテーブルを作成する

1. 以下の DDL ステートメントをコピーして Athena コンソール内に貼り付けます。このクエリでは、ROW FORMAT DELIMITED を指定し、SerDe の指定は省略します。この場合、クエリでは `LazySimpleSerDe` が使用されます。このクエリでは、フィールドはスペースで終了します。

```
CREATE EXTERNAL TABLE IF NOT EXISTS aga_flow_logs (
  version string,
  account string,
  acceleratorid string,
  clientip string,
  clientport int,
  gip string,
  gipport int,
  endpointip string,
  endpointport int,
  protocol string,
  ipaddresstype string,
  numpackets bigint,
  numbytes int,
  starttime int,
  endtime int,
  action string,
  logstatus string,
  agasourceip string,
  agasourceport int,
  endpointregion string,
  agaregion string,
  direction string
)
PARTITIONED BY (dt string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/account_id/globalaccelerator/
region/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

2. ログデータが含まれる Amazon S3 バケットをポイントするように LOCATION 値を変更します。

```
's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/account_id/globalaccelerator/region_code/'
```

3. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が aga_flow_logs テーブルを登録し、そのデータをクエリに使用できるようにします。
4. 次のサンプルクエリのように、パーティションを作成してデータを読み取ります。このクエリは、指定日の 1 つのパーティションを作成します。日付と場所のプレースホルダーを置き換えます。

```
ALTER TABLE aga_flow_logs
ADD PARTITION (dt='YYYY-MM-dd')
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/account_id/
globalaccelerator/region_code/YYYY/MM/dd';
```

AWS Global Accelerator フローログのクエリ例

Example – 特定のエッジロケーションを通過するリクエストをリストする

以下のクエリ例は、LHR エッジロケーションを通過するリクエストをリストします。LIMIT 演算子を使用すると、一度にクエリするログの数を制限できます。

```
SELECT
  clientip,
  agaregion,
  protocol,
  action
FROM
  aga_flow_logs
WHERE
  agaregion LIKE 'LHR%'
LIMIT
  100;
```

Example – HTTPS リクエストの大部分の受信するエンドポイント IP アドレスをリストする

最大数の HTTPS リクエストを受信しているエンドポイント IP アドレスを確認するには、次のクエリを使用します。このクエリは、HTTPS ポート 443 で受信したパケット数をカウントし、送信先 IP アドレス別にグループ分けして、上位 10 の IP アドレスを返します。

```
SELECT
  SUM(numpackets) AS packetcount,
  endpointip
FROM
  aga_flow_logs
WHERE
  endpointport = 443
GROUP BY
  endpointip
ORDER BY
```

```
  packetcount DESC
LIMIT
  10;
```

Amazon GuardDuty の結果のクエリ

Amazon GuardDuty は、AWS 環境内での不正または悪質である可能性がある予期しないアクティビティを識別するために役立つ、セキュリティをモニタリングするサービスです。悪意のある可能性がある予期しないアクティビティが検出されると、GuardDuty がストレージと分析のために Amazon S3 にエクスポートできるセキュリティ結果を生成します。結果を Amazon S3 にエクスポートしたら、Athena を使用してそれらをクエリできます。この記事は、GuardDutyの 結果用のテーブルを Athena で作成し、それらをクエリする方法を説明します。

Amazon GuardDuty の詳細については、「Amazon GuardDuty ユーザーガイド」を参照してください。

前提条件

- Amazon S3 に結果をエクスポートするための GuardDuty 機能を有効にします。ステップについては、「Amazon GuardDuty ユーザーガイド」の「結果のエクスポート」を参照してください。

GuardDuty の 結果のための Athena でのテーブルの作成

Athena から GuardDuty の結果をクエリするには、結果用のテーブルを作成する必要があります。

GuardDuty の結果のために Athena でテーブルをで作成する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. 以下の DDL ステートメントを Athena コンソール内に貼り付けます。`LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/GuardDuty/'` の値を、Amazon S3 にある GuardDuty の結果にポイントするように変更します。

```
CREATE EXTERNAL TABLE `gd_logs` (
  `schemaversion` string,
  `accountid` string,
  `region` string,
  `partition` string,
  `id` string,
  `arn` string,
  `type` string,
```

```
  `resource` string,
  `service` string,
  `severity` string,
  `createdat` string,
  `updatedat` string,
  `title` string,
  `description` string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/GuardDuty/'
 TBLPROPERTIES ('has_encrypted_data'='true')
```

> **Note**
>
> SerDe では、各 JSON ドキュメントが、レコード内のフィールドを区切る行終端文字なしの、1 行のテキストに収まっていることを想定しています。JSON テキストがプリティプリント形式の場合、テーブルを作成した後にクエリを実行しようとすると、以下のようなエラーメッセージが表示される場合があります。「HIVE_CURSOR_ERROR: Row is not a valid JSON Object」、または「HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT」。詳細については、GitHub の OpenX SerDe のドキュメントで「JSON Data Files」(JSON データファイル) を参照してください。

3. Athena コンソールでクエリを実行して、gd_logs テーブルを登録します。クエリが完了すると、結果は Athena からクエリを実行できる状態になります。

クエリの例

以下の例は、Athena から GuardDuty の 結果をクエリする方法を示しています。

Example – DNS データの流出

以下のクエリは、DNS クエリを介してデータを流出している可能性がある Amazon EC2 インスタンスに関する情報を返します。

```
SELECT
    title,
    severity,
    type,
    id AS FindingID,
    accountid,
```

```
    region,
    createdat,
    updatedat,
    json_extract_scalar(service, '$.count') AS Count,
    json_extract_scalar(resource, '$.instancedetails.instanceid') AS InstanceID,
    json_extract_scalar(service, '$.action.actiontype') AS DNS_ActionType,
    json_extract_scalar(service, '$.action.dnsrequestaction.domain') AS DomainName,
    json_extract_scalar(service, '$.action.dnsrequestaction.protocol') AS protocol,
    json_extract_scalar(service, '$.action.dnsrequestaction.blocked') AS blocked
FROM gd_logs
WHERE type = 'Trojan:EC2/DNSDataExfiltration'
ORDER BY severity DESC
```

Example – 不正な IAM ユーザーアクセス

以下のクエリは、すべてのリージョンからの IAM プリンシパルに関するすべての `UnauthorizedAccess:IAMUser` 結果タイプを返します。

```
SELECT title,
         severity,
         type,
         id,
         accountid,
         region,
         createdat,
         updatedat,
         json_extract_scalar(service, '$.count') AS Count,
         json_extract_scalar(resource, '$.accesskeydetails.username') AS IAMPrincipal,
         json_extract_scalar(service,'$.action.awsapicallaction.api') AS
 APIActionCalled
FROM gd_logs
WHERE type LIKE '%UnauthorizedAccess:IAMUser%'
ORDER BY severity desc;
```

GuardDuty の結果のクエリに関するヒント

クエリを作成するときは、次の点に注意してください。

- ネストされた JSON フィールドからデータを抽出するには、Presto `json_extract` 関数、または `json_extract_scalar` 関数を使用します。詳細については、「JSON からのデータの抽出」(JSON からデータを抽出する) を参照してください。
- JSON フィールドのすべての文字が小文字であることを確認します。

- クエリ結果のダウンロードについては、「Athena コンソールを使用したクエリ結果ファイルのダウンロード」を参照してください。

AWS Network Firewall ログのクエリ

AWS Network Firewall は、Amazon Virtual Private Cloud インスタンスに不可欠なネットワーク保護をデプロイするために使用できるマネージドサービスです。AWS Network Firewall は AWS Firewall Manager と連携するため、AWS Network Firewall ルールに基づいてポリシーを構築して、VPC とアカウント全体に集中的に適用できます。の詳細については、「AWS Network Firewall」を参照してください。。AWS Network Firewall

ファイアウォールのステートフルルールエンジンに転送するトラフィックの AWS Network Firewall ログの記録を設定できます。ログを記録すると、ステートフルエンジンがパケットを受信した時間、パケットに関する詳細情報、パケットに対して実行されたステートフルルールアクションなど、ネットワークトラフィックに関する詳細情報が得られます。ログは、設定したログ送信先に公開され、そこでログを取得して表示できます。詳細については、「AWS Network Firewall デベロッパーガイド」の「AWS Network Firewallからのネットワークトラフィックのログ記録」を参照してください。

アラートログのテーブルを作成する

1. 次のサンプル DDL ステートメントをアラートログの構造に合わせて変更します。ステートメントを更新して、最新バージョンのログの列を含めることが必要になる場合があります。詳細については、「AWS Network Firewall デベロッパーガイド」の「ファイアウォールログの内容」を参照してください。

```
CREATE EXTERNAL TABLE network_firewall_alert_logs (
  firewall_name string,
  availability_zone string,
  event_timestamp string,
  event struct<
    timestamp:string,
    flow_id:bigint,
    event_type:string,
    src_ip:string,
    src_port:int,
    dest_ip:string,
    dest_port:int,
    proto:string,
    app_proto:string,
    tls_inspected:boolean,
```

```
    alert:struct<
      alert_id:string,
      alert_type:string,
      action:string,
      signature_id:int,
      rev:int,
      signature:string,
      category:string,
      severity:int,
      rule_name:string,
      alert_name:string,
      alert_severity:string,
      alert_description:string,
      file_name:string,
      file_hash:string,
      packet_capture:string,
      reference_links:array<string>
    >,
    src_country:string,
    dest_country:string,
    src_hostname:string,
    dest_hostname:string,
    user_agent:string,
    url:string
   >
)
 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 LOCATION 's3://DOC-EXAMPLE-BUCKET/path_to_alert_logs_folder/';
```

2. LOCATION 句を変更して Amazon S3 のログ用のフォルダを指定します。
3. Athena クエリエディタで CREATE TABLE クエリを実行します。クエリが完了すると、Athena は network_firewall_alert_logs テーブルを登録し、テーブルがポイントするデータをクエリできる状態にします。

サンプルアラートログクエリ

このセクションのサンプルアラートログクエリでは、TLS 検査が実行されたイベントのうち、重大度が 2 以上のアラートを含むものがフィルタリングされます。

このクエリでは、エイリアスを使用して、その列が属する struct を示す出力列の見出しを作成します。例えば、event.alert.category フィールドの列見出しは、単なる category ではなく event_alert_category になります。列名をさらにカスタマイズするには、好みに合わせて

エイリアスを変更します。例えば、アンダースコアやその他の区切り文字を使用して struct 名とフィールド名を区切ることができます。

テーブル定義とクエリ結果に含めるフィールドに基づいて、列名と struct 参照を必ず変更してください。

```
SELECT
  firewall_name,
  availability_zone,
  event_timestamp,
  event.timestamp AS event_timestamp,
  event.flow_id AS event_flow_id,
  event.event_type AS event_type,
  event.src_ip AS event_src_ip,
  event.src_port AS event_src_port,
  event.dest_ip AS event_dest_ip,
  event.dest_port AS event_dest_port,
  event.proto AS event_protol,
  event.app_proto AS event_app_proto,
  event.tls_inspected AS event_tls_inspected,
  event.alert.alert_id AS event_alert_alert_id,
  event.alert.alert_type AS event_alert_alert_type,
  event.alert.action AS event_alert_action,
  event.alert.signature_id AS event_alert_signature_id,
  event.alert.rev AS event_alert_rev,
  event.alert.signature AS event_alert_signature,
  event.alert.category AS event_alert_category,
  event.alert.severity AS event_alert_severity,
  event.alert.rule_name AS event_alert_rule_name,
  event.alert.alert_name AS event_alert_alert_name,
  event.alert.alert_severity AS event_alert_alert_severity,
  event.alert.alert_description AS event_alert_alert_description,
  event.alert.file_name AS event_alert_file_name,
  event.alert.file_hash AS event_alert_file_hash,
  event.alert.packet_capture AS event_alert_packet_capture,
  event.alert.reference_links AS event_alert_reference_links,
  event.src_country AS event_src_country,
  event.dest_country AS event_dest_country,
  event.src_hostname AS event_src_hostname,
  event.dest_hostname AS event_dest_hostname,
  event.user_agent AS event_user_agent,
  event.url AS event_url
FROM
```

```
  network_firewall_alert_logs
WHERE
  event.alert.severity >= 2
  AND event.tls_inspected = true
LIMIT 10;
```

NetFlow ログ用のテーブルを作成する

1. 次の DDL ステートメントのサンプルを Netflow ログの構造に合わせて変更します。ステートメントを更新して、最新バージョンのログの列を含めることが必要になる場合があります。詳細については、「AWS Network Firewall デベロッパーガイド」の「ファイアウォールログの内容」を参照してください。

```
CREATE EXTERNAL TABLE network_firewall_netflow_logs (
  firewall_name string,
  availability_zone string,
  event_timestamp string,
  event struct<
    timestamp:string,
    flow_id:bigint,
    event_type:string,
    src_ip:string,
    src_port:int,
    dest_ip:string,
    dest_port:int,
    proto:string,
    app_proto:string,
    netflow:struct<
      pkts:int,
      bytes:bigint,
      start:string,
      `end`:string,
      age:int,
      min_ttl:int,
      max_ttl:int,
      tcp_flags:struct<
        syn:boolean,
        fin:boolean,
        rst:boolean,
        psh:boolean,
        ack:boolean,
        urg:boolean
      >,
```

```
      tls_inspected:boolean
    >
  >
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/path_to_netflow_logs_folder/';
```

2. LOCATION 句を変更して Amazon S3 のログ用のフォルダを指定します。
3. Athena クエリエディタで CREATE TABLE クエリを実行します。クエリが完了すると、Athena は network_firewall_netflow_logs テーブルを登録し、テーブルがポイントするデータをクエリできる状態にします。

サンプル Netflow ログクエリ

このセクションのサンプル Netflow ログクエリでは、TLS 検査が実行されたイベントがフィルタリングされます。

このクエリでは、エイリアスを使用して、その列が属する struct を示す出力列の見出しを作成します。例えば、event.netflow.bytes フィールドの列見出しは、単なる bytes ではなく event_netflow_bytes になります。列名をさらにカスタマイズするには、好みに合わせてエイリアスを変更します。例えば、アンダースコアやその他の区切り文字を使用して struct 名とフィールド名を区切ることができます。

テーブル定義とクエリ結果に含めるフィールドに基づいて、列名と struct 参照を必ず変更してください。

```
SELECT
  event.src_ip AS event_src_ip,
  event.dest_ip AS event_dest_ip,
  event.proto AS event_proto,
  event.app_proto AS event_app_proto,
  event.netflow.pkts AS event_netflow_pkts,
  event.netflow.bytes AS event_netflow_bytes,
  event.netflow.tcp_flags.syn AS event_netflow_tcp_flags_syn,
  event.netflow.tls_inspected AS event_netflow_tls_inspected
FROM network_firewall_netflow_logs
WHERE event.netflow.tls_inspected = true
```

Network Load Balancer のログのクエリ

Athena を使用して、Network Load Balancer からのログを分析および処理します。これらのログは、Network Load Balancer に送信された Transport Layer Security (TLS) リクエストに関する詳細情報を受け取ります。これらのアクセスログを使用して、トラフィックパターンを分析し、問題のトラブルシューティングを行うことができます。

Network Load Balancer のアクセスログを分析する前に、ログを宛先の Amazon S3 バケットに保存するためにログを設定します。詳細および各 Network Load Balancer アクセスログエントリの詳細については、「Network Load Balancer のアクセスログ」を参照してください。

- Network Load Balancer ログのテーブルを作成する
- Network Load Balancer のクエリ例

Network Load Balancer ログのテーブルを作成する

1. 以下の DDL ステートメントをコピーして Athena コンソール内に貼り付けます。Network Load Balancer ログレコードの構文をチェックします。以下のクエリを更新して、最新バージョンのレコードのための列と Regex 構文を含めることが必要になる場合があります。

```
CREATE EXTERNAL TABLE IF NOT EXISTS nlb_tls_logs (
             type string,
             version string,
             time string,
             elb string,
             listener_id string,
             client_ip string,
             client_port int,
             target_ip string,
             target_port int,
             tcp_connection_time_ms double,
             tls_handshake_time_ms double,
             received_bytes bigint,
             sent_bytes bigint,
             incoming_tls_alert int,
             cert_arn string,
             certificate_serial string,
             tls_cipher_suite string,
             tls_protocol_version string,
             tls_named_group string,
             domain_name string,
```

```
            alpn_fe_protocol string,
            alpn_be_protocol string,
            alpn_client_preference_list string,
            tls_connection_creation_time string
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*):([0-9]*)
 ([-.0-9]*) ([-.0-9]*) ([-0-9]*) ([-0-9]*) ([-0-9]*) ([^ ]*) ([^ ]*) ([^ ]*)
 ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)$')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/AWS_account_ID/
elasticloadbalancing/region';
```

2. LOCATION の Simple Storage Service (Amazon S3) バケットを変更して、Network Load Balancer ログの宛先を指定します。
3. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が `nlb_tls_logs` テーブルを登録して、その中のデータをクエリ向けに準備します。

Network Load Balancer のクエリ例

証明書の使用回数を確認するには、次の例のようなクエリを使用します。

```
SELECT count(*) AS
         ct,
         cert_arn
FROM "nlb_tls_logs"
GROUP BY  cert_arn;
```

次のクエリでは、1.3 より前の TLS バージョンを使用しているユーザーの数を表示します。

```
SELECT tls_protocol_version,
         COUNT(tls_protocol_version) AS
         num_connections,
         client_ip
FROM "nlb_tls_logs"
WHERE tls_protocol_version < 'tlsv13'
GROUP BY tls_protocol_version, client_ip;
```

次のクエリを使用して、TLS ハンドシェイクに長い時間がかかっている接続を特定します。

```
SELECT *
FROM "nlb_tls_logs"
ORDER BY  tls_handshake_time_ms DESC
LIMIT 10;
```

次のクエリは、過去 30 日間にネゴシエートされた TLS プロトコルのバージョンと暗号化スイートを特定し、その数をカウントするために使用します。

```
SELECT tls_cipher_suite,
         tls_protocol_version,
         COUNT(*) AS ct
FROM "nlb_tls_logs"
WHERE from_iso8601_timestamp(time) > current_timestamp - interval '30' day
        AND NOT tls_protocol_version = '-'
GROUP BY tls_cipher_suite, tls_protocol_version
ORDER BY ct DESC;
```

Amazon Route 53 Resolver クエリログのクエリ

Amazon Route 53 Resolver クエリログ用の Athena テーブルを作成して、それらを Athena からクエリできます。

Route 53 Resolver のクエリロギングは、VPC 内のリソースによって行われた DNS クエリ、インバウンドリゾルバーエンドポイントを使用するオンプレミスリソース、再帰的な DNS 解決にアウトバウンドリゾルバーエンドポイントを使用するクエリ、およびドメインリストをブロック、許可、またはモニタリングするために Route 53 Resolver DNS ファイアウォールルールを使用するクエリをログ記録するためのものです。Resolver クエリログ記録の詳細については、Amazon Route 53 デベロッパーガイドの「Resolver query logging」を参照してください。ログにある各フィールドについては、「Amazon Route 53 デベロッパーガイド」の「Resolver クエリログに表示される値」を参照してください。

Resolver クエリログ用のテーブルの作成

Athena コンソールのクエリエディタを使用して、Route 53 Resolver クエリログ用のテーブルを作成してクエリできます。

Route 53 Resolver クエリログ用の Athena テーブルを作成してクエリする

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。

2. Athena クエリエディタに以下の CREATE TABLE ステートメントを入力します。LOCATION 句の値を、Simple Storage Service (Amazon S3) の Resolver ログの場所に対応する値に置き換えます。

```
CREATE EXTERNAL TABLE r53_rlogs (
  version string,
  account_id string,
  region string,
  vpc_id string,
  query_timestamp string,
  query_name string,
  query_type string,
  query_class
    string,
  rcode string,
  answers array<
    struct<
      Rdata: string,
      Type: string,
      Class: string>
    >,
  srcaddr string,
  srcport int,
  transport string,
  srcids struct<
    instance: string,
    resolver_endpoint: string
    >,
  firewall_rule_action string,
  firewall_rule_group_id string,
  firewall_domain_list_id string
 )

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/aws_account_id/vpcdnsquerylogs/{vpc-id}/'
```

Resolver クエリログデータは JSON 形式であるため、CREATE TABLE ステートメントは JSON SerDe ライブラリを使用してデータを分析します。

> **Note**
>
> SerDe では、各 JSON ドキュメントが、レコード内のフィールドを区切る行終端文字なしの、1 行のテキストに収まっていることを想定しています。JSON テキストがプリティプリント形式の場合、テーブルを作成した後にクエリを実行しようとすると、以下のようなエラーメッセージが表示される場合があります。「HIVE_CURSOR_ERROR: Row is not a valid JSON Object」、または「HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT」。詳細については、GitHub の OpenX SerDe のドキュメントで「JSON Data Files」(JSON データファイル) を参照してください。

3. [Run query] (クエリの実行) を選択します。ステートメントが `r53_rlogs` という名前の Athena テーブルを作成します。このテーブルの列は Resolver ログデータの各フィールドを表します。
4. Athena コンソールのクエリエディタで以下のクエリを実行して、テーブルが作成されたことを検証します。

```
SELECT * FROM "r53_rlogs" LIMIT 10
```

パーティション化の例

次の例は、パーティション射影を使用し、VPC と日付で分割された Resolver クエリログの `CREATE TABLE` ステートメントを示しています。パーティション射影の詳細については、「Amazon Athena でのパーティション射影」を参照してください。

```
CREATE EXTERNAL TABLE r53_rlogs (
  version string,
  account_id string,
  region string,
  vpc_id string,
  query_timestamp string,
  query_name string,
  query_type string,
  query_class string,
  rcode string,
  answers array<
    struct<
      Rdata: string,
      Type: string,
```

```
      Class: string>
    >,
  srcaddr string,
  srcport int,
  transport string,
  srcids struct<
    instance: string,
    resolver_endpoint: string
    >,
  firewall_rule_action string,
  firewall_rule_group_id string,
  firewall_domain_list_id string
)
PARTITIONED BY (
`date` string,
`vpc` string
)
ROW FORMAT SERDE      'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT          'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION              's3://DOC-EXAMPLE-BUCKET/route53-query-logging/
AWSLogs/aws_account_id/vpcdnsquerylogs/'
TBLPROPERTIES(
'projection.enabled' = 'true',
'projection.vpc.type' = 'enum',
'projection.vpc.values' = 'vpc-6446ae02',
'projection.date.type' = 'date',
'projection.date.range' = '2023/06/26,NOW',
'projection.date.format' = 'yyyy/MM/dd',
'projection.date.interval' = '1',
'projection.date.interval.unit' = 'DAYS',
'storage.location.template' = 's3://DOC-EXAMPLE-BUCKET/route53-query-logging/
AWSLogs/aws_account_id/vpcdnsquerylogs/${vpc}/${date}/'
)
```

クエリの例

以下の例は、Resolver クエリログで Athena から実行できるクエリをいくつか説明しています。

例 1 – ログを降順の query_timestamp 順にクエリする

以下のクエリは、query_timestamp 順で降順にログ結果を表示します。

```
SELECT * FROM "r53_rlogs"
```

```
ORDER BY query_timestamp DESC
```

例 2 – 指定された開始時刻と終了時刻の範囲内でログをクエリする

以下のクエリは、2020 年 9 月 24 日の午前 0 時から午前 8 時の間のログをクエリします。開始時刻と終了時刻は、独自の要件に従って置き換えます。

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode
FROM "r53_rlogs"
WHERE (parse_datetime(query_timestamp,'yyyy-MM-dd''T''HH:mm:ss''Z')
     BETWEEN parse_datetime('2020-09-24-00:00:00','yyyy-MM-dd-HH:mm:ss')
     AND parse_datetime('2020-09-24-00:08:00','yyyy-MM-dd-HH:mm:ss'))
ORDER BY query_timestamp DESC
```

例 3 – 指定された DNS クエリ名パターンに基づいてログをクエリする

以下のクエリは、クエリ名に文字列「example.com」が含まれたレコードを選択します。

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode, answers
FROM "r53_rlogs"
WHERE query_name LIKE '%example.com%'
ORDER BY query_timestamp DESC
```

例 4 – 応答のないログリクエストをクエリする

以下のクエリは、リクエストが応答を受け取らなかったログエントリを選択します。

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode, answers
FROM "r53_rlogs"
WHERE cardinality(answers) = 0
```

例 5 – 特定の回答が含まれるログをクエリする

以下のクエリは、answer.Rdata 値に指定された IP アドレスがあるログを表示します。

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode,
 answer.Rdata
FROM "r53_rlogs"
CROSS JOIN UNNEST(r53_rlogs.answers) as st(answer)
WHERE answer.Rdata='203.0.113.16';
```

Amazon SES イベントログのクエリ

Amazon Athena を使用して、Amazon Simple Email Service (Amazon SES) イベントログをクエリできます。

Amazon SES は、ユーザー独自の E メールアドレスとドメインを使用して E メールを送受信するための、簡単でコスト効率の高い方法を提供する E メールプラットフォームです。イベント、メトリクス、および統計を使用して、Amazon SES 送信アクティビティをきめ細かいレベルでモニタリングできます。

ユーザー定義の特性に基づいて、Amazon SES イベントを Amazon CloudWatch、Amazon Data Firehose、または Amazon Simple Notification Service に発行できます。Amazon S3 で情報を保存した後、Amazon Athena からクエリを実行できます。

Firehose、Amazon Athena、Amazon QuickSight を使用して Amazon SES の E メールイベントを分析する方法の詳細については、AWS メッセージングおよびターゲティングブログの「Analyzing Amazon SES event data with AWS Analytics Services」を参照してください。

Amazon VPC フローログのクエリ

Amazon Virtual Private Cloud フローログは、VPC 内のネットワークインターフェイス間で送受信される IP トラフィックに関する情報をキャプチャします。このログを使用してネットワークトラフィックのパターンを調査し、VPC ネットワーク全体の脅威やリスクを特定します。

Amazon VPC フローログのクエリを実行するには、2 つの方法があります。

- Amazon VPC コンソール — Amazon VPC コンソールの Athena 統合機能を使用して、パーティションを含む Athena データベース、ワークグループ、およびフローログテーブルを作成する AWS CloudFormation テンプレートを生成します。テンプレートでは、事前定義されたフローログのクエリのセットも作成します。これは、VPC を通過するトラフィックに関するインサイトの取得に使用できます。

  このアプローチの詳細については、「Amazon VPC ユーザーガイド」の「Amazon Athena を使用したフローログのクエリ」を参照してください。
- Amazon Athena コンソール — Athena コンソールでテーブルやクエリを直接作成します。詳細については、引き続きこのページを参照してください。

カスタム VPC フローログのテーブルの作成とクエリ

Athena でログのクエリを開始する前に、VPC フローログを有効化し、それらが Amazon S3 バケットに保存されるように設定します。ログを作成したら、それを数分間実行していくらかのデータを収集します。ログは、Athena で直接クエリできる GZIP 圧縮形式で作成されます。

VPC フローログの作成時に、フローログで返すフィールドおよびフィールドが表示される順序を指定する際は、カスタム形式を使用できます。フローログレコード詳細については、「Amazon VPC ユーザーガイド」の「フローログレコード」を参照してください。

一般的な考慮事項

Athena で Amazon VPC フローログのテーブルを作成する際は、次の点に注意してください。

- デフォルトでは、Parquet は名前で列にアクセスします。詳細については、「スキーマ更新の処理」を参照してください。
- Athena の列名には、フローログレコードの名前を使用します。Athena スキーマの列名は、Amazon VPC フローログのフィールド名と完全に一致する必要があります。ただし、これらには次の違いがあります。
  - Amazon VPC のログフィールド名のハイフンは、Athena の列名ではアンダースコアに置き換えられます。Athena では、データベース名、テーブル名、および列名に、小文字、数字、アンダースコア文字のみを使用できます。詳細については、「データベース、テーブル、および列の名前」を参照してください。
  - バックティックで囲むことで、Athena で予約されたキーワードであるフローログのレコード名をエスケープします。
- VPC フローログは AWS アカウント 固有です。ログファイルを Amazon S3 に発行すると、Amazon VPC が Amazon S3 で作成するパスには、フローログの作成に使用された AWS アカウント の ID が含まれます。詳細については、「Amazon VPC ユーザーガイド」の「フローログを Amazon S3 に発行する」を参照してください。

Amazon VPC フローログの CREATE TABLE ステートメント

次の手順では、Amazon VPC フローログ用の Amazon VPC テーブルを作成します。カスタム形式でフローログを作成する場合は、フローログの作成時に指定したフィールドと一致するフィールドを、それらに指定したものと同じ順序で使用してテーブルを作成します。

Amazon VPC フローログ用の Athena テーブルを作成するには

1. 一般的な考慮事項 セクションのガイドラインに従って、次のような DDL ステートメントを Athena コンソールクエリエディタに入力します。サンプルステートメントは、「フローログレコード」に記載されているように、Amazon VPC フローログのバージョン 2 から 5 の列を持つテーブルを作成します。異なる列のセットまたは列の順序を使用する場合は、必要に応じてステートメントを変更します。

```
CREATE EXTERNAL TABLE IF NOT EXISTS `vpc_flow_logs` (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  region string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (`date` date)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
```

```
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/{account_id}/
vpcflowlogs/{region_code}/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

以下の点に注意してください。

- このクエリは ROW FORMAT DELIMITED を指定し、SerDe の指定が省略されます。これは、クエリが CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe を使用しているということです。このクエリでは、フィールドはスペースで終了します。
- PARTITIONED BY 句は、date 型を使用します。これにより、クエリで数学演算子を使用して、特定の日付より古いものまたは新しいものを選択できます。

> **Note**
>
> date は DDL ステートメントの予約済みキーワードであるため、バックティック文字でエスケープされます。詳細については、「予約済みキーワード」を参照してください。

- 異なるカスタム形式の VPC フローログの場合、フローログの作成時に指定したフィールドと一致するようにフィールドを変更します。

2. ログデータが含まれる Amazon S3 バケットをポイントするように LOCATION 's3://DOC-EXAMPLE-BUCKET/*prefix*/AWSLogs/*{account_id}*/vpcflowlogs/*{region_code}*/' を変更します。
3. Athena コンソールでクエリを実行します。クエリが完了すると、Athena が vpc_flow_logs テーブルを登録し、その中のデータに対してクエリを発行できるように準備します。
4. 次のサンプルクエリのように、パーティションを作成してデータを読み取れるようにします。このクエリは、指定日の 1 つのパーティションを作成します。必要に応じて、日付と場所のプレースホルダーを置き換えます。

> **Note**
>
> このクエリは、指定日に対して単一のパーティションのみを作成します。プロセスを自動化するには、このクエリを実行し、この方法で year/month/day にパーティションを作成するスクリプトを使用するか、パーティション射影を指定する CREATE TABLE ステートメントを使用します。

```
ALTER TABLE vpc_flow_logs
ADD PARTITION (`date`='YYYY-MM-dd')
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/{account_id}/
vpcflowlogs/{region_code}/YYYY/MM/dd';
```

vpc_flow_logs テーブルのクエリ例

Athena コンソールのクエリエディタを使用して、作成したテーブルで SQL ステートメントを実行します。クエリを保存したり、以前のクエリを表示したり、クエリ結果を CSV 形式でダウンロードできます。次の例では、vpc_flow_logs をテーブルの名前に置き換えます。また、要件に応じて列の値やその他の変数を変更します。

次のクエリ例では、指定された日付に対して最大 100 のフローログを一覧表示します。

```
SELECT *
FROM vpc_flow_logs
WHERE date = DATE('2020-05-04')
LIMIT 100;
```

次のクエリは、すべての拒否された TCP 接続を一覧表示します。新しく作成した日付パーティション列 date を使用して、該当イベントが発生した週の曜日を抽出します。

```
SELECT day_of_week(date) AS
  day,
  date,
  interface_id,
  srcaddr,
  action,
  protocol
FROM vpc_flow_logs
WHERE action = 'REJECT' AND protocol = 6
LIMIT 100;
```

最大数の HTTPS リクエストを受信しているサーバーを確認するには、次のクエリを使用します。HTTPS ポート 443 で受信したパケット数をカウントし、送信先 IP アドレス別にグループ分けして、過去 1 週間の上位 10 のサーバーを返します。

```
SELECT SUM(packets) AS
```

```
  packetcount,
  dstaddr
FROM vpc_flow_logs
WHERE dstport = 443 AND date > current_date - interval '7' day
GROUP BY dstaddr
ORDER BY packetcount DESC
LIMIT 10;
```

Apache Parquet 形式でフローログのテーブルを作成する

以下の手順では、Apache Parquet 形式で Amazon VPC フローログ用の Amazon VPC テーブルを作成します。

Parquet 形式で Amazon VPC フローログ用の Athena テーブルを作成するには

1. 一般的な考慮事項 セクションのガイドラインに従って、次のような DDL ステートメントを Athena コンソールクエリエディタに入力します。サンプルステートメントは、「フローログレコード」に記載されているように、1 時間ごとに Hive パーティションされる Parquet 形式で Amazon VPC フローログのバージョン 2 から 5 の列を持つテーブルを作成します。1 時間ごとのパーティションがない場合は、`PARTITIONED BY` 句から `hour` を削除します。

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS vpc_flow_logs_parquet (
     version int,
     account_id string,
     interface_id string,
     srcaddr string,
     dstaddr string,
     srcport int,
     dstport int,
     protocol bigint,
     packets bigint,
     bytes bigint,
     start bigint,
     `end` bigint,
     action string,
     log_status string,
     vpc_id string,
     subnet_id string,
     instance_id string,
     tcp_flags int,
     type string,
     pkt_srcaddr string,
   ```

```
  pkt_dstaddr string,
  region string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (
  `aws-account-id` string,
  `aws-service` string,
  `aws-region` string,
  `year` string,
  `month` string,
  `day` string,
  `hour` string
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/'
TBLPROPERTIES (
  'EXTERNAL'='true',
  'skip.header.line.count'='1'
  )
```

2. ログデータが含まれる Amazon S3 パスをポイントするようにサンプル `LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/'` を変更します。
3. Athena コンソールでクエリを実行します。
4. Hive 互換形式のデータの場合、Athena コンソールで次のコマンドを実行して、メタストアの Hive パーティションを更新およびロードします。クエリが完了すると、vpc_flow_logs_parquet テーブルでデータをクエリできます。

```
MSCK REPAIR TABLE vpc_flow_logs_parquet
```

Hive 互換データを使用していない場合は、ALTER TABLE ADD PARTITION を実行してパーティションをロードします。

Athena を使用して、Parquet 形式の Amazon VPC フローログをクエリする方法については、AWS ビッグデータ ブログの記事「Apache Parquet 形式の VPC フローログを使用して、ネットワーク分析のパフォーマンスを最適化し、そのコストを削減する」を参照してください。

パーティション射影を使用した Amazon VPC フローログのテーブルの作成とクエリ

次のような CREATE TABLE ステートメントを使用してテーブルを作成およびパーティションし、パーティション射影を使用してパーティションを自動的に入力します。サンプルのテーブル名 test_table_vpclogs をそのテーブルの名前に置き換えます。Amazon VPC ログデータが含まれている Amazon S3 バケットを指定するように LOCATION 句を編集します。

次は、非 Hive スタイルのパーティション形式で配信される VPC フローログ用の CREATE TABLE ステートメントです。この例では、マルチアカウント集計が可能です。複数のアカウントからの VPC フローログを 1 つの Amazon S3 バケットに集中管理する場合は、アカウント ID を Amazon S3 パスに入力する必要があります。

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_table_vpclogs (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
```

```
  pkt_dstaddr string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (accid string, region string, day string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION '$LOCATION_OF_LOGS'
TBLPROPERTIES
(
"skip.header.line.count"="1",
"projection.enabled" = "true",
"projection.accid.type" = "enum",
"projection.accid.values" = "$ACCID_1,$ACCID_2",
"projection.region.type" = "enum",
"projection.region.values" = "$REGION_1,$REGION_2,$REGION_3",
"projection.day.type" = "date",
"projection.day.range" = "$START_RANGE,NOW",
"projection.day.format" = "yyyy/MM/dd",
"storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/AWSLogs/${accid}/vpcflowlogs/
${region}/${day}"
)
```

test_table_vpclogs のクエリ例

次のクエリ例では、前述の CREATE TABLE ステートメントで作成された test_table_vpclogs のクエリを実行します。クエリの test_table_vpclogs を独自のテーブルの名前で置き換えます。また、要件に応じて列の値やその他の変数を変更します。

指定された期間の最初の 100 個のアクセスログエントリを時系列順に返すには、次のようなクエリを実行します。

```
SELECT *
FROM test_table_vpclogs
WHERE day >= '2021/02/01' AND day < '2021/02/28'
ORDER BY day ASC
LIMIT 100
```

指定された期間に上位 10 個の HTTP パケットを受信したサーバーを表示するには、次のようなクエリを実行します。このクエリは、HTTPS ポート 443 で受信したパケット数をカウントし、送信先 IP アドレス別にグループ分けして、過去 1 週間の上位 10 エントリを返します。

```
SELECT SUM(packets) AS packetcount,
       dstaddr
FROM test_table_vpclogs
WHERE dstport = 443
  AND day >= '2021/03/01'
  AND day < '2021/03/31'
GROUP BY dstaddr
ORDER BY packetcount DESC
LIMIT 10
```

指定された期間内に作成されたログを返すには、次のようなクエリを実行します。

```
SELECT interface_id,
       srcaddr,
       action,
       protocol,
       to_iso8601(from_unixtime(start)) AS start_time,
       to_iso8601(from_unixtime("end")) AS end_time
FROM test_table_vpclogs
WHERE DAY >= '2021/04/01'
  AND DAY < '2021/04/30'
```

指定された期間の送信元 IP アドレスのアクセスログを返すには、次のようなクエリを実行します。

```
SELECT *
FROM test_table_vpclogs
WHERE srcaddr = '10.117.1.22'
  AND day >= '2021/02/01'
  AND day < '2021/02/28'
```

すべての拒否された TCP 接続を一覧表示するには、次のようなクエリを実行します。

```
SELECT day,
       interface_id,
       srcaddr,
       action,
       protocol
FROM test_table_vpclogs
```

```
WHERE action = 'REJECT' AND protocol = 6 AND day >= '2021/02/01' AND day < '2021/02/28'
LIMIT 10
```

10.117 で始まる IP アドレス範囲のアクセスログを返すには、次のようなクエリを実行します。

```
SELECT *
FROM test_table_vpclogs
WHERE split_part(srcaddr,'.', 1)='10'
  AND split_part(srcaddr,'.', 2) ='117'
```

特定の時間範囲内の送信先 IP アドレスのアクセスログを返すには、次のようなクエリを実行します。

```
SELECT *
FROM test_table_vpclogs
WHERE dstaddr = '10.0.1.14'
  AND day >= '2021/01/01'
  AND day < '2021/01/31'
```

パーティションプロジェクションを使用した Apache Parquet 形式でのフローログのテーブルの作成

次の VPC フローログのパーティションプロジェクション CREATE TABLE ステートメントは Apache Parquet 形式であり、Hive と互換性がなく、曜日ではなく時間と日付でパーティショニングされています。サンプルのテーブル名 test_table_vpclogs_parquet をそのテーブルの名前に置き換えます。Amazon VPC ログデータが含まれている Amazon S3 バケットを指定するように LOCATION 句を編集します。

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_table_vpclogs_parquet (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
```

```
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (region string, date string, hour string)
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/{account_id}/vpcflowlogs/'
TBLPROPERTIES (
"EXTERNAL"="true",
"skip.header.line.count" = "1",
"projection.enabled" = "true",
"projection.region.type" = "enum",
"projection.region.values" = "us-east-1,us-west-2,ap-south-1,eu-west-1",
"projection.date.type" = "date",
"projection.date.range" = "2021/01/01,NOW",
"projection.date.format" = "yyyy/MM/dd",
"projection.hour.type" = "integer",
"projection.hour.range" = "00,23",
"projection.hour.digits" = "2",
"storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/${account_id}/
vpcflowlogs/${region}/${date}/${hour}"
)
```

追加リソース

Athena を使用して VPC フローログを分析する方法の詳細については、次の「AWS ビッグデータ ブログ の投稿」を参照してください。

- ポイントアンドクリック Amazon Athena インテグレーションを使用した VPC フローログの分析
- Amazon Athena および Amazon QuickSight を使用した VPC フローログの分析
- Apache Parquet 形式の VPC フローログでパフォーマンスを最適化し、ネットワーク分析のコストを削減する

AWS WAF ログのクエリ

AWS WAF は、保護されたウェブアプリケーションがクライアントから受信する HTTP および HTTPS リクエストを監視して制御できるようにするウェブアプリケーションファイアウォールです。AWS WAF ウェブアクセスコントロールリスト (ACL) 内のルールを設定することにより、ウェブリクエストの処理方法を定義します。その後、ウェブアプリケーションにウェブ ACL を関連付けて保護します。AWS WAF で保護できるウェブアプリケーションリソースの例には、Amazon CloudFront ディストリビューション、Amazon API Gateway REST API、Application Load Balancers などがあります。AWS WAF の詳細については、「AWS WAF デベロッパーガイド」の「AWS WAF」を参照してください。

AWS WAF ログには、AWS WAF が AWS リソースからリクエストを受信した時間、このリクエストの詳細、各リクエストが適合したルールに対するアクションなど、ウェブ ACL によって分析されたトラフィックに関する情報が含まれます。

AWS WAF ウェブ ACL を設定して、複数の宛先のいずれかにログを発行し、そこでクエリを実行して表示できます。ウェブ ACL ログの設定および AWS WAF ログの内容の詳細については、「AWS WAF デベロッパーガイド」の「AWS WAF ウェブ ACL トラフィックのログ」を参照してください。

AWS WAF ログを中央データレイクリポジトリに集約して、それらを Athena でクエリする方法の例については、「AWS ビッグデータブログ」のブログ記事「OpenSearch Service、Amazon Athena、および Amazon QuickSight を使用して AWS WAF ログを分析する」を参照してください。

このトピックでは、パーティショニングを使用するものと使用しないもの、2 つの `CREATE TABLE` ステートメントの例を示します。

> **Note**
>
> このトピックの `CREATE TABLE` ステートメントは、v1 および v2 AWS WAF ログの両方に使用できます。v1 では、`webaclid` フィールドに ID が含まれます。v2 では、`webaclid` フィールドに完全な ARN が含まれます。ここでの `CREATE TABLE` ステートメントは、`string` データ型を使用して、アグノスティックにこのコンテンツを取り扱います。

トピック

- パーティション射影を使用して Athena で AWS WAF S3 ログ用テーブルの作成
- パーティショニングなしでの AWS WAF ログのテーブルの作成
- AWS WAF ログのクエリ例

パーティション射影を使用して Athena で AWS WAF S3 ログ用テーブルの作成

AWS WAF ログには、パーティションスキームを事前に指定できる既知の構造があるため、Athena のパーティション射影機能を使用することで、クエリランタイムを短縮し、パーティション管理を自動化することが可能です。新しいデータが追加されると、パーティション射影は新しいパーティションを自動で追加します。このため、ALTER TABLE ADD PARTITION を使用してパーティションを手動で追加する必要がなくなります。

次の CREATE TABLE ステートメント例では、指定された日付から現在までの 4 つの異なる AWS WAF リージョンの AWS ログで、パーティション射影を自動的に使用しています。この例の PARTITION BY 句は地域と日付でパーティショニングしますが、要件に応じてこれを変更できます。ログ出力と一致するように、必要に応じてフィールドを変更します。LOCATION および storage.location.template 句では、*bucket* と *accountID* の各プレースホルダーを AWS WAF ログの Amazon S3 バケットの場所を特定する値に置き換えます。projection.day.range では、*2021*/*01*/*01* を使用を開始する日に置き換えます。クエリが正常に実行されると、テーブルをクエリできます。パーティションをロードするのに、ALTER TABLE ADD PARTITION を実行する必要はありません。

```
CREATE EXTERNAL TABLE `waf_logs`(
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
  `action` string,
  `terminatingrulematchdetails` array <
                                    struct <
                                        conditiontype: string,
                                        sensitivitylevel: string,
                                        location: string,
                                        matcheddata: array < string >
                                          >
                                     >,
  `httpsourcename` string,
```

```
  `httpsourceid` string,
  `rulegrouplist` array <
                      struct <
                          rulegroupid: string,
                          terminatingrule: struct <
                                              ruleid: string,
                                              action: string,
                                              rulematchdetails: array <
                                                                  struct <
                                                                      conditiontype:
string,

sensitivitylevel: string,
                                                                      location:
string,
                                                                      matchedd ata:
array < string >
                                                                         >
                                                                  >
                                           >,
                          nonterminatingmatchingrules: array <
                                                               struct <
                                                                   ruleid: string,
                                                                   action: string,
                                                                   overriddenaction:
string,
                                                                   rulematchdetails:
array <

struct <

   conditiontype: string,

   sensitivitylevel: string,

   location: string,

   matcheddata: array < string >

      >
                                                                  >,
                                                                  challengeresponse:
struct <
```

```
 responsecode: string,

 solvetimestamp: string
                                                                        >,
                                                                captchareresponse:
 struct <

 responsecode: string,

 solvetimestamp: string
                                                                        >
                                                                    >
                                                               >,
                          excludedrules: string
                            >
                       >,
`ratebasedrulelist` array <
                         struct <
                             ratebasedruleid: string,
                             limitkey: string,
                             maxrateallowed: int
                               >
                          >,
  `nonterminatingmatchingrules` array <
                                    struct <
                                        ruleid: string,
                                        action: string,
                                        rulematchdetails: array <
                                                             struct <
                                                                 conditiontype: string,
                                                                 sensitivitylevel:
 string,
                                                                 location: string,
                                                                 matcheddata: array <
 string >
                                                                    >
                                                               >,
                                        challengeresponse: struct <
                                                            responsecode: string,
                                                            solvetimestamp: string
                                                             >,
                                        captchareresponse: struct <
                                                            responsecode: string,
```

```
                                                                solvetimestamp: string
                                                                 >
                                        >
                                     >,
  `requestheadersinserted` array <
                                struct <
                                    name: string,
                                    value: string
                                      >
                                 >,
  `responsecodesent` string,
  `httprequest` struct <
                    clientip: string,
                    country: string,
                    headers: array <
                                struct <
                                    name: string,
                                    value: string
                                      >
                                 >,
                    uri: string,
                    args: string,
                    httpversion: string,
                    httpmethod: string,
                    requestid: string
                      >,
  `labels` array <
               struct <
                   name: string
                     >
                >,
  `captcharesponse` struct <
                       responsecode: string,
                       solvetimestamp: string,
                       failureReason: string
                         >,
  `challengeresponse` struct <
                       responsecode: string,
                       solvetimestamp: string,
                       failureReason: string
                       >,
  `ja3Fingerprint` string,
  `oversizefields` string,
  `requestbodysize` int,
```

```
  `requestbodysizeinspectedbywaf` int
)
PARTITIONED BY (
`region` string,
`date` string)
ROW FORMAT SERDE
  'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/AWSLogs/accountID/WAFLogs/region/DOC-EXAMPLE-WEBACL/'
TBLPROPERTIES(
 'projection.enabled' = 'true',
 'projection.region.type' = 'enum',
 'projection.region.values' = 'us-east-1,us-west-2,eu-central-1,eu-west-1',
 'projection.date.type' = 'date',
 'projection.date.range' = '2021/01/01,NOW',
 'projection.date.format' = 'yyyy/MM/dd',
 'projection.date.interval' = '1',
 'projection.date.interval.unit' = 'DAYS',
 'storage.location.template' = 's3://DOC-EXAMPLE-BUCKET/AWSLogs/accountID/WAFLogs/
${region}/DOC-EXAMPLE-WEBACL/${date}/')
```

Note

この例にある LOCATION 句のパスの形式は標準ですが、実装した AWS WAF 設定によって異なる場合があります。例えば、次の例の AWS WAF ログパスは CloudFront 分散用です。

```
s3://DOC-EXAMPLE-BUCKET/AWSLogs/12345678910/WAFLogs/cloudfront/
cloudfronyt/2022/08/08/17/55/
```

AWS WAF ログテーブルの作成またはクエリ中に問題が発生した場合は、ログデータの場所を確定するか、AWS Support までお問い合わせください。

パーティション射影の詳細については、「Amazon Athena でのパーティション射影」を参照してください。

パーティショニングなしでの AWS WAF ログのテーブルの作成

このセクションでは、パーティショニングまたはパーティション射影を使用せずに AWS WAF ログのテーブルを作成する方法について説明します。

> **Note**
>
> パフォーマンスおよびコスト上の理由により、クエリにパーティション化されていないスキーマを使用することは推奨されません。詳細については、AWS Big Data ブログの「Amazon Athena のパフォーマンスチューニング Tips トップ 10」を参照してください。

AWS WAF テーブルを作成するには

1. 以下の DDL ステートメントをコピーして Athena コンソール内に貼り付けます。ログ出力と一致するように、必要に応じてフィールドを変更します。ログを保存するバケットに対応するように Amazon S3 バケットの `LOCATION` を変更します。

   このクエリでは、OpenX JSON SerDe を使用します。

   > **Note**
   >
   > SerDe では、各 JSON ドキュメントが、レコード内のフィールドを区切る行終端文字なしの、1 行のテキストに収まっていることを想定しています。JSON テキストがプリティプリント形式の場合、テーブルを作成した後にクエリを実行しようとすると、以下のようなエラーメッセージが表示される場合があります。「HIVE_CURSOR_ERROR: Row is not a valid JSON Object」、または「HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT」。詳細については、GitHub の OpenX SerDe のドキュメントで「JSON Data Files」(JSON データファイル) を参照してください。

```
CREATE EXTERNAL TABLE `waf_logs`(
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
  `action` string,
```

```
  `terminatingrulematchdetails` array <
                                    struct <
                                        conditiontype: string,
                                        sensitivitylevel: string,
                                        location: string,
                                        matcheddata: array < string >
                                          >
                                     >,
  `httpsourcename` string,
  `httpsourceid` string,
  `rulegrouplist` array <
                        struct <
                            rulegroupid: string,
                            terminatingrule: struct <
                                                ruleid: string,
                                                action: string,
                                                rulematchdetails: array <
                                                                    struct <
                                                                        conditiontype:
string,

sensitivitylevel: string,
                                                                        location:
string,
                                                                        matcheddata:
array < string >
                                                                           >
                                                                     >
                                               >,
                            nonterminatingmatchingrules: array <
                                                                struct <
                                                                    ruleid: string,
                                                                    action: string,
                                                                    overriddenaction:
string,
                                                                    rulematchdetails:
array <

 struct <

     conditiontype: string,

     sensitivitylevel: string,
```

```
     location: string,

     matcheddata: array < string >

        >
                                                                    >,
                                                                   challengeresponse:
struct <

responsecode: string,

solvetimestamp: string
                                                                        >,
                                                                   captcharesponse:
struct <

responsecode: string,

solvetimestamp: string
                                                                        >
                                                                    >
                                                               >,
                         excludedrules: string
                           >
                      >,
`ratebasedrulelist` array <
                         struct <
                             ratebasedruleid: string,
                             limitkey: string,
                             maxrateallowed: int
                               >
                          >,
  `nonterminatingmatchingrules` array <
                                    struct <
                                        ruleid: string,
                                        action: string,
                                        rulematchdetails: array <
                                                              struct <
                                                                  conditiontype:
string,
                                                                  sensitivitylevel:
string,
                                                                  location: string,
```

```
                                                                matcheddata: array <
string >
                                                                    >
                                                                >,
                                                challengeresponse: struct <
                                                                responsecode: string,
                                                                solvetimestamp: string
                                                                >,
                                                captcharesponse: struct <
                                                                responsecode: string,
                                                                solvetimestamp: string
                                                                >
                                                  >
                                              >,
 `requestheadersinserted` array <
                                 struct <
                                     name: string,
                                     value: string
                                       >
                                  >,
 `responsecodesent` string,
 `httprequest` struct <
                    clientip: string,
                    country: string,
                    headers: array <
                                struct <
                                    name: string,
                                    value: string
                                      >
                                 >,
                    uri: string,
                    args: string,
                    httpversion: string,
                    httpmethod: string,
                    requestid: string
                      >,
 `labels` array <
              struct <
                  name: string
                    >
               >,
 `captcharesponse` struct <
                      responsecode: string,
                      solvetimestamp: string,
```

```
                        failureReason: string
                          >,
  `challengeresponse` struct <
                        responsecode: string,
                        solvetimestamp: string,
                        failureReason: string
                        >,
  `ja3Fingerprint` string,
  `oversizefields` string,
  `requestbodysize` int,
  `requestbodysizeinspectedbywaf` int
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/'
```

2. Athena コンソールのクエリエディタで CREATE EXTERNAL TABLE ステートメントを実行します。これで waf_logs テーブルが登録され、その中のデータを Athena からのクエリに使用できるようになります。

AWS WAF ログのクエリ例

次の多くのクエリ例では、このドキュメントで前に作成したパーティション射影テーブルを使用します。また、要件に応じて例のテーブル名、列の値、その他の変数を変更します。クエリのパフォーマンスを改善してコストを削減するには、フィルター条件にパーティション列を追加します。

- Count the number of referrers that contain a specified term
- Count all matched IP addresses in the last 10 days that have matched excluded rules
- Group all counted managed rules by the number of times matched
- Group all counted custom rules by number of times matched

日付と時刻の使用

- Return the timestamp field in human-readable ISO 8601 format
- Return records from the last 24 hours
- Return records for a specified date range and IP address

- For a specified date range, count the number of IP addresses in five minute intervals
- Count the number of X-Forwarded-For IP in the last 10 days

ブロックされたリクエストとアドレスでの作業

- Extract the top 100 IP addresses blocked by a specified rule type
- Count the number of times a request from a specified country has been blocked
- Count the number of times a request has been blocked, grouping by specific attributes
- Count the number of times a specific terminating rule ID has been matched
- Retrieve the top 100 IP addresses blocked during a specified date range

Example – 指定された用語を含むリファラーの数を計上する

以下のクエリは、指定された日付範囲内での「amazon」という用語を含むリファラーの数を計上します。

```
WITH test_dataset AS
  (SELECT header FROM waf_logs
    CROSS JOIN UNNEST(httprequest.headers) AS t(header) WHERE "date" >= '2021/03/01'
    AND "date" < '2021/03/31')
SELECT COUNT(*) referer_count
FROM test_dataset
WHERE LOWER(header.name)='referer' AND header.value LIKE '%amazon%'
```

Example – 一致する除外ルールがある、過去 10 日間のすべての一致した IP アドレスを計上する

以下のクエリは、IP アドレスがルールグループ内の除外ルールに一致した過去 10 日間の回数を計上します。

```
WITH test_dataset AS
  (SELECT * FROM waf_logs
    CROSS JOIN UNNEST(rulegrouplist) AS t(allrulegroups))
SELECT
  COUNT(*) AS count,
```

```
  "httprequest"."clientip",
  "allrulegroups"."excludedrules",
  "allrulegroups"."ruleGroupId"
FROM test_dataset
WHERE allrulegroups.excludedrules IS NOT NULL AND from_unixtime(timestamp/1000) > now()
 - interval '10' day
GROUP BY "httprequest"."clientip", "allrulegroups"."ruleGroupId",
 "allrulegroups"."excludedrules"
ORDER BY count DESC
```

Example – カウントされたすべてのマネージドルールをマッチした回数でグループ化する

2022 年 10 月 27 日より前にウェブ ACL 設定でルールグループのルールアクションをカウントに設定した場合、AWS WAF はウェブ ACL JSON 内のオーバーライドを `excludedRules` として保存しました。これで、ルールをカウントにオーバーライドする JSON 設定が `ruleActionOverrides` 設定に追加されました。詳しくは、「AWS WAF デベロッパーガイド」の「ルールグループのアクションオーバーライド」をご覧ください。新しいログ構造からカウントモードのマネージドルールを抽出するには、次の例のように、`excludedRules` フィールドの代わりに `ruleGroupList` セクションの `nonTerminatingMatchingRules` をクエリします。

```
SELECT
 count(*) AS count,
 httpsourceid,
 httprequest.clientip,
 t.rulegroupid,
 t.nonTerminatingMatchingRules
FROM "waf_logs"
CROSS JOIN UNNEST(rulegrouplist) AS t(t)
WHERE action <> 'BLOCK' AND cardinality(t.nonTerminatingMatchingRules) > 0
GROUP BY t.nonTerminatingMatchingRules, action, httpsourceid, httprequest.clientip,
 t.rulegroupid
ORDER BY "count" DESC
Limit 50
```

Example – カウントされたすべてのカスタムルールをマッチした回数でグループ化する

次のクエリは、カウントされたすべてのカスタムルールをマッチした回数でグループ化します。

```
SELECT
  count(*) AS count,
         httpsourceid,
```

```
         httprequest.clientip,
         t.ruleid,
         t.action
FROM "waf_logs"
CROSS JOIN UNNEST(nonterminatingmatchingrules) AS t(t)
WHERE action <> 'BLOCK' AND cardinality(nonTerminatingMatchingRules) > 0
GROUP BY t.ruleid, t.action, httpsourceid, httprequest.clientip
ORDER BY "count" DESC
Limit 50
```

カスタムルールとマネージドルールグループのログの場所については、「AWS WAF デベロッパーガイド」の「モニタリングとチューニング」を参照してください。

日付と時刻の使用

Example – 人が読み込み可能な ISO 8601 形式のタイムスタンプフィールドを返す

以下のクエリは、from_unixtime および to_iso8601 関数を使用して、timestamp フィールドを人が読み込み可能な ISO 8601 形式 (例えば、1576280412771 ではなく 2019-12-13T23:40:12.000Z) で返します。このクエリは、HTTP ソース名、ソース ID、およびリクエストも返します。

```
SELECT to_iso8601(from_unixtime(timestamp / 1000)) as time_ISO_8601,
       httpsourcename,
       httpsourceid,
       httprequest
FROM waf_logs
LIMIT 10;
```

Example – 過去 24 時間のレコードを返す

以下のクエリは、WHERE 句でフィルターを使用して、過去 24 時間のレコードに関する HTTP ソース名、HTTP ソース ID、および HTTP リクエストフィールドを返します。

```
SELECT to_iso8601(from_unixtime(timestamp/1000)) AS time_ISO_8601,
       httpsourcename,
       httpsourceid,
       httprequest
FROM waf_logs
WHERE from_unixtime(timestamp/1000) > now() - interval '1' day
LIMIT 10;
```

Example – 指定された日付範囲と IP アドレスのレコードを返す

以下のクエリは、指定されたクライアント IP アドレスの指定された日付範囲内のレコードをリストします。

```
SELECT *
FROM waf_logs
WHERE httprequest.clientip='53.21.198.66' AND "date" >= '2021/03/01' AND "date" <
 '2021/03/31'
```

Example – 指定された日付範囲について、5 分間隔で IP アドレスの数を計上する

以下のクエリは、特定の日付範囲について、5 分間隔で IP アドレスの数を計上します。

```
WITH test_dataset AS
  (SELECT
     format_datetime(from_unixtime((timestamp/1000) -
 ((minute(from_unixtime(timestamp / 1000))%5) * 60)),'yyyy-MM-dd HH:mm') AS
 five_minutes_ts,
     "httprequest"."clientip"
     FROM waf_logs
     WHERE "date" >= '2021/03/01' AND "date" < '2021/03/31')
SELECT five_minutes_ts,"clientip",count(*) ip_count
FROM test_dataset
GROUP BY five_minutes_ts,"clientip"
```

Example – 過去 10 日間の X-Forwarded-For IP の数をカウントする

次のクエリは、リクエストヘッダーをフィルタリングし、過去 10 日間の X-Forwarded-For IP の数をカウントします。

```
WITH test_dataset AS
  (SELECT header
   FROM waf_logs
   CROSS JOIN UNNEST (httprequest.headers) AS t(header)
   WHERE from_unixtime("timestamp"/1000) > now() - interval '10' DAY)
SELECT header.value AS ip,
       count(*) AS COUNT
FROM test_dataset
WHERE header.name='X-Forwarded-For'
GROUP BY header.value
```

```
ORDER BY COUNT DESC
```

日付関数と時刻関数の詳細については、Trino ドキュメントの「Date and Time Functions and Operators」(日付と時刻の関数と演算子) を参照してください。

ブロックされたリクエストとアドレスでの作業

Example – 指定されたルールタイプによってブロックされた上位 100 個の IP アドレスを抽出する

以下のクエリは、指定した日付範囲内で RATE_BASED 終了ルールによってブロックされた上位 100 個の IP アドレスを抽出し、計上します。

```
SELECT COUNT(httpRequest.clientIp) as count,
httpRequest.clientIp
FROM waf_logs
WHERE terminatingruletype='RATE_BASED' AND action='BLOCK' and "date" >= '2021/03/01'
AND "date" < '2021/03/31'
GROUP BY httpRequest.clientIp
ORDER BY count DESC
LIMIT 100
```

Example – 指定された国からのリクエストがブロックされた回数を計上する

次のクエリは、アイルランド (IE) に属し、RATE_BASED 終了ルールによってブロックされた IP アドレスからリクエストが到着した回数を数えます。

```
SELECT
  COUNT(httpRequest.country) as count,
  httpRequest.country
FROM waf_logs
WHERE
  terminatingruletype='RATE_BASED' AND
  httpRequest.country='IE'
GROUP BY httpRequest.country
ORDER BY count
LIMIT 100;
```

Example – リクエストがブロックされた回数を計上し、特定の属性でグループ化する

以下のクエリは、リクエストがブロックされた回数をカウントし、結果を WebACL、RuleId、ClientIP、および HTTP リクエスト URI によってグループ化します。

```
SELECT
  COUNT(*) AS count,
  webaclid,
  terminatingruleid,
  httprequest.clientip,
  httprequest.uri
FROM waf_logs
WHERE action='BLOCK'
GROUP BY webaclid, terminatingruleid, httprequest.clientip, httprequest.uri
ORDER BY count DESC
LIMIT 100;
```

Example – 特定の終了ルール ID が一致した回数を計上する

以下のクエリは、特定の終了ルール ID が (WHERE terminatingruleid='e9dd190d-7a43-4c06-bcea-409613d9506e') に一致した回数を計上します。その結果を WebACL、Action、ClientIP、および HTTP Request URI によってグループ化します。

```
SELECT
  COUNT(*) AS count,
  webaclid,
  action,
  httprequest.clientip,
  httprequest.uri
FROM waf_logs
WHERE terminatingruleid='e9dd190d-7a43-4c06-bcea-409613d9506e'
GROUP BY webaclid, action, httprequest.clientip, httprequest.uri
ORDER BY count DESC
LIMIT 100;
```

Example – 指定した日付範囲内でブロックされた上位 100 個の IP アドレスを取得する

以下のクエリは、指定された日付範囲内でブロックされた上位 100 個の IP アドレスを抽出します。このクエリは、IP アドレスがブロックされた回数もリストします。

```
SELECT "httprequest"."clientip", "count"(*) "ipcount", "httprequest"."country"
FROM waf_logs
WHERE "action" = 'BLOCK' and "date" >= '2021/03/01'
AND "date" < '2021/03/31'
GROUP BY "httprequest"."clientip", "httprequest"."country"
```

```
ORDER BY "ipcount" DESC limit 100
```

Amazon S3 のクエリについては、以下のトピックを参照してください。

- AWS ナレッジセンターの「Athena を使用して Amazon S3 サーバーのアクセスログを分析するにはどうすればよいですか?」
- 「Amazon Simple Storage Service ユーザーガイド」の「Amazon Athena を使用してリクエストのアクセスログをクエリする」
- Amazon Simple Storage Service ユーザーガイドの「AWS CloudTrail を使用した Amazon S3 リクエストの識別」

Simple Storage Service (Amazon S3) に保存されたウェブサーバーログのクエリ

Athena を使用して、Amazon S3 に保存されているウェブサーバーログをクエリすることができます。このセクションのトピックでは、Athena でテーブルを作成して、さまざまな形式のウェブサーバーログをクエリする方法を説明します。

トピック

- Simple Storage Service (Amazon S3) に保存されている Apache ログのクエリ
- Simple Storage Service (Amazon S3) に保存されているインターネットインフォメーションサーバー (IIS) ログのクエリ

Simple Storage Service (Amazon S3) に保存されている Apache ログのクエリ

Amazon Athena を使用して、Simple Storage Service (Amazon S3) アカウントに保存されている Apache HTTP サーバーのログファイルをクエリできます。このトピックでは、テーブルスキーマを作成して、一般的なログ形式の Apache アクセスログファイルをクエリする方法を説明します。

一般的なログ形式のフィールドには、クライアント IP アドレス、クライアント ID、ユーザー ID、リクエスト受信時のタイムスタンプ、クライアントリクエストのテキスト、サーバステータスコード、およびクライアントに返されたオブジェクトのサイズが含まれます。

以下の例は、Apache の一般的なログ形式を示しています。

```
198.51.100.7 - Li [10/Oct/2019:13:55:36 -0700] "GET /logo.gif HTTP/1.0" 200 232
198.51.100.14 - Jorge [24/Nov/2019:10:49:52 -0700] "GET /index.html HTTP/1.1" 200 2165
```

```
198.51.100.22 - Mateo [27/Dec/2019:11:38:12 -0700] "GET /about.html HTTP/1.1" 200 1287
198.51.100.9 - Nikki [11/Jan/2020:11:40:11 -0700] "GET /image.png HTTP/1.1" 404 230
198.51.100.2 - Ana [15/Feb/2019:10:12:22 -0700] "GET /favicon.ico HTTP/1.1" 404 30
198.51.100.13 - Saanvi [14/Mar/2019:11:40:33 -0700] "GET /intro.html HTTP/1.1" 200 1608
198.51.100.11 - Xiulan [22/Apr/2019:10:51:34 -0700] "GET /group/index.html HTTP/1.1"
 200 1344
```

Athena での Apache ログ用のテーブルの作成

Amazon S3 に保存されている Apache ログをクエリする前に、Athena のテーブルスキーマを作成して、Athena がログデータを読み取ることができるようにする必要があります Apache ログ用の Athena テーブルを作成するには、Grok SerDe を使用できます。Grok SerDe の使用に関する詳細については、「AWS Glue デベロッパーガイド」の「Grok カスタム分類子の書き込み」を参照してください。

Athena で Apache ウェブサーバーログ用のテーブルを作成する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. 以下の DDL ステートメントを Athena クエリエディタに貼り付けます。LOCATION 's3://DOC-EXAMPLE-BUCKET/*apache-log-folder*/' の値を変更して、Amazon S3 内の Apache ログをポイントするようにします。

```
CREATE EXTERNAL TABLE apache_logs (
  client_ip string,
  client_id string,
  user_id string,
  request_received_time string,
  client_request string,
  server_status string,
  returned_obj_size string
  )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.format'='^%{IPV4:client_ip} %{DATA:client_id} %{USERNAME:user_id}
 %{GREEDYDATA:request_received_time} %{QUOTEDSTRING:client_request}
 %{DATA:server_status} %{DATA: returned_obj_size}$'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
```

```
LOCATION
   's3://DOC-EXAMPLE-BUCKET/apache-log-folder/';
```

3. Athena コンソールでクエリを実行して、apache_logs テーブルを登録します。クエリが完了すると、Athena からログをクエリできるようになります。

Apache ログに対する選択クエリの例

Example – 404 エラーのフィルタリング

以下のクエリ例は、リクエストの受信時刻、クライアントリクエストのテキスト、およびサーバーステータスコードを apache_logs テーブルから選択します。WHERE 句は、HTTP ステータスコード 404 (ページが見つかりません) をフィルターします。

```
SELECT request_received_time, client_request, server_status
FROM apache_logs
WHERE server_status = '404'
```

以下の画像は、Athena クエリエディタでのクエリの結果を示しています。

Results

	request_received_time	client_request	server_status
1	[11/Jan/2020:11:40:11 -0700]	GET /image.png HTTP/1.1	404
2	[15/Feb/2019:10:12:22 -0700]	GET /favicon.ico HTTP/1.1	404

Example – 正常に行われたリクエストのフィルタリング

以下の例のクエリでは、ユーザー ID、リクエスト受信時刻、クライアントリクエストのテキスト、およびサーバーステータスコードを apache_logs テーブルから選択しています。WHERE 句は、HTTP ステータスコード 200 (成功) をフィルターします。

```
SELECT user_id, request_received_time, client_request, server_status
FROM apache_logs
WHERE server_status = '200'
```

以下の画像は、Athena クエリエディタでのクエリの結果を示しています。

Results

	user_id	request_received_time	client_request	server_status
1	Li	[10/Oct/2019:13:55:36 -0700]	GET /logo.gif HTTP/1.0	200
2	Jorge	[24/Nov/2019:10:49:52 -0700]	GET /index.html HTTP/1.1	200
3	Mateo	[27/Dec/2019:11:38:12 -0700]	GET /about.html HTTP/1.1	200
4	Saanvi	[14/Mar/2019:11:40:33 -0700]	GET /intro.html HTTP/1.1	200
5	Xiulan	[22/Apr/2019:10:51:34 -0700]	GET /group/index.html HTTP/1.1	200

Example — タイムスタンプによるフィルタリング

次の例では、リクエストの受信時間が指定されたタイムスタンプよりも長いレコードをクエリします。

```
SELECT * FROM apache_logs WHERE request_received_time > 10/Oct/2023:00:00:00
```

Simple Storage Service (Amazon S3) に保存されているインターネットインフォメーションサーバー (IIS) ログのクエリ

Amazon Athena を使用して、Amazon S3 アカウントに保存されている Microsoft インターネットインフォメーションサービス (IIS) ウェブサーバーログをクエリできます。IIS はさまざまなログファイル形式を使用しますが、このトピックでは、W3C 拡張および IIS ログファイル形式のログを Athena からクエリするためのテーブルスキーマの作成方法について説明します。

W3C 拡張および IIS ログファイル形式は 1 文字の区切り文字 (それぞれスペースとカンマ) を使用し、引用符で囲まれた値を持たないため、LazySimpleSerDe を使用してそれらの Athena テーブルを作成できます。

W3C 拡張ログファイル形式

W3C 拡張ログファイルデータ形式には、スペースで区切られたフィールドがあります。W3C 拡張ログに表示されるフィールドは、どのログフィールドを含めるかを選択するウェブサーバー管理者によって決定されます。以下のログデータ例には、`date`, `time`、`c-ip`、`s-ip`、`cs-method`、`cs-uri-stem`、`sc-status`、`sc-bytes`、`cs-bytes`、`time-taken`、および `cs-version` のフィールドがあります。

```
2020-01-19 22:48:39 203.0.113.5 198.51.100.2 GET /default.html 200 540 524 157 HTTP/1.0
```

```
2020-01-19 22:49:40 203.0.113.10 198.51.100.12 GET /index.html 200 420 324 164 HTTP/1.0
2020-01-19 22:50:12 203.0.113.12 198.51.100.4 GET /image.gif 200 324 320 358 HTTP/1.0
2020-01-19 22:51:44 203.0.113.15 198.51.100.16 GET /faq.html 200 330 324 288 HTTP/1.0
```

Athena での W3C 拡張ログ用のテーブルの作成

W3C 拡張ログをクエリする前に、テーブルスキーマを作成して Athena がログデータを読み取ることができるようにする必要があります。

Athena で W3C 拡張ログ用のテーブルを作成する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. 以下の点に注意しながら、以下のような DDL ステートメントを Athena コンソールに貼り付けます。

   a. クエリするログのフィールドに対応するように、例にある列を追加または削除します。

   b. W3C 拡張ログファイル形式の列名にはハイフン (-) が含まれていますが、例の CREATE TABLE ステートメントは、Athena の命名規則に従って、それらをアンダースコア (_) に置き換えます。

   c. スペース区切り文字を指定するには、FIELDS TERMINATED BY ' ' を使用します。

   d. LOCATION 's3://DOC-EXAMPLE-BUCKET/*w3c-log-folder*/' の値を変更して、Amazon S3 内の W3C 拡張ログをポイントするようにします。

```
CREATE EXTERNAL TABLE `iis_w3c_logs`(
  date_col string,
  time_col string,
  c_ip string,
  s_ip string,
  cs_method string,
  cs_uri_stem string,
  sc_status string,
  sc_bytes string,
  cs_bytes string,
  time_taken string,
  cs_version string
  )
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ' '
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
```

```
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION   's3://DOC-EXAMPLE-BUCKET/w3c-log-folder/'
```

3. Athena コンソールでクエリを実行して、iis_w3c_logs テーブルを登録します。クエリが完了すると、Athena からログをクエリできるようになります。

W3C 拡張ログに対する選択クエリの例

以下のクエリ例は、日付、時刻、リクエストターゲット、およびリクエストにかかった時間をテーブル iis_w3c_logs から選択します。WHERE 句は、HTTP メソッドが GET で、HTTP ステータスコードが 200 (成功) のケースをフィルターします。

```
SELECT date_col, time_col, cs_uri_stem, time_taken
FROM iis_w3c_logs
WHERE cs_method = 'GET' AND sc_status = '200'
```

以下の画像は、Athena クエリエディタでのクエリの結果を示しています。

Results

	date_col	time_col	cs_uri_stem	time_taken
1	2020-01-19	22:48:39	/default.html	157
2	2020-01-19	22:49:40	/index.html	164
3	2020-01-19	22:50:12	/image.gif	358
4	2020-01-19	22:51:44	/faq.html	288

日付フィールドと時刻フィールドの統合

スペースで区切られた date フィールドと time フィールドはログソースデータ内の個別のエントリですが、必要に応じてそれらをタイムスタンプに統合できます。SELECT または CREATE TABLE AS SELECT クエリで concat() および date_parse() 関数を使用して、日付と時刻の列を連結し、タイムスタンプ形式に変換します。以下の例では、CTAS クエリを使用して、derived_timestamp 列が含まれる新しいテーブルを作成しています。

```
CREATE TABLE iis_w3c_logs_w_timestamp AS
```

```
SELECT
  date_parse(concat(date_col,' ', time_col),'%Y-%m-%d %H:%i:%s') as derived_timestamp,
  c_ip,
  s_ip,
  cs_method,
  cs_uri_stem,
  sc_status,
  sc_bytes,
  cs_bytes,
  time_taken,
  cs_version
FROM iis_w3c_logs
```

テーブルが作成されたら、次の例にあるように、新しいタイムスタンプ列を直接クエリできます。

```
SELECT derived_timestamp, cs_uri_stem, time_taken
FROM iis_w3c_logs_w_timestamp
WHERE cs_method = 'GET' AND sc_status = '200'
```

以下の画像は、クエリの結果を示しています。

Results

	derived_timestamp	cs_uri_stem	time_taken
1	2020-01-19 22:48:39.000	/default.html	157
2	2020-01-19 22:49:40.000	/index.html	164
3	2020-01-19 22:50:12.000	/image.gif	358
4	2020-01-19 22:51:44.000	/faq.html	288

IIS ログファイル形式

W3C 拡張形式とは異なり、IIS ログファイル形式には、固定されたフィールドのセットがあり、区切り文字としてカンマが含まれています。LazySimpleSerDe はカンマを区切り文字として扱い、カンマの後のスペースを次のフィールドの先頭として扱います。

以下の例は、IIS ログファイル形式のサンプルデータを示しています。

```
203.0.113.15, -, 2020-02-24, 22:48:38, W3SVC2, SERVER5, 198.51.100.4, 254, 501, 488,
 200, 0, GET, /index.htm, -,
```

```
203.0.113.4, -, 2020-02-24, 22:48:39, W3SVC2, SERVER6, 198.51.100.6, 147, 411, 388,
 200, 0, GET, /about.html, -,
203.0.113.11, -, 2020-02-24, 22:48:40, W3SVC2, SERVER7, 198.51.100.18, 170, 531, 468,
 200, 0, GET, /image.png, -,
203.0.113.8, -, 2020-02-24, 22:48:41, W3SVC2, SERVER8, 198.51.100.14, 125, 711, 868,
 200, 0, GET, /intro.htm, -,
```

Athena での IIS ログファイル用のテーブルの作成

Amazon S3 に保存されている IIS ログファイル形式のログをクエリするには、まずテーブルスキーマを作成して、Athena がログデータを読み取ることができるようにします。

Athena で IIS ログファイル形式のログ用のテーブルを作成する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. 以下の点に注意しながら、以下の DDL ステートメントを Athena コンソールに貼り付けます。
   a. カンマ区切り文字を指定するには、`FIELDS TERMINATED BY ','` を使用します。
   b. Amazon S3 内にある IIS ログ形式のログファイルをポイントするように、LOCATION 's3://DOC-EXAMPLE-BUCKET/*iis-log-file-folder*/' 内の値を変更します。

```
CREATE EXTERNAL TABLE `iis_format_logs`(
client_ip_address string,
user_name string,
request_date string,
request_time string,
service_and_instance string,
server_name string,
server_ip_address string,
time_taken_millisec string,
client_bytes_sent string,
server_bytes_sent string,
service_status_code string,
windows_status_code string,
request_type string,
target_of_operation string,
script_parameters string
   )
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
```

```
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/iis-log-file-folder/'
```

3. Athena コンソールでクエリを実行して、iis_format_logs テーブルを登録します。クエリが完了すると、Athena からログをクエリできるようになります。

IIS ログ形式の選択クエリの例

以下のクエリ例は、リクエスト日、リクエスト時刻、リクエストターゲット、およびリクエストにかかったミリ秒単位の時間をテーブル iis_format_logs から選択します。WHERE 句は、リクエストタイプが GET で、HTTP ステータスコードが 200 (成功) のケースをフィルターします。このクエリでは、クエリを正常に行うために ' GET' および ' 200' の先頭にスペースが必要であることに注意してください。

```
SELECT request_date, request_time, target_of_operation, time_taken_millisec
FROM iis_format_logs
WHERE request_type = ' GET' AND service_status_code = ' 200'
```

以下の画像は、サンプルデータに対するクエリの結果の例を示しています。

Results

	request_date	request_time	target_of_operation	time_taken_millisec
1	2020-02-24	22:48:38	/index.htm	254
2	2020-02-24	22:48:39	/about.html	147
3	2020-02-24	22:48:40	/image.png	170
4	2020-02-24	22:48:41	/intro.htm	125

NCSA ログファイル形式

IIS は NCSA ロギング形式も使用します。この形式には、スペースで区切られた固定数の ASCII テキスト形式のフィールドがあります。その構造は、Apache アクセスログに使用される一般的なログ形式に似ています。NCSA 共通ログ形式のフィールドには、クライアント IP アドレス、クライアント ID (通常は使用されません)、ドメイン\ユーザー ID、リクエスト受信時のタイムスタンプ、クライアントリクエストのテキスト、サーバステータスコード、およびクライアントに返されたオブジェクトのサイズが含まれます。

以下の例は、IIS 用に記録された NCSA 共通ログ形式のデータを示しています。

```
198.51.100.7 - ExampleCorp\Li [10/Oct/2019:13:55:36 -0700] "GET /logo.gif HTTP/1.0" 200
 232
198.51.100.14 - AnyCompany\Jorge [24/Nov/2019:10:49:52 -0700] "GET /index.html
 HTTP/1.1" 200 2165
198.51.100.22 - ExampleCorp\Mateo [27/Dec/2019:11:38:12 -0700] "GET /about.html
 HTTP/1.1" 200 1287
198.51.100.9 - AnyCompany\Nikki [11/Jan/2020:11:40:11 -0700] "GET /image.png HTTP/1.1"
 404 230
198.51.100.2 - ExampleCorp\Ana [15/Feb/2019:10:12:22 -0700] "GET /favicon.ico HTTP/1.1"
 404 30
198.51.100.13 - AnyCompany\Saanvi [14/Mar/2019:11:40:33 -0700] "GET /intro.html
 HTTP/1.1" 200 1608
198.51.100.11 - ExampleCorp\Xiulan [22/Apr/2019:10:51:34 -0700] "GET /group/index.html
 HTTP/1.1" 200 1344
```

Athena での IIS NCSA ログ用のテーブルの作成

CREATE TABLE ステートメントには、Apache のウェブサーバーログの場合と同じような Grok SerDe および grok パターンを使用できます。Apache ログとは異なり、grok パターンは domain\user_id 内のバックスラッシュの存在を考慮するために、%{DATA:user_id} ではなく %{USERNAME:user_id} を 3 番目のフィールドに使用します。Grok SerDe の使用に関する詳細については、「AWS Glue デベロッパーガイド」の「Grok カスタム分類子の書き込み」を参照してください。

Athena で IIS NCSA ウェブサーバーログ用のテーブルを作成する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. 以下の DDL ステートメントを Athena クエリエディタに貼り付けます。LOCATION 's3://DOC-EXAMPLE-BUCKET/*iis-ncsa-logs*/' の値を変更して、Amazon S3 内の IIS NCSA ログをポイントするようにします。

```
CREATE EXTERNAL TABLE iis_ncsa_logs(
  client_ip string,
  client_id string,
  user_id string,
  request_received_time string,
  client_request string,
  server_status string,
  returned_obj_size string
```

```
  )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.format'='^%{IPV4:client_ip} %{DATA:client_id} %{DATA:user_id}
 %{GREEDYDATA:request_received_time} %{QUOTEDSTRING:client_request}
 %{DATA:server_status} %{DATA: returned_obj_size}$'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://DOC-EXAMPLE-BUCKET/iis-ncsa-logs/';
```

3. Athena コンソールでクエリを実行して、iis_ncsa_logs テーブルを登録します。クエリが完了すると、Athena からログをクエリできるようになります。

IIS NCSA ログに対する選択クエリの例

Example – 404 エラーのフィルタリング

以下のクエリ例は、リクエストの受信時刻、クライアントリクエストのテキスト、およびサーバーステータスコードを iis_ncsa_logs テーブルから選択します。WHERE 句は、HTTP ステータスコード 404 (ページが見つかりません) をフィルターします。

```
SELECT request_received_time, client_request, server_status
FROM iis_ncsa_logs
WHERE server_status = '404'
```

以下の画像は、Athena クエリエディタでのクエリの結果を示しています。

Results

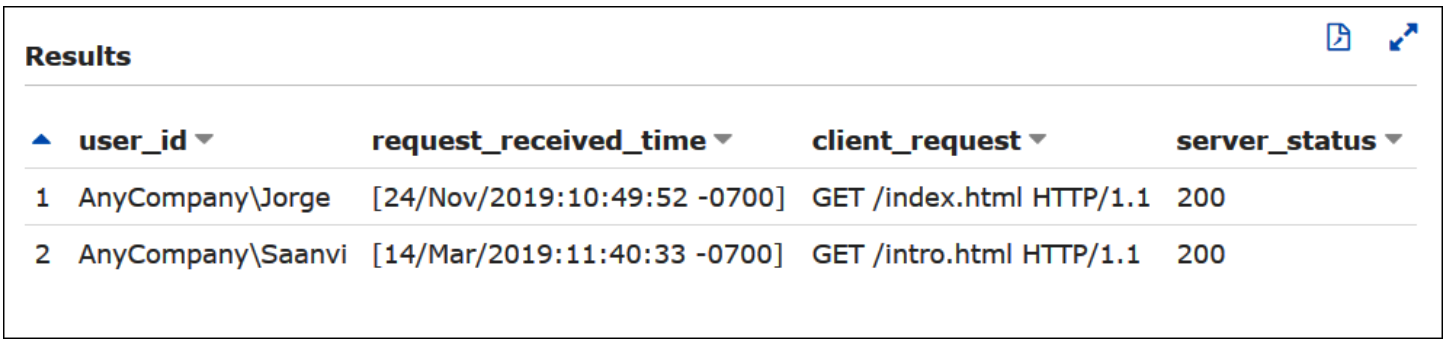
	request_received_time	client_request	server_status
1	[11/Jan/2020:11:40:11 -0700]	GET /image.png HTTP/1.1	404
2	[15/Feb/2019:10:12:22 -0700]	GET /favicon.ico HTTP/1.1	404

Example – 特定のドメインからの正常に行われたリクエストのフィルタリング

以下のクエリ例は、ユーザー ID、リクエスト受信時刻、クライアントリクエストのテキスト、およびサーバーステータスコードを iis_ncsa_logs テーブルから選択します。WHERE 句は、AnyCompany ドメイン内のユーザーから HTTP ステータスコードが 200 (成功) のリクエストをフィルターします。

```
SELECT user_id, request_received_time, client_request, server_status
FROM iis_ncsa_logs
WHERE server_status = '200' AND user_id LIKE 'AnyCompany%'
```

以下の画像は、Athena クエリエディタでのクエリの結果を示しています。



Results

	user_id	request_received_time	client_request	server_status
1	AnyCompany\Jorge	[24/Nov/2019:10:49:52 -0700]	GET /index.html HTTP/1.1	200
2	AnyCompany\Saanvi	[14/Mar/2019:11:40:33 -0700]	GET /intro.html HTTP/1.1	200

Athena ACID トランザクションの使用

「ACID トランザクション」という用語は、データベーストランザクションにおけるデータの整合性を保証する一連のプロパティ (アトミック性、整合性、分離性、および耐久性) を指します。ACID トランザクションを使用すると、データレイクに対するクエリにおいて読み込みの整合性を維持することで、既存のクエリを分離しながら、複数のユーザーがアトミックな方法で Amazon S3 オブジェクトを同時に確実に追加および削除できます。Athena ACID トランザクションは、Athena SQL データ操作言語 (DML) による挿入、削除、更新、およびタイムトラベルオペレーションのための、単一テーブルのサポートを追加します。お客様ならびに同時に実行している複数のユーザーは、Athena ACID トランザクションを使用することで、Amazon S3 の行レベルのデータ対して信頼性の高い変更を行うことができます。Athena トランザクションにより、ロックのセマンティクスと調整が自動的に処理されます。カスタムのレコードロックソリューションは必要ありません。

使い慣れた SQL 構文で Athena ACID トランザクションを行うことで、ビジネスや規制に関するデータの更新が簡素化されます。たとえば、データの消去リクエストに応答する場合は、SQL の DELETE オペレーションを実行します。手動によるレコードの修正には、単一の UPDATE ステートメントを使用します。また、最近削除されたデータを回復する場合は、SELECT ステートメントにより、タイムトラベルクエリを発行します。

Athena ACID トランザクションは共有テーブルフォーマット上に構築されるため、Amazon EMR や Apache Spark など、同じく共有テーブル形式をサポートしている他のサービスやエンジンとの互換性を持ちます。

Athena トランザクションは、Athena コンソール、API オペレーション、ODBC および JDBC ドライバを介して利用できます。

トピック

- Linux Foundation Delta Lake テーブルへのクエリ
- Athena を使用した Apache Hudi データセットのクエリ
- Apache Iceberg テーブルの使用

Linux Foundation Delta Lake テーブルへのクエリ

Linux Foundation Delta Lake は、ビッグデータ分析用のテーブル形式です。Amazon Athena を使用すると、マニフェストファイルを生成したり、`MSCK REPAIR` ステートメントを実行したりすることなく、Amazon S3 に保存されている Delta Lake テーブルを直接読み込むことができます。

Delta Lake 形式には、各データファイルの列ごとの最小値と最大値が格納されます。Athena の実装では、この情報を利用して述語でのファイルスキップが可能になり、不要なファイルが考慮対象から除外されます。

考慮事項と制約事項

Athena の Delta Lake サポートには、次の考慮事項と制限があります。

- [AWS Glue カタログ付きのテーブルのみ] - Delta Lake のネイティブサポートは、AWS Glue に登録されたテーブルでのみサポートされます。別のメタストアに登録されている Delta Lake テーブルがある場合でも、それを保持してプライマリのメタストアとして扱うことができます。Delta Lake のメタデータはメタストアではなくファイルシステム (Amazon S3 など) に保存されるため、Athena では Delta Lake テーブルから読み込むには AWS Glue にある場所プロパティのみが必要です。
- [V3 engine only] (V3 エンジンのみ) - Delta Lake クエリは Athena エンジンバージョン 3 でのみサポートされます。作成するワークグループが Athena エンジンバージョン 3 を使用するように設定する必要があります。
- Delta Lake リーダーバージョン – 最大でバージョン 3 の Delta Lake リーダープロトコルがサポートされています。

- 列マッピングと timestampNtz – Delta 列マッピングです。これは、Delta テーブル列と基盤となる Parquet ファイル列が異なる名前を使用できるようにし、タイムゾーンなしのタイムスタンプ (timestampNtz) がサポートされます。
- [No time travel support] (タイムトラベルのサポートなし) - Delta Lake のタイムトラベル機能を使用するクエリはサポートされていません。
- [Read only] (読み込み専用) - UPDATE、INSERT または DELETE のような DML ステートメントの記述はサポートされていません。
- Lake Formation サポート — Lake Formation の統合は、AWS Glueと同期したスキーマの Delta Lake テーブルに利用できます。詳細については、「AWS Lake Formation デベロッパーガイド」の「Amazon Athena での AWS Lake Formation の使用」および「Delta Lake テーブルのアクセス許可をセットアップする」を参照してください。
- [Limited DDL support] (限定的な DDL サポート) - 次の DDL ステートメントは以下の CREATE EXTERNAL TABLE、SHOW COLUMNS、SHOW TBLPROPERTIES、SHOW PARTITIONS、SHOW CREATE TABLE、および DESCRIBE をサポートしています。CREATE EXTERNAL TABLE ステートメントの使用方法については、「開始」のセクションを参照してください。
- S3 Glacier オブジェクトのスキップはサポートされていません — Linux Foundation Delta Lake テーブルのオブジェクトが Amazon S3 Glacier ストレージクラスにある場合、read_restored_glacier_objects テーブルプロパティを false に設定しても効果はありません。

  例えば、次のコマンドを実行したとします。

  ```
  ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
  ```

  Iceberg および Delta Lake テーブルでは、コマンドは「サポートされていないテーブルプロパティキー: read_restored_glacier_objects」というエラーを生成します。Hudi テーブルでは、ALTER TABLE コマンドはエラーを発生しませんが、Amazon S3 Glacier オブジェクトはまだスキップされません。ALTER TABLE コマンドの後に SELECT クエリを実行すると、引き続きすべてのオブジェクトが返されます。

サポートされる非パーティション列のデータ型

非パーティション列では、Athena がサポートしている CHAR 以外のデータ型をすべてのデータ型がサポートされます (CHAR は Delta Lake プロトコル自体ではサポートされていません)。サポートされているデータ型は次のとおりです。

```
boolean
tinyint
smallint
integer
bigint
double
float
decimal
varchar
string
binary
date
timestamp
array
map
struct
```

サポートされているパーティション列のデータ型

Athena は、パーティション列で次のデータ型を持つテーブルをサポートしています。

```
boolean
integer
smallint
tinyint
bigint
decimal
float
double
date
timestamp
varchar
```

Athena でのデータ型についての詳細は、「Amazon Athena のデータ型」を参照してください。

開始

クエリを実行するには、Delta Lake テーブルが AWS Glue に存在している必要があります。テーブルが Amazon S3 にあるものの AWS Glue にはない場合には、次の構文を使用して `CREATE EXTERNAL TABLE` ステートメントを実行します。テーブルがすでに AWS Glue に存在する場合 (たとえば、Apache Spark や AWS Glue で他のエンジンを使用している場合)、この手順は省略できます。

```
CREATE EXTERNAL TABLE
  [db_name.]table_name
  LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
  TBLPROPERTIES ('table_type' = 'DELTA')
```

列定義、SerDe ライブラリ、およびその他のテーブルプロパティが省略されていることに注意してください。従来の Hive テーブルとは異なり、Delta Lake テーブルのメタデータは Delta Lake のトランザクションログから推測され、AWS Glue に直接同期されます。

> **Note**
>
> Delta Lake テーブルでは、LOCATION および table_type 以外のプロパティを含む CREATE TABLE ステートメントは使用できません。

Delta Lake テーブルを読み込む

Delta Lake テーブルをクエリするには、次の標準 SQL SELECT 構文を使用します。

```
[ WITH with_query [, ...] ]SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ OFFSET count [ ROW | ROWS ] ]
[ LIMIT [ count | ALL ] ]
```

SELECT 構文の詳細については、Athena ドキュメントの「SELECT」を参照してください。

Delta Lake 形式には、各データファイルの列ごとの最小値と最大値が格納されます。Athena では、この情報を利用して述語でのファイルスキップが可能になり、不要なファイルが考慮対象から除外されます。

Delta Lake メタデータの同期

Athena を使用して Delta Lake テーブルを作成する場合、Athena はスキーマ、パーティション列、およびテーブルプロパティなどのテーブルメタデータを含むテーブルメタデータを AWS Glue に同期します。時間が経過すると、このメタデータは、トランザクションログ内の基になるテーブルメタ

データとの同期を失う可能性があります。テーブルを最新の状態に保つには、以下のいずれかのオプションを選択できます。

- Delta Lake テーブルの AWS Glue クローラーを使用する。詳細については、「AWS Big Data ブログ」の「AWS Glue クローラーによるネイティブな Delta Lake テーブルサポートの概要」と「AWS Glue デベロッパーガイド」の「AWS Glue クローラーのスケジュール」を参照してください。
- Athena にテーブルをドロップして再作成する。
- SDK、CLI、または AWS Glue コンソールを使用して、AWS Glue のスキーマを手動で更新する。

以下の機能を使用するには、AWS Glue スキーマがトランザクションログと常に同じスキーマである必要があることに注意してください。

- Lake Formation
- ビュー
- 行フィルターと列フィルター

この機能がワークフローで必要なく、この互換性を維持しない場合は、Athena で `CREATE TABLE` DDL を使用し、AWS Glue で Amazon S3 パスを SerDe パラメータとして追加できます。

Athena と AWS Glue コンソールを使用して Delta Lake テーブルを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. Athena クエリエディタで、次の DDL を使用して Delta Lake テーブルを作成します。この方法を使用する場合、`TBLPROPERTIES` は `'table_type' = 'delta'` ではなく `'spark.sql.sources.provider' = 'delta'`の値はである必要があります。

   Apache Spark (Athena for Apache Spark) または他のほとんどのエンジンを使用してテーブルを作成すると、(タイプ `array<string>` の `col` という名前の単一の列を含む) この同じスキーマが挿入されることに注意してください。

   ```
   CREATE EXTERNAL TABLE
      [db_name.]table_name(col array<string>)
      LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
      TBLPROPERTIES ('spark.sql.sources.provider' = 'delta')
   ```

3. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。

4. ナビゲーションペインで、[データカタログ]、[テーブル] を選択します。
5. テーブルのリストで、目的のテーブルのリンクを選択します。
6. テーブルのページで、[アクション]、[テーブルの編集] を選択します。
7. [Serde パラメータ] セクションで、値 **s3://DOC-EXAMPLE-BUCKET/*your-folder*/** を含むキー **path** を追加します。
8. [Save] を選択します。

追加リソース

AWS Glue での Delta Lake テーブルの使用および Athena でのそれらのクエリの説明については、AWS Big Data Blog の「Handle UPSERT data operations using open-source Delta Lake and AWS Glue」を参照してください。

Athena を使用した Apache Hudi データセットのクエリ

Apache Hudi は、増分データ処理を簡素化するオープンソースのデータ管理フレームワークです。レコードレベルの挿入、更新、upsert、および削除アクションは、はるかに細かく処理され、オーバーヘッドを軽減します。`Upsert` は、レコードがまだ存在しない場合は既存のデータセットに挿入し、存在する場合はレコードを更新する機能のことです。

Hudi は、分析のパフォーマンス問題を引き起こす可能性のある小さなファイルを多数作成することなく、データの挿入と更新イベントを処理します。Apache Hudi は自動的に変更を追跡し、ファイルをマージして、最適なサイズを維持します。これにより、多数の小さなファイルをモニタリングし、より少ない数の大きなファイルに書き換えるカスタムソリューションを構築する必要がなくなります。

Hudi データセットは、次のユースケースに適しています。

- 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、個人情報を削除したり、個人データの使用方法を変更したりする権利を実施するプライバシー規制を遵守する。
- 特定のデータの挿入および更新イベントを必要とするセンサーやその他のモノのインターネット (IoT) デバイスからのストリーミングデータを操作する。
- 変更データキャプチャ (CDC) システムの実装

Hudi によって管理されるデータセットは、オープンストレージ形式を使用して Amazon S3 に保存されます。現在、Athena は短縮された Hudi データセットを読み取ることができますが、Hudi

データを書き込むことはできません。Athena は、Athena エンジンバージョン 2 で Hudi バージョン 0.8.0 まで、Athena エンジンバージョン 3 では Hudi バージョン 0.14.0 までをサポートします。これは変更される可能性があります。Athena は、Hudi のそれ以降のバージョンで作成されたテーブルとの読み取り互換性は保証できません。Athena エンジンのバージョニングについては、「Athena エンジンのバージョニング」を参照してください。Hudi の機能とバージョニングの詳細については、Apache ウェブサイトの「Hudi ドキュメント」を参照してください。

Hudi データセットテーブルタイプ

Hudi データセットは、次のタイプのいずれかになります。

- 書き込み時コピー (CoW) – データは列形式 (Parquet) で保存され、更新ごとに書き込み中にファイルの新しいバージョンが作成されます。
- 読み取り時マージ (MoR) – データは、列形式 (Parquet) 形式と行形式 (Avro) の組み合わせを使用して保存されます。更新は、行形式の `delta` ファイルに記録され、必要に応じて圧縮されて、新しいバージョンの列形式のファイルが作成されます。

CoW データセットでは、レコードが更新されるたびに、そのレコードを含むファイルが更新された値で書き換えられます。MoR データセットでは、更新があるたびに、Hudi によって変更されたレコードの行のみが書き込まれます。MoR は、読み取りが少なく書き込みまたは変更が多いワークロードに適しています。CoW は、頻繁に変更されないデータの読み取りが多いワークロードに適しています。

Hudi にはデータにアクセスするためのクエリタイプが 3 種類用意されています。

- スナップショットクエリ – 指定されたコミットまたは圧縮アクションの時点で最新のテーブルスナップショットを参照するクエリです。MoR テーブルの場合、スナップショットクエリは、クエリ時に最新のファイルスライスのベースファイルとデルタファイルをマージして、テーブルの最新の状態を提示します。
- 増分クエリ – クエリは、指定されたコミット/圧縮以降にテーブルに書き込まれた新しいデータのみを参照します。これにより、増分データパイプラインを有効にするための変更ストリームが効果的に提供されます。
- 読み取り最適化クエリ – MoR テーブルの場合、クエリは圧縮された最新のデータを参照します。CoW テーブルの場合、クエリはコミットされた最新のデータを参照します。

次の表は、各テーブルタイプに対して実行できる Hudi クエリタイプを示しています。

テーブルタイプ	使用可能な Hudi クエリタイプ
書き込み時にコピー	スナップショット、増分
読み取り時にマージ	スナップショット、増分、読み取り最適化

現在、Athena はスナップショットクエリと読み取り最適化クエリをサポートしていますが、増分クエリはサポートしていません。MoR テーブルでは、すべての 読み取り最適化クエリに対して提示されるデータは圧縮されています。これにより、パフォーマンスは向上しますが、最新のデルタコミットは含まれていません。Snapshot クエリには、最も新しいデータが含まれています。しかし、いくつかの計算オーバーヘッドが生じ、これらのクエリのパフォーマンスが低下します。

テーブルとクエリタイプ間でのトレードオフの詳細については、Apache Hudi ドキュメントの「テーブルとクエリのタイプ」を参照してください。

Hudi 用語の変更: ビューはクエリに変更されました

リリースバージョン 0.5.1 以降、Apache Hudi は用語の一部を変更しました。以前、ビューと呼ばれていたものは、最近のリリースではクエリと呼ばれています。次の表は、新旧用語の変更をまとめたものです。

旧用語	新用語
CoW: 読み取り最適化ビュー MoR: リアルタイムビュー	スナップショットクエリ
増分ビュー	増分クエリ
MoR 読み取り最適化ビュー	読み取り最適化クエリ

ブートストラップ操作からのテーブル

Apache Hudi バージョン 0.6.0 以降では、ブートストラップ操作機能によって、既存の Parquet データセットのパフォーマンスが向上します。ブートストラップ操作では、データセットを書き換える代わりに、メタデータのみを生成し、データセットはそのまま残すことができます。

Athena を使用して、Amazon S3 のデータに基づいて他のテーブルと同様に、ブートストラップ操作からテーブルをクエリできます。`CREATE TABLE` ステートメントで、`LOCATION` 句で Hudi テーブルのパスを指定します。

Amazon EMR で、ブートストラップオペレーションを使用して Hudi テーブルを作成する方法の詳細については、「AWS ビッグデータブログ」の記事「New features from Apache Hudi available in Amazon EMR」(Amazon EMR で利用可能な Apache Hudi の新機能) を参照してください。

Hudi メタデータのリスト

Apache Hudi には、ファイルの一覧表示、列統計を使用したデータのスキップ、ブルームフィルターベースのインデックスなど、パフォーマンスを改善するためのインデックス機能を含むメタデータテーブルがあります。

これらの機能のうち、Athena は現在、ファイルリストインデックスのみをサポートしています。ファイルリストインデックスは、ファイルに対するパーティションのマッピングを維持するインデックスから情報を取得することにより、「ファイルのリスト」などのファイルシステム呼び出しを排除します。これにより、ファイルシステムのビューを取得するために、テーブルパスの下のすべてのパーティションを再帰的にリストする必要がなくなります。大規模なデータセットを扱う場合、このインデックス作成により、書き込みやクエリ中にファイルのリストを取得するときに発生するレイテンシーが大幅に低減されます。また、Amazon S3 `LIST` 呼び出しでのリクエスト制限のスロットリングなどのボトルネックも回避できます。

> **Note**
>
> 現時点では、Athena はデータスキップやブルームフィルターインデックス作成をサポートしていません。

Hudi メタデータテーブルの有効化

メタデータテーブルベースのファイルリストはデフォルトでは無効になっています。Hudi メタデータテーブルと関連ファイルのリスト機能を有効にするには、`hudi.metadata-listing-enabled` テーブルプロパティを `TRUE` に設定します。

例

次のALTER TABLE SET TBLPROPERTIES の例では、サンプル partition_cow テーブルでメタデータテーブルを有効にします。

```
ALTER TABLE partition_cow SET TBLPROPERTIES('hudi.metadata-listing-enabled'='TRUE')
```

考慮事項と制約事項

- Athena は増分クエリをサポートしていません。
- Athena は、Hudi データに対する CTAS または INSERT INTO をサポートしません。Athena による Hudi データセットへの書き込みのサポートをご希望の場合は、<athena-feedback@amazon.com> までフィードバックをお送りください。

  Hudi データの記述の詳細については、次のリソースを参照してください。

  - 「Amazon EMR リリースガイド」の「Hudi データセットの操作」。
  - Apache Hudi ドキュメントの「Writing Data」。
- Athena での Hudi テーブルに対する MSCK REPAIR TABLE の使用はサポートされていません。AWS Glue 以外で作成された Hudi テーブルをロードする必要がある場合は、ALTER TABLE ADD PARTITION を使用してください。
- S3 Glacier オブジェクトのスキップはサポートされていません — Apache Hudi テーブル内のオブジェクトが Amazon S3 Glacier ストレージクラスにある場合、read_restored_glacier_objects テーブルプロパティを false に設定しても効果はありません。

  例えば、次のコマンドを実行したとします。

  ```
  ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
  ```

  Iceberg および Delta Lake テーブルでは、コマンドは「サポートされていないテーブルプロパティキー: read_restored_glacier_objects」というエラーを生成します。Hudi テーブルでは、ALTER TABLE コマンドはエラーを発生しませんが、Amazon S3 Glacier オブジェクトはまだスキップされません。ALTER TABLE コマンドの後に SELECT クエリを実行すると、引き続きすべてのオブジェクトが返されます。

動画

次の動画では、Amazon Athena を使用して、Amazon S3 ベースのデータレイクにある Apache Hudi データセットの読み取り最適化クエリをする方法が紹介されています。

Query Apache Hudi datasets using Amazon Athena

Hudi テーブルの作成

このセクションでは、Hudi データのパーティション分割されたテーブルとパーティション分割されていないテーブルのための Athena の CREATE TABLE ステートメントの例を提供します。

AWS Glue で既に Hudi テーブルを作成している場合は、Athena でそれらを直接クエリできます。Athena でパーティション化された Hudi テーブルを作成するときは、クエリを実行する前に、`ALTER TABLE ADD PARTITION` を実行して Hudi データをロードする必要があります。

書き込み時コピー (CoW) テーブル作成の例

パーティション化されていない CoW テーブル

以下の例は、Athena でパーティション分割されていない CoW テーブルを作成します。

```
CREATE EXTERNAL TABLE `non_partition_cow`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/non_partition_cow/'
```

パーティション化された CoW テーブル

以下の例は、Athena でパーティション分割された CoW テーブルを作成します。

```
CREATE EXTERNAL TABLE `partition_cow`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/partition_cow/'
```

次の ALTER TABLE ADD PARTITION の例では、サンプル partition_cow テーブルに 2 つのパーティションを追加します。

```
ALTER TABLE partition_cow ADD
  PARTITION (event_type = 'one') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_cow/one/'
  PARTITION (event_type = 'two') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_cow/two/'
```

読み取り時マージ (MOR) テーブル作成の例

Hudi は、MoR のメタストアに 2 つのテーブルを作成します。1 つはスナップショットクエリ用のテーブルで、もう 1 つは読み取り最適化クエリ用のテーブルです。両方のテーブルがクエリ可能です。0.5.1 より前の Hudi バージョンでは、読み取り最適化クエリのテーブルには、テーブルの作成時に指定した名前がありました。Hudi バージョン 0.5.1 以降、デフォルトでテーブル名の末尾に _ro が付きます。スナップショットクエリのテーブルの名前は、指定した名前に _rt が付きます。

パーティション化されていない読み取り時マージ (MOR) テーブル

次の例では、読み取り最適化クエリのために、Athena でパーティション分割されていない MoR テーブルを作成します。読み取り最適化クエリでは、入力形式 HoodieParquetInputFormat を使用することに注意してください。

```
CREATE EXTERNAL TABLE `nonpartition_mor`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/nonpartition_mor/'
```

次の例では、スナップショットクエリのために、Athena でパーティション分割されていない MoR テーブルを作成します。スナップショットクエリの場合は、入力形式 HoodieParquetRealtimeInputFormat を使用します。

```
CREATE EXTERNAL TABLE `nonpartition_mor_rt`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
```

```
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/nonpartition_mor/'
```

パーティション化された読み取り時マージ (MOR) テーブル

次の例では、読み取り最適化クエリのために、Athena でパーティション分割された MoR テーブルを作成します。

```
CREATE EXTERNAL TABLE `partition_mor`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/partition_mor/'
```

次の ALTER TABLE ADD PARTITION の例では、サンプル partition_mor テーブルに 2 つのパーティションを追加します。

```
ALTER TABLE partition_mor ADD
  PARTITION (event_type = 'one') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/one/'
  PARTITION (event_type = 'two') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/two/'
```

次の例では、スナップショットクエリのために、Athena でパーティション分割された MoR テーブルを作成します。

```
CREATE EXTERNAL TABLE `partition_mor_rt`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/partition_mor/'
```

同様に、次の ALTER TABLE ADD PARTITION の例では、サンプル partition_mor_rt テーブルに 2 つのパーティションを追加します。

```
ALTER TABLE partition_mor_rt ADD
  PARTITION (event_type = 'one') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/one/'
  PARTITION (event_type = 'two') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/two/'
```

追加リソース

- AWS Glue カスタムコネクタと AWS Glue 2.0 ジョブを使用して、Athena でクエリできる Apache Hudi テーブルを作成する方法については、「AWS ビッグデータブログ」の「AWS Glue カスタムコネクタを使用した Apache Hudi テーブルへの書き込み」を参照してください。
- データレイクのデータ処理フレームワークを構築するための Apache Hudi、AWS Glue、および Amazon Athena の使用に関する記事については、AWS ビッグデータブログの「Simplify

operational data processing in data lakes using AWS Glue and Apache Hudi」を参照してください。

Apache Iceberg テーブルの使用

Athena では、Apache Iceberg テーブルに対して読み込み、タイムトラベル、書き込み、DDL の各クエリがサポートされています。Apache Iceberg テーブルでは、データに Apache Parquet 形式が使用され、メタストアに AWS Glue カタログが使用されています。

Apache Iceberg は、非常に大規模な分析データセット用のオープンテーブル形式です。Iceberg は、大量のファイルのコレクションをテーブルとして管理し、レコードレベルの挿入、更新、削除、タイムトラベルクエリなどの最新の分析データレイクオペレーションをサポートします。Iceberg の仕様では、スキーマやパーティションの進化などのテーブル進化をシームレスに行うことが可能であり、Amazon S3 での使用に最適化して設計されています。Iceberg は、同時書き込みシナリオでのデータの正確性の保証にも役立ちます。

Apache Iceberg の詳細については、https://iceberg.apache.org/ を参照してください。

考慮事項と制約事項

Iceberg テーブルに対する Athena のサポートには、次の考慮事項と制限があります。

- Iceberg バージョンのサポート – Athena は Apache Iceberg バージョン 1.4.2 をサポートします。
- AWS Glue カタログのテーブルのみ – オープンソースの Glue カタログの実装で定義されている仕様に基づく AWS Glue カタログに対して作成された Iceberg テーブルのみが Athena でサポートされています。
- AWS Glue によるテーブルロックのサポートのみ – オープンソースの Glue カタログ実装はプラグインのカスタムロックをサポートしますが、Athena は AWS Glue オプティミスティックロックのみをサポートします。Athena を使用して他のロックが実装されている Iceberg テーブルを変更すると、データが失われ、トランザクションが中断する可能性があります。
- サポートされているファイル形式 - 次の表に示すように、Athena での Iceberg ファイル形式のサポートは、Athena エンジンのバージョンによって異なります。

Athena エンジンバージョン	Parquet	ORC	Avro
2	はい	いいえ	いいえ

Athena エンジンバージョン	Parquet	ORC	Avro
3	はい	はい	はい

- Iceberg v2 テーブル – Athena は、Iceberg v2 テーブルを作成し、操作します。v1 テーブルと v2 テーブルの違いについては、Apache Iceberg ドキュメントの形式バージョンの変更を参照してください。
- タイムゾーンのない時刻型の表示 – タイムゾーンのない時刻型とタイムスタンプ型は UTC で表示されます。時刻の列のフィルター式でタイムゾーンが指定されていない場合は、UTC が使用されます。
- タイムスタンプ関連のデータの精度 – Iceberg はタイムスタンプデータ型について、マイクロ秒精度をサポートしていますが、Athena は読み込みと書き込みの両方でタイムスタンプに対してミリ秒の精度しかサポートしていません。手動圧縮オペレーション中に書き換えられるデータについて、Athena は、時間関連の列でミリ秒の精度しか保持しません。
- サポートされていないオペレーション - Iceberg テーブルに対して次の Athena オペレーションはサポートされていません。
  - ALTER TABLE SET LOCATION
- ビュー - ビューの使用 で説明されているように Athena ビューを作成する場合に `CREATE VIEW` を使用します。Iceberg ビュー仕様を使用してビューを作成することに興味がある場合は、athena-feedback@amazon.com までご連絡ください。
- AWS Lake Formation でサポートされていない TTF 管理コマンド — Lake Formation を使用して Apache Iceberg、Apache Hudi、Linux Foundation Delta Lake などのトランザクションテーブル形式 (TTF) の読み取りアクセス権限を管理できますが、Lake Formation を使用して `VACUUM`、`MERGE`、`UPDATE`、`OPTIMIZE` など、これらのテーブル形式を使用する操作の権限を管理することはできません。Lake Formation と Athena の統合の詳細については、「AWS Lake Formation 開発者ガイド」の「Amazon Athena での AWS Lake Formation の使用」を参照してください。
- ネストされたフィールドによるパーティション分割 — ネストされたフィールドによるパーティション分割はサポートされていません。*`############## NOT_SUPPORTED: Partitioning by nested field is unsupported: column_name.nested_field_name`*」(未サポート: ネストされたフィールドによるパーティション分割はサポートされていません: column_name.nested_field_name) というメッセージが表示されます。
- S3 Glacier オブジェクトのスキップはサポートされていません — Apache Iceberg テーブル内のオブジェクトが Amazon S3 Glacier ストレージクラスにある場

合、read_restored_glacier_objects テーブルプロパティを false に設定しても効果はありません。

例えば、次のコマンドを実行したとします。

```
ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
```

Iceberg および Delta Lake テーブルでは、コマンドは「サポートされていないテーブルプロパティキー: read_restored_glacier_objects」というエラーを生成します。Hudi テーブルでは、ALTER TABLE コマンドはエラーを発生しませんが、Amazon S3 Glacier オブジェクトはまだスキップされません。ALTER TABLE コマンドの後に SELECT クエリを実行すると、引き続きすべてのオブジェクトが返されます。

Athena でサポートしてほしい機能につきましては、athena-feedback@amazon.com までご意見をお寄せください。

トピック

Iceberg テーブルの作成

Athena で使用する Iceberg テーブルを作成するには、このページに記載されている CREATE TABLE ステートメントまたは AWS Glue クローラーを使用できます。

CREATE TABLE ステートメントの使用

Athena は Iceberg v2 テーブルを作成します。v1 テーブルと v2 テーブルの違いについては、Apache Iceberg ドキュメントの形式バージョンの変更を参照してください。

Athena CREATE TABLE は、データを含めずに Iceberg テーブルを作成します。テーブルで Iceberg オープンソース Glue カタログが使用されている場合は、Apache Spark などの外部システムからテーブルを直接クエリできます。外部テーブルを作成する必要はありません。

> **Warning**
>
> CREATE EXTERNAL TABLE を実行すると、「External keyword not supported for table type ICEBERG」(テーブルタイプ ICEBERG では外部キーワードはサポートされていません) というエラーメッセージが表示されます。

Athena から Iceberg テーブルを作成するには、次に示す構文の概要のように、テーブルプロパティの 'table_type' を TBLPROPERTIES 句の 'ICEBERG' に設定します。

```
CREATE TABLE
  [db_name.]table_name (col_name data_type [COMMENT col_comment] [, ...] )
  [PARTITIONED BY (col_name | transform, ... )]
  LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
  TBLPROPERTIES ( 'table_type' ='ICEBERG' [, property_name=property_value] )
```

Iceberg テーブルでクエリできるデータ型の詳細については、「Athena の Iceberg テーブルでサポートされているデータ型」を参照してください。

パーティション

パーティションを持つ Iceberg テーブルを作成するには、PARTITIONED BY 構文を使用します。パーティション化に使用される列は、最初に列宣言で指定する必要があります。PARTITIONED BY 句には、列の型を含めることはできません。CREATE TABLE 構文にパーティション変換を定義することもできます。パーティション化に複数の列を指定するには、次の例のように、列をカンマ (,) 文字で区切ります。

```
CREATE TABLE iceberg_table (id bigint, data string, category string)
  PARTITIONED BY (category, bucket(16, id))
  LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
```

```
  TBLPROPERTIES ( 'table_type' = 'ICEBERG' )
```

次の表に、使用できるパーティション変換関数を示します。

機能	説明	サポートされている型
`year(ts)`	年によるパーティション	`date`, `timestamp`
`month(ts)`	月によるパーティション	`date`, `timestamp`
`day(ts)`	日によるパーティション	`date`, `timestamp`
`hour(ts)`	時間によるパーティション	`timestamp`
`bucket(N, col)`	ハッシュ値 mod *N* のバケットによるパーティション これは、Hive テーブルのハッシュバケット作成と同じ概念です。	`int`, `long`, `decimal`, `date`, `timestamp` , `string`, `binary`
`truncate( L, col)`	*L* に切り捨てられた値によるパーティション	`int`, `long`, `decimal`, `string`

Athena は Iceberg の隠しパーティション化をサポートしています。詳細については、「Apache Iceberg ドキュメント」の「Iceberg's hidden partitioning」(Iceberg の隠しパーティション化) を参照してください。

テーブルプロパティ

このセクションでは、CREATE TABLE ステートメントの TBLPROPERTIES 句でキーと値のペアとして指定できるテーブルプロパティについて説明します。Athena では、Iceberg テーブルを作成または変更するために、テーブルプロパティで事前定義されたキーと値のペアのリストのみが使用できま

す。次の表に、指定可能なテーブルプロパティを示します。圧縮オプションの詳細については、このドキュメントの「Iceberg テーブルの最適化」を参照してください。Athena による、特定のオープンソーステーブル設定プロパティのサポートをご希望の場合は、athena-feedback@amazon.com までフィードバックをお送りください。

format

説明	ファイルデータ形式
使用できるプロパティ値	サポートされるファイル形式と圧縮の組み合わせは、Athena エンジンのバージョンによって異なります。詳細については、「ファイル形式別の Iceberg テーブル圧縮サポート」を参照してください。
デフォルト値	parquet

write_compression

説明	ファイル圧縮コーデック
使用できるプロパティ値	サポートされるファイル形式と圧縮の組み合わせは、Athena エンジンのバージョンによって異なります。詳細については、「ファイル形式別の Iceberg テーブル圧縮サポート」を参照してください。
デフォルト値	デフォルトの書き込み圧縮は Athena エンジンのバージョンによって異なります。詳細については、「ファイル形式別の Iceberg テーブル圧縮サポート」を参照してください。

optimize_rewrite_data_file_threshold

説明	データ最適化固有の設定。最適化が必要なデータファイルの数が指定されたしきい値より少ない場合、データファイルは書き換えられません。これにより、蓄積するデータファイルの数を増やしてターゲットサイズに近いファイルを生成し、不要な計算をスキップしてコストを削減できます。

使用できるプロパティ値	正数。50 未満にする必要があります。
デフォルト値	5

optimize_rewrite_delete_file_threshold

説明	データ最適化固有の設定。データファイルに関連付けられた削除ファイルの数がしきい値より少ない場合、データファイルは書き換えられません。これにより、データファイルごとに蓄積する削除ファイルの数を増やしてコストを削減できます。
使用できるプロパティ値	正数。50 未満にする必要があります。
デフォルト値	2

vacuum_min_snapshots_to_keep

説明	テーブルのメインブランチに保持するスナップショットの最小数。 この値は vacuum_max_snapshot_age_seconds プロパティよりも優先されます。残りの最小スナップショット数が vacuum_max_snapshot_age_seconds で指定した経過日数よりも古い場合、そのスナップショットは保持され、vacuum_max_snapshot_age_seconds の値は無視されます。
使用できるプロパティ値	正数。
デフォルト値	1

vacuum_max_snapshot_age_seconds

説明	メインブランチに保持するスナップショットの最大保存期間。vacuum_min_snapshots_to_keep で指定された残りの最小スナップショットが指定された経過日数よりも古い場合、この値は無視されます。このテーブル動作プロパティは、Apache Iceberg 設定の history.expire.max-snapshot-age-ms プロパティに対応しています。
使用できるプロパティ値	正数。
デフォルト値	432,000 秒 (5 日間)

vacuum_max_metadata_files_to_keep

説明	以前のメタデータファイルで、テーブルのメインブランチに保持するものの最大数。
使用できるプロパティ値	正数。
デフォルト値	100

CREATE TABLE ステートメントの例

次の例では、3 つの列を持つ Iceberg テーブルを作成します。

```
CREATE TABLE iceberg_table (
  id int,
  data string,
  category string)
PARTITIONED BY (category, bucket(16,id))
LOCATION 's3://DOC-EXAMPLE-BUCKET/iceberg-folder'
TBLPROPERTIES (
  'table_type'='ICEBERG',
  'format'='parquet',
  'write_compression'='snappy',
  'optimize_rewrite_delete_file_threshold'='10'
)
```

CREATE TABLE AS SELECT (CTAS)

この CREATE TABLE AS ステートメントを使用して Iceberg テーブルを作成する詳細については、CTAS テーブルのプロパティ セクションに特に注目して「CREATE TABLE AS」を参照してください。

AWS Glue クローラーの使用

AWS Glue クローラーを使用すると、Iceberg テーブルを AWS Glue Data Catalog に自動的に登録できます。別の Iceberg カタログから移行する場合は、AWS Glue クローラーを作成してスケジュールし、Iceberg テーブルが配置されている Amazon S3 パスを指定できます。AWS Glue クローラーがトラバースできる Amazon S3 パスの最大の階層の深さを指定できます。クローラーをスケジュールすると、AWS Glue クローラーは実行するたびにスキーマ情報を抽出し、スキーマの変更で AWS Glue Data Catalog を更新します。AWS Glue クローラーは、複数のスナップショット間でスキーマのマージをサポートし、AWS Glue Data Catalog 内の最新のメタデータファイルの場所を更新します。詳細については、「AWS Glue のデータカタログとクローラー」を参照してください。

Iceberg テーブルの管理

Athena は、Iceberg テーブルに対して次のテーブル DDL オペレーションをサポートしています。

ALTER TABLE RENAME

テーブル名を変更します。

Iceberg テーブルのテーブルメタデータが Simple Storage Service (Amazon S3) に保存されるため、基礎となるテーブル情報に影響を与えることなく、Iceberg マネージドテーブルのデータベース名とテーブル名を更新できます。

概要

```
ALTER TABLE [db_name.]table_name RENAME TO [new_db_name.]new_table_name
```

例

```
ALTER TABLE my_db.my_table RENAME TO my_db2.my_table2
```

ALTER TABLE SET PROPERTIES

プロパティを Iceberg テーブルに追加し、割り当てられた値を設定します。

Iceberg 仕様に従って、テーブルプロパティは、AWS Glue ではなく、Iceberg テーブルメタデータファイルに保存されます。Athena はカスタムテーブルプロパティを受け入れません。使用できるキーと値のペアついては、「テーブルプロパティ」セクションを参照してください。Athena による、特定のオープンソーステーブル設定プロパティのサポートをご希望の場合は、athena-feedback@amazon.com までフィードバックをお送りください。

概要

```
ALTER TABLE [db_name.]table_name SET TBLPROPERTIES ('property_name' =
 'property_value' [ , ... ])
```

例

```
ALTER TABLE iceberg_table SET TBLPROPERTIES (
  'format'='parquet',
  'write_compression'='snappy',
  'optimize_rewrite_delete_file_threshold'='10'
)
```

ALTER TABLE UNSET PROPERTIES

Iceberg テーブルから既存のプロパティを削除します。

概要

```
ALTER TABLE [db_name.]table_name UNSET TBLPROPERTIES ('property_name' [ , ... ])
```

例

```
ALTER TABLE iceberg_table UNSET TBLPROPERTIES ('write_compression')
```

DESCRIBE TABLE

テーブル情報の説明です。

概要

```
DESCRIBE [FORMATTED] [db_name.]table_name
```

FORMATTED オプションを指定すると、テーブルの場所やプロパティなどの追加情報が出力に表示されます。

例

```
DESCRIBE iceberg_table
```

DROP TABLE

Iceberg テーブルを削除します。

> **Warning**
>
> Athena では Iceberg テーブルがマネージドテーブルと見なされるため、Iceberg テーブルを削除すると、テーブル内のデータもすべて削除されます。

概要

```
DROP TABLE [IF EXISTS] [db_name.]table_name
```

例

```
DROP TABLE iceberg_table
```

SHOW CREATE TABLE

CREATE TABLE DDL ステートメントを表示します。Athena で Iceberg テーブルを再作成するために使用できます。Athena がテーブル構造を再現できない場合 (たとえば、テーブルにカスタムテーブルプロパティが指定されているため)、「UNSUPPORTED」(サポートされていません) というエラーがスローされます。

概要

```
SHOW CREATE TABLE [db_name.]table_name
```

例

```
SHOW CREATE TABLE iceberg_table
```

SHOW TABLE PROPERTIES

Iceberg テーブルのテーブルプロパティを 1 つ以上表示します。Athena でサポートされているテーブルプロパティのみが表示されます。

概要

```
SHOW TBLPROPERTIES [db_name.]table_name [('property_name')]
```

例

```
SHOW TBLPROPERTIES iceberg_table
```

Iceberg テーブルデータのクエリの実行

SELECT クエリでは、*table_name* の後に次のプロパティを使用して Iceberg テーブルメタデータをクエリできます。

- $files — テーブルの現在のデータファイルを表示します。
- $manifests — テーブルの現在のファイルマニフェストを表示します。
- $history — テーブルの履歴を表示します。
- $partitions — テーブルの現在のパーティションを表示します。
- $snapshots — テーブルのスナップショットを表示します。
- $refs — テーブルのリファレンスを表示します。

Syntax

次のステートメントは、Iceberg テーブルのファイルを一覧表示します。

```
SELECT * FROM "dbname"."tablename$files"
```

次のステートメントは、Iceberg テーブルのマニフェストを一覧表示します。

```
SELECT * FROM "dbname"."tablename$manifests"
```

次のステートメントは、Iceberg テーブルの履歴を一覧表示します。

```
SELECT * FROM "dbname"."tablename$history"
```

以下は、Iceberg テーブルのパーティションを表示する例です。

```
SELECT * FROM "dbname"."tablename$partitions"
```

以下は、Iceberg テーブルのスナップショットをリストする例です。

```
SELECT * FROM "dbname"."tablename$snapshots"
```

以下は、Iceberg テーブルのリファレンスを表示する例です。

```
SELECT * FROM "dbname"."tablename$refs"
```

Iceberg テーブルスキーマの進化

Iceberg スキーマの更新は、メタデータのみの変更です。スキーマの更新を実行しても、データファイルは変更されません。

Iceberg 形式では、次のスキーマ進化の変更がサポートされています。

- Add – 新しい列をテーブルまたはネストされた `struct` に追加します。
- Drop – 既存の列をテーブルまたはネストされた `struct` から削除します。
- Rename – 既存の列またはネストされた `struct` のフィールドの名前を変更します。
- 順序変更 - 列の順序を変更します。
- 型昇格 – 列、`struct` フィールド、`map` キー、`map` 値、または `list` 要素の型を広げます。Iceberg テーブルでは、現時点で次のケースがサポートされています。
  - 整数から大きな整数
  - 浮動小数点から倍精度浮動小数点
  - 10 進数型の精度を上げる

ALTER TABLE ADD COLUMNS

既存の Iceberg テーブルに 1 つまたは複数の列を追加します。

概要

```
ALTER TABLE [db_name.]table_name ADD COLUMNS (col_name data_type [,...])
```

例

次の例では、string 型の comment 列を Iceberg テーブルに追加しています。

```
ALTER TABLE iceberg_table ADD COLUMNS (comment string)
```

次の例では、struct 型の point 列を Iceberg テーブルに追加しています。

```
ALTER TABLE iceberg_table
ADD COLUMNS (point struct<x: double, y: double>)
```

次の例では、構造体の配列である points 列を Iceberg テーブルに追加しています。

```
ALTER TABLE iceberg_table
ADD COLUMNS (points array<struct<x: double, y: double>>)
```

ALTER TABLE DROP COLUMN

既存の Iceberg テーブルから列を削除します。

概要

```
ALTER TABLE [db_name.]table_name DROP COLUMN col_name
```

例

```
ALTER TABLE iceberg_table DROP COLUMN userid
```

ALTER TABLE CHANGE COLUMN

列の名前、型、順序、またはコメントを変更します。

> **Note**
>
> ALTER TABLE REPLACE COLUMNS はサポートされていません。REPLACE COLUMNS は、すべての列を削除してから新しい列を追加するため、Iceberg ではサポートされていません。CHANGE COLUMN は、スキーマ進化に推奨される構文です。

概要

```
ALTER TABLE [db_name.]table_name
  CHANGE [COLUMN] col_old_name col_new_name column_type
  [COMMENT col_comment] [FIRST|AFTER column_name]
```

例

```
ALTER TABLE iceberg_table CHANGE comment blog_comment string AFTER id
```

SHOW COLUMNS

テーブル内の列を表示します。

概要

```
SHOW COLUMNS (FROM|IN) [db_name.]table_name
```

例

```
SHOW COLUMNS FROM iceberg_table
```

Iceberg テーブルデータのクエリとタイムトラベルの実行

Iceberg データセットをクエリするには、次のような標準の SELECT ステートメントを使用します。クエリは Apache Iceberg format v2 spec に従い、位置と等価削除の両方の merge-on-read を実行します。

```
SELECT * FROM [db_name.]table_name [WHERE predicate]
```

クエリ時間を最適化するために、すべての述語がデータが存在する場所にプッシュダウンされます。

タイムトラベルとバージョントラベルのクエリ

各 Apache Iceberg テーブルは、それが含まれている Simple Storage Service (Amazon S3) オブジェクトのバージョン管理されたマニフェストを維持します。以前のバージョンのマニフェストを、タイムトラベルおよびバージョントラベルのクエリに使用できます。

Athena でのタイムトラベルクエリは、Amazon S3 での指定された日付と時刻における、一貫したスナップショットからの履歴データをクエリします。Athena でのバージョントラベルクエリは、Amazon S3 において指定されたスナップショット ID に関する履歴データをクエリします。

タイムトラベルクエリ

タイムトラベルクエリを実行するには、次の例のように、SELECT ステートメントのテーブル名の後に FOR TIMESTAMP AS OF *timestamp* を使用します。

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF timestamp
```

タイムトラベルで指定されるシステム時刻は、タイムスタンプか、タイムゾーン付きタイムスタンプのいずれかです。これを指定しない場合、Athena は値を UTC 時間のタイムスタンプと見なします。

次のタイムトラベルクエリの例では、指定された日時の CloudTrail データを選択します。

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF TIMESTAMP '2020-01-01 10:00:00 UTC'
```

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF (current_timestamp - interval '1' day)
```

バージョントラベルクエリ

バージョントラベルクエリを実行する (つまり、指定したバージョンで一貫したスナップショットを表示する) には、次の例のように、SELECT ステートメントのテーブル名の後に FOR VERSION AS OF *version* を使用します。

```
SELECT * FROM [db_name.]table_name FOR VERSION AS OF version
```

version パラメータは Iceberg テーブルバージョンに関連付けられている bigint スナップショット ID です。

次の例では、バージョントラベルクエリが、指定したバージョンのデータを選択しています。

```
SELECT * FROM iceberg_table FOR VERSION AS OF 949530903748831860
```

> **Note**
>
> Athena エンジンのバージョン 2 で FOR SYSTEM_TIME AS OF および FOR SYSTEM_VERSION AS OF の句は、Athena エンジンバージョン 3 の FOR TIMESTAMP AS OF および FOR VERSION AS OF の句に置き換えられました。

スナップショット ID の取得

以下の例にあるように、Iceberg が提供する Java SnapshotUtil クラスを使用して、Iceberg スナップショット ID を取得することができます。

```
import org.apache.iceberg.Table;
import org.apache.iceberg.aws.glue.GlueCatalog;
import org.apache.iceberg.catalog.TableIdentifier;
import org.apache.iceberg.util.SnapshotUtil;

import java.text.SimpleDateFormat;
import java.util.Date;

Catalog catalog = new GlueCatalog();

Map<String, String> properties = new HashMap<String, String>();
properties.put("warehouse", "s3://DOC-EXAMPLE-BUCKET/my-folder");
catalog.initialize("my_catalog", properties);

Date date = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").parse("2022/01/01 00:00:00");
long millis = date.getTime();

TableIdentifier name = TableIdentifier.of("db", "table");
Table table = catalog.loadTable(name);
long oldestSnapshotIdAfter2022 = SnapshotUtil.oldestAncestorAfter(table, millis);
```

タイムトラベルとバージョントラベルの組み合わせ

次の例のように、同じクエリでタイムトラベルとバージョントラベルの構文を使用することで、時間とバージョニングに対し異なる条件を指定できます。

```
SELECT table1.*, table2.* FROM
  [db_name.]table_name FOR TIMESTAMP AS OF (current_timestamp - interval '1' day) AS
 table1
  FULL JOIN
  [db_name.]table_name FOR VERSION AS OF 5487432386996890161 AS table2
  ON table1.ts = table2.ts
  WHERE (table1.id IS NULL OR table2.id IS NULL)
```

Iceberg テーブルによるビューの作成とクエリ

Iceberg テーブルで Athena ビューを作成してクエリするには、ビューの使用 で説明されている `CREATE VIEW` ビューを使用します。

例：

```
CREATE VIEW view1 AS SELECT * FROM iceberg_table
```

```
SELECT * FROM view1
```

Iceberg ビュー仕様を使用してビューを作成することに興味がある場合は、athena-feedback@amazon.com までご連絡ください。

Lake Formation のきめ細かなアクセス制御を使用した方法

Athena エンジンのバージョン 3 は、列レベルと行レベルのセキュリティアクセス制御を含む、Iceberg テーブルによる Lake Formation のきめ細かいアクセス制御をサポートしています。このアクセス制御は、タイムトラベルクエリやスキーマの進化を行ったテーブルを使用して機能します。詳細については、「Lake Formation のきめ細かなアクセス制御と Athena ワークグループ」を参照してください。

Athena 以外で Iceberg テーブルを作成した場合は、Apache Iceberg SDK バージョン 0.13.0 以降を使用して、Iceberg テーブルの列情報が AWS Glue Data Catalog に入力されるようにします。AWS Glue のIceberg テーブルに列情報が含まれていない場合は、Athena ALTER TABLE SET PROPERTIES ステートメントまたは最新の Iceberg SDK を使用してテーブルを修正し、AWS Glue の列情報を更新できます。

Iceberg テーブルデータの更新

Iceberg テーブルデータは、INSERT、UPDATE、および DELETE クエリを使用して、Athena で直接管理できます。データ管理トランザクションごとに新しいスナップショットが生成され、タイムトラベルを使用してクエリできます。UPDATE および DELETE ステートメントは、Iceberg 形式 v2 の行レベル position delete 仕様に従い、スナップショットを分離します。

次のコマンドを使用して、Iceberg テーブルでデータ管理オペレーションを実行します。

INSERT INTO

Iceberg テーブルにデータを挿入します。Athena Iceberg `INSERT INTO` は、現在の `INSERT INTO` が外部 Hive テーブルに対してクエリする場合と同じく、スキャンしたデータ量の分だけ課金されます。Iceberg テーブルにデータを挿入するには、次の構文を使用します。*query* は `VALUES (val1, val2, ...)` または `SELECT (col1, col2, …) FROM [db_name.]table_name`

WHERE *predicate* のいずれかです。SQL 構文とセマンティクスの詳細については、「INSERT INTO」を参照してください。

```
INSERT INTO [db_name.]table_name [(col1, col2, …)] query
```

次の例では、テーブル iceberg_table に値を挿入します。

```
INSERT INTO iceberg_table VALUES (1,'a','c1')
```

```
INSERT INTO iceberg_table (col1, col2, ...) VALUES (val1, val2, ...)
```

```
INSERT INTO iceberg_table SELECT * FROM another_table
```

DELETE

Athena Iceberg DELETE は、Iceberg 位置削除ファイルをテーブルに書き込みます。これは、読み込み時マージ削除と呼ばれます。コピーオンライト削除とは対照的に、ファイルデータを書き換えないため、読み込み時マージ削除のほうが効率的です。Athena は、Iceberg データを読み込むときに、Iceberg 位置削除ファイルをデータファイルとマージして、テーブルの最新のビューを生成します。こうした位置削除ファイルを削除するには、REWRITE DATA 圧縮アクションを実行します。DELETE オペレーションは、スキャンしたデータ量の分だけ課金されます。構文については、「DELETE」を参照してください。

次の例では、category の値が c3 の iceberg_table から行を削除します。

```
DELETE FROM iceberg_table WHERE category='c3'
```

UPDATE

Athena Iceberg UPDATE は、Iceberg 位置削除ファイルと新しく更新された行を同じトランザクション内のデータファイルとして書き込みます。UPDATE は、INSERT INTO と DELETE を組み合わせたものと考えることができます。UPDATE オペレーションは、スキャンしたデータ量の分だけ課金されます。構文については、「UPDATE」を参照してください。

次の例では、テーブル iceberg_table の指定された値を更新します。

```
UPDATE iceberg_table SET category='c2' WHERE category='c1'
```

MERGE INTO

条件付きで Iceberg テーブルに行を更新、削除、または挿入します。単一のステートメントで、更新、削除、挿入のアクションを組み合わせることができます。構文については、「MERGE INTO」を参照してください。

> **Note**
>
> MERGE INTO はトランザクションで、Athena エンジンバージョン 3 の Apache Iceberg テーブルでのみサポートされています。

次の例では、ソーステーブル s にあるテーブル t からすべての顧客を削除します。

```
MERGE INTO accounts t USING monthly_accounts_update s
ON t.customer = s.customer
WHEN MATCHED
THEN DELETE
```

次の例では、ソーステーブル s の顧客情報でターゲットテーブル t を更新します。テーブル s 内の顧客の行が一致する顧客行が t にある場合、この例ではテーブル t の購入をインクリメントします。テーブル t がテーブル s 内の顧客の行と一致しない場合、この例では、顧客の行をテーブル s からテーブル t に挿入します。

```
MERGE INTO accounts t USING monthly_accounts_update s
    ON (t.customer = s.customer)
    WHEN MATCHED
        THEN UPDATE SET purchases = s.purchases + t.purchases
    WHEN NOT MATCHED
        THEN INSERT (customer, purchases, address)
             VALUES(s.customer, s.purchases, s.address)
```

次の例では、ソーステーブル s の情報でターゲットテーブル t を条件付きで更新します。この例では、送信元アドレスが Centreville である一致するターゲット行をすべて削除します。一致するすべての行について、この例では送信元購入を追加し、ターゲットアドレスを送信元アドレスに設定します。ターゲットテーブルに一致するものがない場合、この例では送信元テーブルから行を挿入します。

```
MERGE INTO accounts t USING monthly_accounts_update s
```

```
    ON (t.customer = s.customer)
    WHEN MATCHED AND s.address = 'Centreville'
        THEN DELETE
    WHEN MATCHED
        THEN UPDATE
            SET purchases = s.purchases + t.purchases, address = s.address
    WHEN NOT MATCHED
        THEN INSERT (customer, purchases, address)
              VALUES(s.customer, s.purchases, s.address)
```

Iceberg テーブルの最適化

Iceberg テーブルにデータが蓄積するにつれて、ファイルを開くために必要な処理時間が長くなるため、クエリの効率が徐々に低下します。テーブルに delete files が含まれている場合は、追加の計算コストが発生します。Iceberg では、delete files には行レベルの削除が格納され、エンジンは削除した行をクエリ結果に適用する必要があります。

Iceberg テーブルでのクエリのパフォーマンスを最適化するために、Athena はテーブルメンテナンスコマンドとして手動圧縮をサポートしています。圧縮によって、表の内容を変更することなく、テーブルの構造レイアウトが最適化されます。

OPTIMIZE

`OPTIMIZE table REWRITE DATA` 圧縮アクションは、関連する削除ファイルのサイズと数に基づいて、データファイルをより最適化されたレイアウトに書き換えます。構文とテーブルプロパティの詳細については、「OPTIMIZE」を参照してください。

例

次の例では、削除ファイルをデータファイルにマージし、ターゲットファイルサイズに近いファイルを生成します。ここでは、`category` の値が `c1` です。

```
OPTIMIZE iceberg_table REWRITE DATA USING BIN_PACK
  WHERE category = 'c1'
```

VACUUM

`VACUUM` はスナップショットの有効期限切れと孤立ファイルの削除を行います。これらのアクションにより、メタデータのサイズが小さくなり、現在のテーブル状態にないファイルのうち、テーブル用に指定された保持期間よりも古いファイルが削除されます。構文の詳細については、「VACUUM」を参照してください。

例

次の例では、テーブルプロパティを使用して過去 3 日間のデータを保持するようにテーブル `iceberg_table` を設定し、`VACUUM` を使用して古いスナップショットを期限切れにし、孤立ファイルをテーブルから削除します。

```
ALTER TABLE iceberg_table SET TBLPROPERTIES (
  'vacuum_max_snapshot_age_seconds'='259200'
)

VACUUM iceberg_table
```

Athena の Iceberg テーブルでサポートされているデータ型

Athena は、次のデータ型が含まれている Iceberg テーブルをクエリできます。

```
binary
boolean
date
decimal
double
float
int
list
long
map
string
struct
timestamp without time zone
```

Iceberg テーブル型の詳細については、Apache のドキュメントの Iceberg の「Schemas」ページを参照してください。

次の表に、Athena のデータ型と Iceberg のテーブルデータ型の関係を示します。

Iceberg の型	Athena の型	メモ
`boolean`	`boolean`	
-	`tinyint`	Athena の Iceberg テーブルではサポートされていません。

Iceberg の型	Athena の型	メモ
-	smallint	Athena の Iceberg テーブルではサポートされていません。
int	int	Athena DML ステートメントでは、この型は INTEGER です。
long	bigint	
double	double	
float	float	
decimal(P, S)	decimal(P, S)	P は精度、S はスケールです。
-	char	Athena の Iceberg テーブルではサポートされていません。
string	string	Athena DML ステートメントでは、この型は VARCHAR です。
binary	binary	
date	date	
time	-	CREATE TABLE などの Athena Iceberg DDL ステートメントでサポートされているのは、Iceberg タイムスタンプ (タイムゾーンなし) のみですが、Athena を介してすべてのタイムスタンプ型をクエリできます。
timestamp	timestamp	
timestamp tz	timestamp tz	
list<E>	array	
map<K,V>	map	
struct<...>	struct	
fixed(L)	-	Athena では現在 fixed(L) 型はサポートされていません。

Athena でのデータ型の詳細については、「Amazon Athena のデータ型」を参照してください。

Iceberg テーブルに対するその他の Athena オペレーション

データベースレベルのオペレーション

CASCADE オプションを指定して DROP DATABASE を使用すると、Iceberg テーブルデータも削除されます。次の DDL オペレーションは、Iceberg テーブルには影響しません。

- CREATE DATABASE
- ALTER DATABASE SET DBPROPERTIES
- SHOW DATABASES
- SHOW TABLES
- SHOW VIEWS

パーティション関連のオペレーション

Iceberg テーブルでは隠しパーティション化が使用されるため、物理パーティションを直接操作する必要はありません。そのため、Athena の Iceberg テーブルでは、次のパーティション関連の DDL オペレーションがサポートされていません。

- SHOW PARTITIONS
- ALTER TABLE ADD PARTITION
- ALTER TABLE DROP PARTITION
- ALTER TABLE RENAME PARTITION

Athena での Iceberg パーティションの進化についてご意見がございましたら、athena-feedback@amazon.com までお寄せください。

Iceberg テーブルのアンロード

Iceberg テーブルは、Simple Storage Service (Amazon S3) のフォルダ内のファイルにアンロードできます。詳細については、UNLOAD を参照してください。

MSCK REPAIR

Iceberg テーブルではテーブルレイアウト情報が追跡されるため、Hive テーブルの場合のような MSCK REPAIR TABLE の実行は必要なく、サポートもされていません。

追加リソース

Apache Iceberg テーブルで Athena を使用する方法の詳細記事については、「AWS Big Data Blog」の以下の投稿を参照してください。

- Accelerate data science feature engineering on transactional data lakes using Amazon Athena with Apache Iceberg
- Amazon Athena、Amazon EMR、および AWS Glue を使用した Apache Iceberg データレイクの構築
- Perform upserts in a data lake using Amazon Athena and Apache Iceberg
- Build a transactional data lake using Apache Iceberg, AWS Glue, and cross-account data shares using AWS Lake Formation and Amazon Athena
- Use Apache Iceberg in a data lake to support incremental data processing
- Build a real-time GDPR-aligned Apache Iceberg data lake
- Automate replication of relational sources into a transactional data lake with Apache Iceberg and AWS Glue
- Interact with Apache Iceberg tables using Amazon Athena and cross account fine-grained permissions using AWS Lake Formation
- Build a serverless transactional data lake with Apache Iceberg, Amazon EMR Serverless, and Amazon Athena

Amazon Athena のセキュリティ

AWS ではクラウドセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された AWS のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、AWS と顧客の間の責任共有です。責任共有モデルでは、この責任が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されています。

- クラウドのセキュリティ — AWS は、AWS クラウドで AWS のサービス を実行するインフラストラクチャを保護する責任を負います。また AWS は、お客様が使用するサービスを安全に提供します。セキュリティの有効性は、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審査機関によって定期的にテストおよび検証されています。Athena に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」を参照してください。

- クラウド内のセキュリティ — お客様の責任は、使用する AWS のサービス に応じて異なります。また、お客様は、お客様のデータの機密性、組織の要件、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。

本書は、Amazon Athena を使用するときに責任共有モデルを適用する方法を理解するために役立ちます。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ように Athena を設定する方法を説明します。また、Athena リソースのモニタリングと保護に役立つその他の AWS のサービス を使用する方法も説明します。

トピック

Athena におけるデータ保護

AWS の責任共有モデルは、Amazon Athena 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、AWS クラウド のすべて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を担います。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。

データを保護するため、AWS アカウント 認証情報を保護し、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セットアップ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。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス 内のすべてのデフォルトセキュリティ管理に加え、AWS 暗号化ソリューションを使用します。
- 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 により AWS にアクセスするときに FIPS 140-2 検証済み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 [Name] (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、AWS CLI、または AWS SDK で Athena または他の AWS のサービス を使用する場合も同様です。タグ、または名前に使用される自由形式のテキストフィールドに入力されるデータは、請求または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。

追加のセキュリティ手順として、aws:CalledVia グローバル条件コンテキストキーを使用して、リクエストを Athena から実行されるものだけに制限できます。詳細については、「Athena での CalledVia コンテキストキーの使用」を参照してください。

複数タイプのデータの保護

Athena を使用してデータベースとテーブルを作成するときは、複数のタイプのデータが関与します。これらのデータのタイプには、Amazon S3 に保存されているソースデータ、データを検出するためにクエリまたは AWS Glue クローラを実行するときに作成されるデータベースとテーブルのメタデータ、クエリ結果データ、およびクエリ履歴が含まれます。このセクションでは、各タイプのデータについて説明し、それらのデータを保護するためのガイダンスを提供します。

- ソースデータ – データベースとテーブル用のデータは Amazon S3 に保存され、Athena はこのデータを変更しません。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 におけるデータ保護」を参照してください。ソースデータへのアクセスはユーザーが制御し、ソースデータは Amazon S3 で暗号化できます。Athena を使用して、Amazon S3 で暗号化されたデータセットに基づくテーブルを作成できます。
- データベースとテーブルのメタデータ (スキーマ) – Athena はスキーマオンリード (schema-on-read) テクノロジーを使用します。これは、Athena がクエリを実行するときに、テーブル定義が Amazon S3 のデータに適用されることを意味します。定義したスキーマは、明示的に削除しない

限り、自動的に保存されます。Athena では、DDL ステートメントを使用してデータカタログメタデータを変更できます。テーブルの定義とスキーマは、Amazon S3 に保存されている基盤となるデータに影響を及ぼすことなく削除することもできます。Athena で使用するデータベースとテーブルのメタデータは AWS Glue Data Catalog に保存されます。

AWS Identity and Access Management (IAM) を使用して、AWS Glue Data Catalog に登録されたデータベースとテーブルにきめ細かなアクセスポリシーを定義することができます。AWS Glue Data Catalog のメタデータを暗号化することもできます。メタデータを暗号化する場合、それらのデータにアクセスするためには、暗号化されたメタデータに対するアクセス許可を使用します。

- 保存されたクエリを含むクエリ結果とクエリ履歴 – クエリ結果は Amazon S3 の場所に保存されます。この場所は、グローバルに指定することも、ワークグループごとに指定することも可能です。指定しない場合、Athena はその都度デフォルトの場所を使用します。クエリ結果と保存されたクエリを格納する Amazon S3 バケットへのアクセスは、ユーザーが制御します。さらに、Amazon S3 に保存するクエリ結果の暗号化を選択することもできます。ユーザーには、Amazon S3 の場所にアクセスして、ファイルを復号化するための適切な許可が必要です。詳細については、本書の「Amazon S3 に保存された Athena のクエリ結果の暗号化」を参照してください。

  Athena では、クエリ履歴が 45 日間保持されます。クエリ履歴を表示するには、Athena API、コンソール、および AWS CLI を使用できます。クエリを 45 日より長い期間保持するには、それらを保存してください。保存されたクエリへのアクセスを保護するには、Athena でワークグループを使用して、保存されたクエリを表示する権限を持つユーザーのみにそれらへのアクセスを制限します。

トピック

保管中の暗号化

Amazon Athena では、同じリージョン内、および限られた数のリージョン間で Amazon S3 内の暗号化されたデータに対するクエリを実行できます。Amazon S3 内のクエリ結果、および AWS Glue データカタログ内のデータを暗号化することもできます。

Athena では、以下のアセットを暗号化できます。

- Amazon S3 内にあるすべてのクエリ結果。Athena は、これらを Amazon S3 の結果の場所と呼ばれる場所に保存します。Amazon S3 に保存されたクエリ結果は、基盤となるデータセットが Amazon S3 で暗号化されているかどうかにかかわらず、暗号化できます。詳細については、Amazon S3 に保存された Athena のクエリ結果の暗号化 を参照してください。
- AWS Glue データカタログのデータ。詳細については、AWS Glue データカタログの暗号化されたメタデータに対するアクセス許可 を参照してください。

> **Note**
>
> Athena を使用して暗号化されたテーブルを読み取るときに Athena が使用する暗号化オプションは、クエリ結果用のオプションではなく、テーブルデータに指定されたオプションです。クエリ結果とテーブルデータに別個の暗号化方法またはキーが設定されている場合、Athena はクエリ結果の暗号化または復号化に使用される暗号化オプションとキーを使用せずにテーブルデータを読み取ります。
> ただし、暗号化されたデータが含まれるテーブルへのデータの挿入に Athena を使用する場合、Athena はクエリ結果に指定された暗号化設定を使用して、挿入されたデータを暗号化します。例えば、クエリ結果に `CSE_KMS` 暗号化を指定すると、Athena はクエリ結果の暗号化に使用されたものと同じ AWS KMS キー ID を使用して、挿入されたテーブルデータを `CSE_KMS` で暗号化します。

トピック

- サポートされる Amazon S3 の暗号化オプション
- Amazon S3 の暗号化されたデータに対する許可
- AWS Glue データカタログの暗号化されたメタデータに対するアクセス許可
- Amazon S3 に保存された Athena のクエリ結果の暗号化
- Amazon S3 内の暗号化されたデータセットに基づくテーブルの作成

サポートされる Amazon S3 の暗号化オプション

Athena は、Amazon S3 内のデータセットとクエリ結果について、以下の暗号化オプションをサポートしています。

暗号化タイプ	説明	クロスリージョンサポート
SSE-S3	Amazon S3 マネージドキーを使用したサーバー側の暗号化 (SSE)	はい
SSE-KMS	AWS Key Management Service カスタマー管理のキーを使用したサーバー側の暗号化 (SSE)。 **Note** この暗号化タイプの場合、Athena でのテーブルの作成時に、データが暗号化されていることを指定する必要はありません。	はい
CSE-KMS	AWS KMS カスタマー管理のキーを使用したクライアント側の暗号化 (CSE)。Athena では、このオプションに `'has_encrypted_data'='true'` を指定する `TBLPROPERTIES` 句を使った `CREATE TABLE` ステートメントの使用が必要になります。詳細については、「Amazon S3 内の暗号化されたデータセットに基づくテーブルの作成」を参照してください。	いいえ

Amazon S3 での AWS KMS 暗号化の詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS Key Management Service とは」と「Amazon Simple Storage Service (Amazon S3) が AWS KMS を使用する方法」を参照してください。Athena での SSE-KMS または CSE-KMS の使用に関する詳細については、「AWS ビッグデータブログ」の「発表: Amazon Athena が暗号化されたデータのクエリのサポートを追加」を参照してください。

サポートされていないオプション

次の暗号化オプションはサポートされていません。

- SSE と顧客提供のキー (SSE-C)
- クライアント側マネージドキーを使用したクライアント側の暗号化
- 非対称キー。

Amazon S3 の暗号化オプションを比較するには、「Amazon Simple Storage Service ユーザーガイド」の「暗号化を使用したデータの保護」を参照してください。

クライアント側の暗号化のためのツール

クライアント側の暗号化には、以下の 2 つのツールを利用できることに留意してください。

- Amazon S3 暗号化クライアント – これは Amazon S3 のデータのみを暗号化し、Athena によってサポートされています。
- AWS Encryption SDK – この SDK は、AWS のどこでもデータを暗号化するために使用できますが、Athena では直接サポートされていません。

これらのツールには互換性がなく、一方のツールで暗号化されたデータをもう一方のツールで復号化することはできません。Athena が直接サポートするのは Amazon S3 暗号化クライアントのみです。SDK を使用してデータを暗号化する場合、Athena からクエリを実行することはできますが、データが暗号化されたテキストとして返されます。

AWS Encryption SDK で暗号化されたデータを Athena を使用してクエリする場合は、データをダウンロードして復号化してから、Amazon S3 暗号化クライアントを使用して再度暗号化する必要があります。

Amazon S3 の暗号化されたデータに対する許可

Amazon S3 で使用する暗号化のタイプに応じて、Athena で使用されるポリシーに「Allow」アクションとしても知られる許可を追加することが必要になる場合があります。

- SSE-S3 – 暗号化に SSE-S3 を使用する場合、Athena ユーザーのポリシーに追加の許可は必要ありません。適切な Amazon S3 の場所、および Athena のアクションに対する適切な Amazon S3 許可を持っているだけで十分です。適切な Athena および Amazon S3 の許可を可能にするポリシーの詳細については、「Amazon Athena の AWS 管理ポリシー」および「Amazon S3 へのアクセス権」を参照してください。
- AWS KMS – 暗号化に AWS KMS を使用する場合、Athena ユーザーには、Athena と Amazon S3 へのアクセス許可に加えて、特定の AWS KMS アクションの実行が許可されている必要があります。これらのアクションは、Amazon S3 内のデータの暗号化に使用される AWS KMS のカスタマーマネージドキー (CMK) のキーポリシーを編集することによって許可します。適切な AWS KMS キーポリシーにキーユーザーを追加するには、AWS KMS コンソール (https://console.aws.amazon.com/kms) を使用できます。AWS KMS キーポリシーにユーザーを追加する方法については、「AWS Key Management Service デベロッパーガイド」の「CMK の使用をキーユーザーに許可する」を参照してください。

> **Note**
>
> 高度なキーポリシーの管理者は、キーポリシーを調整することができます。`kms:Decrypt` は、Athena ユーザーが暗号化されたデータセットを使用するために許可される最小限のアクションです。暗号化されたクエリ結果を使用する場合、許可される最小限のアクションは `kms:GenerateDataKey` と `kms:Decrypt` です。

Athena でクエリする Amazon S3 内のデータセットに AWS KMS で暗号化されたオブジェクトが大量に含まれている場合、AWS KMS はクエリ結果をスロットリングする場合があります。小さいオブジェクトが大量にある場合は特に、スロットルされる可能性が高くなります。Athena は再試行リクエストを撤回しますが、それでもスロットリングエラーが発生する可能性があります。多数の暗号化されたオブジェクトを操作しているときにこの問題が発生した場合は、Simple Storage Service (Amazon S3) バケットキーを有効にして KMS への呼び出し数を減らすのも 1 つの方法です。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 バケットキーを使用した SSE-KMS のコストの削減」を参照してください。他にも、AWS KMS のサービスクォータを引き上げることができます。詳細については、「AWS Key Management Service デベロッパーガイド」の「クォータ」を参照してください。

Athena で Amazon S3 を使用する場合の許可に関するトラブルシューティング情報については、「Athena のトラブルシューティング」トピックの「アクセス許可」セクションを参照してください。

AWS Glue データカタログの暗号化されたメタデータに対するアクセス許可

AWS Glue Data Catalog のメタデータを暗号化する場合は、Athena へのアクセスに使用する ポリシーに `"kms:GenerateDataKey"`、`"kms:Decrypt"`、および `"kms:Encrypt"` アクションを追加する必要があります。詳細については、Athena から AWS Glue Data Catalog の暗号化されたメタデータにアクセスする を参照してください。

Amazon S3 に保存された Athena のクエリ結果の暗号化

Athena コンソールまたは JDBC や ODBC を使用して、クエリ結果の暗号化をセットアップします。ワークグループを使用すると、クエリ結果の暗号化を強制できます。

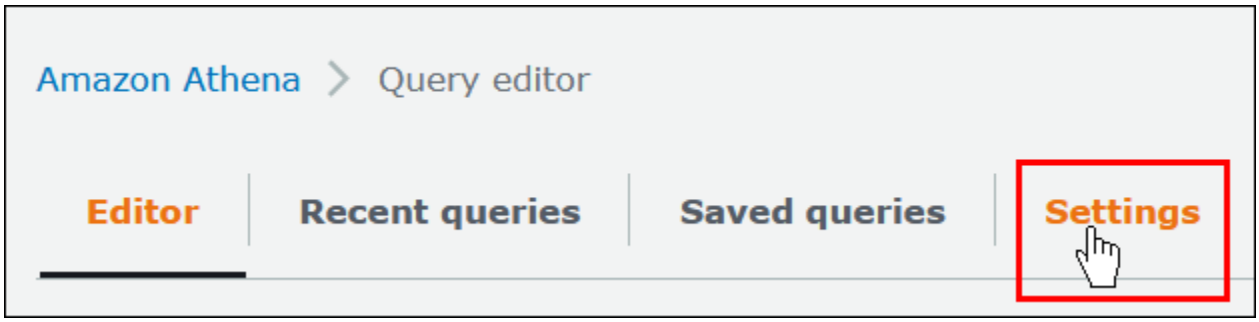
コンソールでは、クエリ結果の暗号化は次の 2 つの方法で設定できます。

- クライアント側の設定 – クエリ結果を暗号化することを指定するためのコンソールの [Settings] (設定)、または API オペレーションの使用は、クライアント側の設定の使用として知られています。クライアント側設定には、クエリ結果の場所と暗号化が含まれます。指定した場合は、ワークグループの設定によって上書きされない限り、それらの設定が使用されます。
- ワークグループ設定 — ワークグループを作成または編集して [Override client-side settings] (クライアント側の設定を上書きする) フィールドを選択すると、このワークグループで実行されるすべてのクエリでワークグループの暗号化およびクエリ結果の場所の設定が使用されます。詳細については、「ワークグループ設定がクライアント側の設定を上書きする」を参照してください。

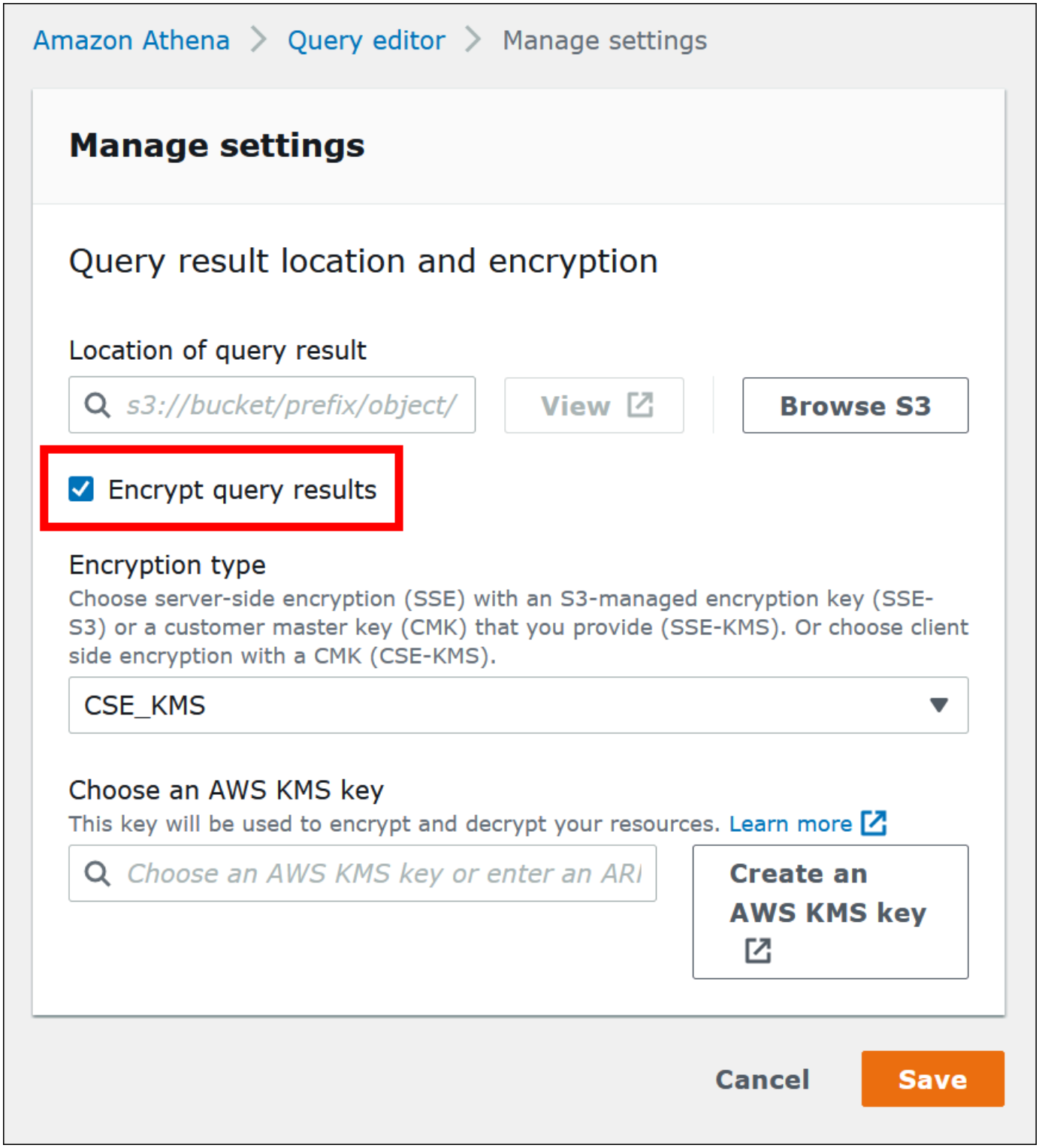
コンソールを使用して Amazon S3 に保存されているクエリ結果を暗号化する

> **Important**
>
> ワークグループで [Override client-side settings] (クライアント側の設定を上書きする) フィールドを選択している場合、ワークグループのすべてのクエリではこのワークグループ設定が使用されます。API オペレーション、JDBC および ODBCドライバーにより Athena コンソールの [Settings] (設定) タブで指定されている暗号化設定およびクエリ結果の場所は使用されません。詳細については、「ワークグループ設定がクライアント側の設定を上書きする」を参照してください。

1. Athena コンソールで [Settings] (設定) を選択します。



2. [Manage] (管理) を選択します。
3. [Location of query result] (クエリ結果の場所) で、Amazon S3 パスを入力または選択します。これは、クエリ結果が保存される Amazon S3 内の場所です。
4. [Encrypt query results] を選択します。



5. [Encryption type] で、[CSE-KMS]、[SSE-KMS]、[SSE-S3] のいずれかを選択します。これら 3 つのうち、[CSE-KMS] は最高レベルの暗号化を提供し、[SSE-S3] は最低レベルの暗号化を提供します。
6. [SSE-KMS] または [CSE-KMS] を選択した場合は、AWS KMS キーを指定します。

- [AWS KMS キーを選択] で、アカウントに既存の AWS KMS カスタマーマネージドキー (CMK) へのアクセス権がある場合は、そのエイリアスを選択するか、AWS KMS キーの ARN を入力します。
- アカウントに既存のカスタマーマネージドキー (CMK) へのアクセス権がない場合は、[Create an AWS KMS key] (AWS KMS キーを作成する) を選択し、コンソールを開きます。詳細については、「デベロッパーガイド」の「AWS Key Management Serviceキーの作成」を参照してください。

> **Note**
>
> Athena は、データの読み書きに対称キーのみをサポートします。

7. Athena コンソールに戻り、エイリアスまたは ARN で作成したキーを選択します。
8. [Save] を選択します。

JDBC または ODBC を使用する場合の Athena のクエリ結果の暗号化

JDBC または ODBC ドライバーを使用して接続する場合は、ドライバーオプションを設定して、使用する暗号化のタイプと、Amazon S3 のステージングディレクトリの場所を指定します。Athena がサポートする暗号化プロトコルのいずれかを使用してクエリ結果を暗号化するように JDBC または ODBC ドライバーを設定する場合は、「ODBC および JDBC ドライバーを使用した Amazon Athena への接続」を参照してください。

Amazon S3 内の暗号化されたデータセットに基づくテーブルの作成

テーブルを作成するときは、データセットが Amazon S3 で暗号化されていることを Athena で指定します。これは、SSE-KMS を使用するときは必要ありません。SSE-S3 と AWS KMS の両暗号化の場合、Athena はデータセットの復号とテーブルの作成の方法を判断できるため、キー情報を提供する必要はありません。

テーブルを作成するユーザーを含めて、クエリを実行するユーザーは、このトピックで前述したアクセス許可を持っている必要があります。

> **Important**
>
> EMRFS と共に Amazon EMR を使用して暗号化された Parquet ファイルをアップロードする場合は、`fs.s3n.multipart.uploads.enabled` を `false` に設定して、マルチパート

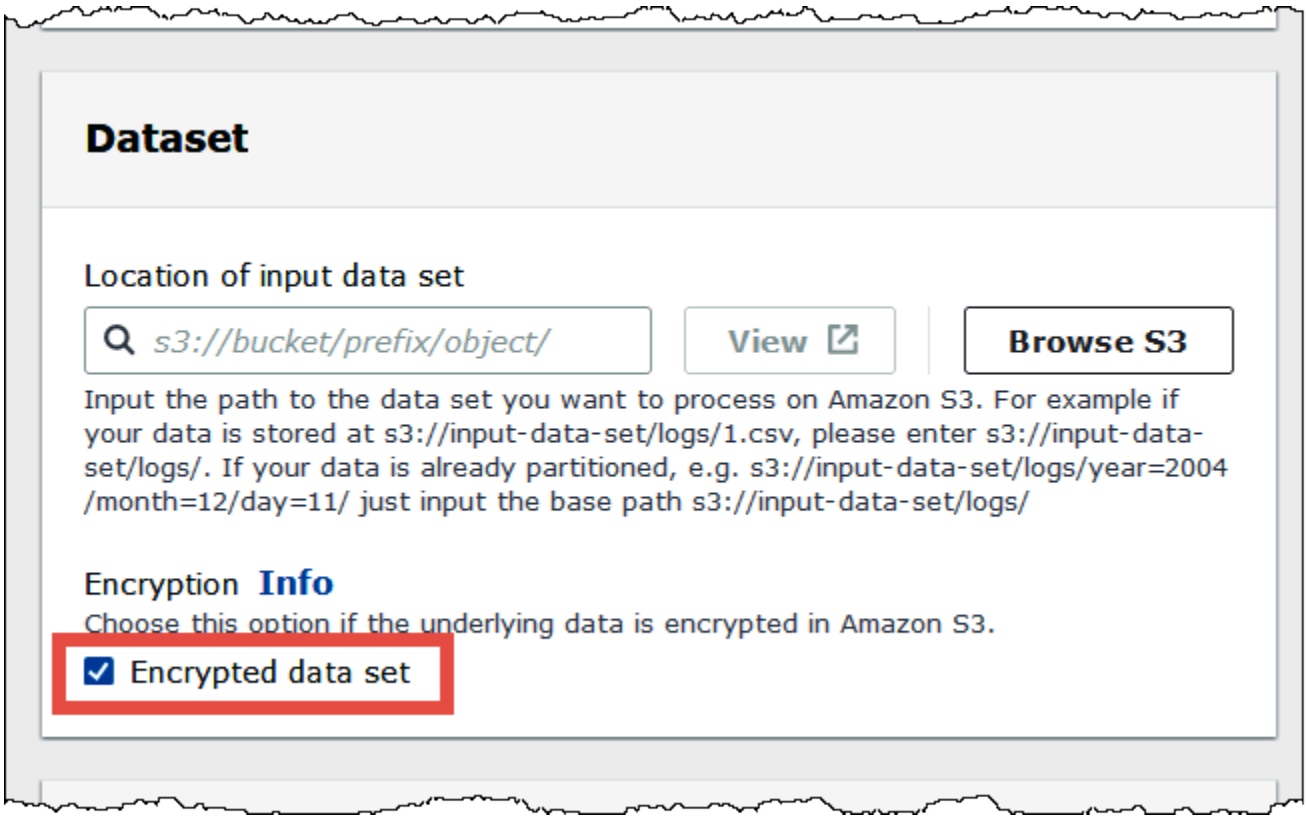
アップロードを無効にする必要があります。これを行わないと、Athena は Parquet ファイルの長さを判断できず、HIVE_CANNOT_OPEN_SPLIT エラーが発生します。詳細については、「Amazon EMR 管理ガイド」の「Amazon S3 用のマルチパートアップロードを設定する」を参照してください。

データセットが Amazon S3 で暗号化されていることを指定するには、以下のいずれかの手順を実行します。SSE-KMS を使用する場合、このステップは不要です。

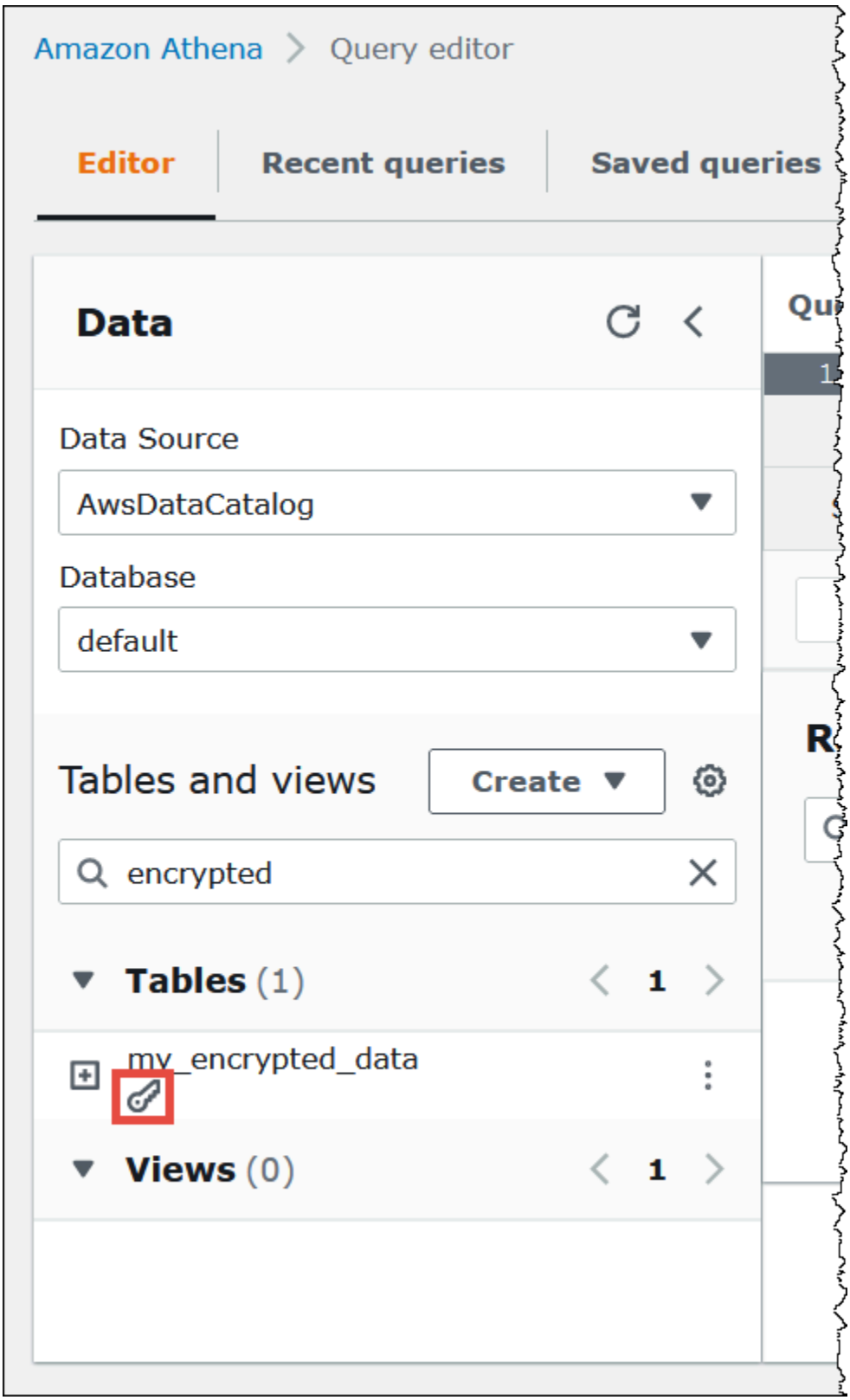
- 次の例のように、CREATE TABLE ステートメントで、`'has_encrypted_data'='true'` を指定する TBLPROPERTIES 句を使用します。

```
CREATE EXTERNAL TABLE 'my_encrypted_data' (
   `n_nationkey` int,
   `n_name` string,
   `n_regionkey` int,
   `n_comment` string)
ROW FORMAT SERDE
   'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
   'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
LOCATION
   's3://DOC-EXAMPLE-BUCKET/folder_with_my_encrypted_data/'
TBLPROPERTIES (
   'has_encrypted_data'='true')
```

- `statement.executeQuery()` を使用して CREATE TABLE ステートメントを実行するときには、前の例にあるように、JDBC ドライバーを使用して TBLPROPERTIES 値を設定します。
- Athena コンソールを使用して、フォームを使用したテーブルの作成およびテーブルの場所の指定を行う場合、[Encrypted data set] (暗号化されたデータセット) のオプションを選択します。



Athena コンソールのテーブルのリストでは、暗号化されたテーブルにはキー型のアイコンが表示されます。


Amazon Athena
Query editor
Editor
Recent queries
Saved queries
Data
Data Source
AwsDataCatalog
Database
default
Tables and views
Create
encrypted
Tables (1)
1
my_encrypted_data
Views (0)
1

転送中の暗号化

Amazon S3 での保管時のデータの暗号化に加えて、Amazon Athena は、Athena と Amazon S3 間、および Athena とそれにアクセスするカスタマーアプリケーション間での転送されるデータに Transport Layer Security (TLS) 暗号化を使用します。

Amazon S3 バケットの IAM ポリシーで `aws:SecureTransport condition` を使用して、HTTPS (TLS) 経由での暗号化された接続のみを許可するようにしてください。

JDBC または ODBC クライアントにストリーミングされるクエリ結果は、TLS を使用して暗号化されます。JDBC および ODBC ドライバーの最新バージョンとそのドキュメントについては、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

Athena フェデレーテッドデータソースコネクタの場合、TLS を使用した転送中の暗号化のサポートは個々のコネクタによって異なります。詳細については、個々のデータソースコネクタのドキュメントを参照してください。

キー管理

Amazon Athena では、Amazon S3 のデータセットと Athena のクエリ結果を暗号化する AWS Key Management Service (AWS KMS) がサポートされています。AWS KMS では Amazon S3 オブジェクトの暗号化にカスタマー管理キー (CMK) を使用しており、エンベロープ暗号化に依存しています。

AWS KMS で、以下のアクションを実行できます。

- キーを作成する
- 新しい CMK の独自のキーマテリアルをインポートする

> **Note**
>
> Athena は、データの読み書きに対称キーのみをサポートします。

詳細については、「Amazon Simple Storage Service が AWS KMS を使用する方法」と「AWS Key Management Service デベロッパーガイド」の「AWS Key Management Service とは」を参照してください。AWS によって作成および管理されるアカウントのキーを表示するには、ナビゲーションペインで [AWS マネージドキー] を選択します。

SSE-KMS で暗号化されたオブジェクトをアップロードしたり、それらへアクセスしたりする場合は、セキュリティを強化するために AWS 署名バージョン 4 を使用します。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「リクエスト認証での署名バージョンの指定」を参照してください。

Athena ワークロードが大量のデータを暗号化する場合は、Amazon S3 バケットキーを使用してコストを削減できます。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 バケットキーを使用した SSE-KMS のコストの削減」を参照してください。

インターネットトラフィックのプライバシー

トラフィックは、Athena とオンプレミスアプリケーション間、および Athena と Amazon S3 間の両方で保護されます。Athena と他のサービス (AWS Glue や AWS Key Management Service など) 間のトラフィックでは、デフォルトで HTTPS が使用されます。

- Athena とオンプレミスのクライアントおよびアプリケーション間のトラフィックについては、JDBC または ODBC クライアントにストリーミングされるクエリ結果が Transport Layer Security (TLS) を使用して暗号化されます。

  プライベートネットワークと AWS 間の接続オプションのいずれかを使用できます。

  - Site-to-Site VPN AWS VPN 接続。詳細については、「AWS Site-to-Site VPN ユーザーガイド」の「Site-to-Site VPNAWS VPN とは」を参照してください。
  - AWS Direct Connect 接続。詳細については、「AWS Direct Connect ユーザーガイド」の「AWS Direct Connect とは?」を参照してください。
- Athena と Amazon S3 バケット間のトラフィックについては、Transport Layer Security (TLS) が Athena と Amazon S3 間、および Athena とそれにアクセスするカスタマーアプリケーション間で転送されるデータを暗号化します。Amazon S3 バケットの IAM ポリシーで `aws:SecureTransport condition` を使用して、HTTPS (TLS) 経由での暗号化された接続のみを許可するようにしてください。現在、Athena では Amazon S3 バケット内のデータへのアクセスにパブリックエンドポイントを使用していますが、これによりデータがパブリックインターネットを通過するわけではありません。Athena および Amazon S3 間のすべてのトラフィックは AWS ネットワーク経由でルーティングされ、TLS を使用して暗号化されます。
- コンプライアンスプログラム — Amazon Athena は、SOC、PCI、FedRAMP など複数の AWS コンプライアンスプログラムに準拠しています。詳細については、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」を参照してください。

Athena でのアイデンティティとアクセス権の管理

Amazon Athena では、AWS Identity and Access Management (IAM) ポリシーを使用して、Athena オペレーションへのアクセスを制限します。Athena の許可の完全なリストについては、「サービス認証リファレンス」の「Amazon Athena 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。

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

Athena クエリの実行に必要な許可には、以下が含まれます。

- クエリを実行する基盤となるデータが保存されている Amazon S3 の場所。詳細については、「Amazon Simple Notification Service ユーザーガイド」の「Amazon S3 での Identity and Access Management」を参照してください。
- データベースやテーブルなど AWS Glue Data Catalog に保存しているメタデータとリソース (暗号化されたメタデータに対する追加のアクションを含む)。詳細については、「AWS Glue デベロッパーガイド」の「AWS Glue の IAM アクセス許可のセットアップ」と「AWS Glue での暗号化のセットアップ」を参照してください。
- Athena API アクション。Athena での API アクションのリストについては、「Amazon Athena API リファレンス」の「アクション」を参照してください。

以下のトピックでは、Athena の特定の分野に関する許可を詳しく説明します。

トピック

- Amazon Athena の AWS 管理ポリシー
- JDBC および ODBC 接続を介したアクセス
- Amazon S3 へのアクセス権
- Athena での Amazon S3 バケットへのクロスアカウントアクセス
- AWS Glue Data Catalog のデータベースとテーブルへのきめ細かなアクセス
- AWS Glue データカタログへのクロスアカウントアクセス
- Athena から AWS Glue Data Catalog の暗号化されたメタデータにアクセスする
- ワークグループとタグへのアクセス
- 準備済みステートメントへのアクセスを許可する

- Athena での CalledVia コンテキストキーの使用
- 外部 Hive メタストア用の Athena データコネクタへのアクセスを許可する
- 外部 Hive メタストアへの Lambda 関数アクセスを許可する
- Athena Federated Query を許可する IAM 許可ポリシーの例
- Amazon Athena ユーザー定義関数 (UDF) を許可する IAM 許可ポリシーの例
- Athena での ML へのアクセスの許可
- Athena API へのフェデレーションアクセスの有効化

Amazon Athena の AWS 管理ポリシー

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースで権限を提供できるように設計されているため、ユーザー、グループ、ロールへの権限の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権の権限を付与しない場合があることにご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。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 マネージドポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されている権限を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS のサービス を起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージド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。

Athena で管理ポリシーを使用する際の考慮事項

マネージドポリシーは使用しやすく、サービスの高度化に伴い必要になるアクションで自動的に更新されます。Athena で管理ポリシーを使用する場合は、次の点に注意してください。

- AWS Identity and Access Management (IAM) を使用して自分自身または他のユーザーに対して Amazon Athena サービスアクションを許可または拒否するには、ユーザーやグループなどのプリンシパルにアイデンティティベースのポリシーをアタッチします。
- 各アイデンティティベースのポリシーは、許可または拒否されるアクションを定義するステートメントで構成されます。ポリシーをユーザーにアタッチするための詳細情報および手順については、

「IAM ユーザーガイド」の「管理ポリシーのアタッチ」を参照してください。アクションの一覧については、Amazon Athena API リファレンスを参照してください。

- カスタマーマネージドポリシーとアイデンティティベースのインラインポリシーは、アクセス権を微調整するために、ポリシー内でより詳細な Athena アクションを指定できるようにします。`AmazonAthenaFullAccess` ポリシーを開始点として使用し、次に Amazon Athena API リファレンスに示されている特定のアクションを許可または拒否することをお勧めします。インラインポリシーの詳細については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシー」を参照してください。
- JDBC を使用して接続するプリンシパルに対しては、JDBC ドライバーの認証情報をアプリケーションに提供する必要があります。詳細については、「JDBC および ODBC 接続を介したアクセス」を参照してください。
- AWS Glue データカタログを暗号化している場合は、Athena のアイデンティティベースの IAM ポリシーで追加のアクションを指定する必要があります。詳細については、「Athena から AWS Glue Data Catalog の暗号化されたメタデータにアクセスする」を参照してください。
- ワークグループを作成して使用する場合、ポリシーにワークグループアクションへの関連するアクセス許可が含まれていることを確認してください。詳細については、「the section called “ ワークグループにアクセスするための IAM ポリシー”」および「the section called “ワークグループのポリシーの例”」を参照してください。

AWS 管理ポリシー: AmazonAthenaFullAccess

`AmazonAthenaFullAccess` マネージドポリシーは Athena への完全なアクセス権を付与します。

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「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。「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。

アクセス許可のグループ化

AmazonAthenaFullAccess ポリシーは、以下のアクセス許可セットにグループ化されます。

- **athena** – Athena リソースへのアクセスをプリンシパルに許可します。
- **glue** – AWS Glue データベース、テーブル、およびパーティションへのアクセスをプリンシパルに許可します。これは、プリンシパルが Athena で AWS Glue Data Catalog を使用するのに必要です。
- **s3** – Amazon S3 からのクエリ結果の書き込みと読み込み、Amazon S3 に存在する公開されている Athena データサンプルの読み込み、およびバケットの一覧表示をプリンシパルに許可します。これは、プリンシパルが Athena を使用して Amazon S3 を操作できるようにするために必要です。
- **sns** – Amazon SNS トピックの一覧表示と、トピック属性の取得をプリンシパルに許可します。これにより、プリンシパルは Amazon SNS トピックを Athena とともに使用して、モニタリングおよびアラートの目的で使用できるようになります。
- **cloudwatch** – CloudWatch アラームの作成、読み込み、削除をプリンシパルに許可します。詳細については、「CloudWatch のメトリクスとイベントを使用したコストの管理とクエリのモニタリング」を参照してください。
- **lakeformation** - プリンシパルが一時的な認証情報を要求して、Lake Formation に登録されているデータレイクの場所にあるデータにアクセスできるようにします。詳細については、「AWS Lake Formation デベロッパーガイド」の「Underlying Data Access Control」(基盤となるデータアクセスコントロール) を参照してください。
- **datazone**— プリンシパルで Amazon DataZone プロジェクト、ドメイン、環境を一覧表示できるようにします。Athena での DataZone の使用に関する詳細については、「Athena で Amazon DataZone を使用する」を参照してください。
- **pricing** - AWS Billing and Cost Management へのアクセスを提供します。詳細については、「AWS Billing and Cost Management API リファレンス」の「GetProducts」を参照してください。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "BaseAthenaPermissions",
            "Effect": "Allow",
            "Action": [
                "athena:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseGluePermissions",
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:StartColumnStatisticsTaskRun",
                "glue:GetColumnStatisticsTaskRun",
                "glue:GetColumnStatisticsTaskRuns"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseQueryResultsPermissions",
```

```
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-athena-query-results-*"
            ]
        },
        {
            "Sid": "BaseAthenaExamplesPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::athena-examples*"
            ]
        },
        {
            "Sid": "BaseS3BucketPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseSNSPermissions",
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics",
```

```
                "sns:GetTopicAttributes"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseCloudWatchPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:GetMetricData"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseLakeFormationPermissions",
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseDataZonePermissions",
            "Effect": "Allow",
            "Action": [
                "datazone:ListDomains",
                "datazone:ListProjects",
                "datazone:ListAccountEnvironments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BasePricingPermissions",
            "Effect": "Allow",
```

```
            "Action": [
                "pricing:GetProducts"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS 管理ポリシー: AWSQuicksightAthenaAccess

AWSQuicksightAthenaAccess は Amazon QuickSight が Athena との統合に必要とするアクションにアクセス権を付与します。AWSQuicksightAthenaAccess ポリシーは IAM ID にアタッチできます。このポリシーは、Athena で Amazon QuickSight を使用するプリンシパルのみにアタッチします。このポリシーには、非推奨となったために現行のパブリック API に含まれていない、または JDBC ドライバーおよび ODBC ドライバーのみで使用される Athena 用のアクションがいくつか含まれています。

アクセス許可のグループ化

AWSQuicksightAthenaAccess ポリシーは、以下のアクセス許可セットにグループ化されます。

- **athena** – Athena リソースに対するクエリの実行をプリンシパルに許可します。
- **glue** – AWS Glue データベース、テーブル、およびパーティションへのアクセスをプリンシパルに許可します。これは、プリンシパルが Athena で AWS Glue Data Catalog を使用するのに必要です。
- **s3** – Amazon S3 からのクエリ結果の書き込みおよび読み込みを行うことをプリンシパルに許可します。
- **lakeformation** - プリンシパルが一時的な認証情報を要求して、Lake Formation に登録されているデータレイクの場所にあるデータにアクセスできるようにします。詳細については、「AWS Lake Formation デベロッパーガイド」の「Underlying Data Access Control」(基盤となるデータアクセスコントロール) を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:GetWorkGroup",
                "athena:GetDataCatalog",
                "athena:GetDatabase",
                "athena:GetTableMetadata",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:ListTableMetadata"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition"
```

```
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-athena-query-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Athena による AWS 管理ポリシーの更新

Athena の AWS 管理ポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。

変更	説明	日付
AmazonAthenaFullAccess - 既存のポリシーへの更新	`datazone:ListDomains` 、`datazone:ListProje`	2024 年 1 月 3 日

変更	説明	日付
	cts 、datazone:ListAccountEnvironments のアクセス許可が追加され、Athena ユーザーが Amazon DataZone のドメイン、プロジェクト、環境を操作できるようになりました。詳細については、「Athena で Amazon DataZone を使用する」を参照してください。	
AmazonAthenaFullAccess - 既存のポリシーへの更新	glue:StartColumnStatisticsTaskRun 、glue:GetColumnStatisticsTaskRun 、glue:GetColumnStatisticsTaskRuns のアクセス許可が追加されたのは、Athena に AWS Glue を呼び出してコストベースのオプティマイザー機能の統計を取得する権限を与えるためです。詳細については、「コストベースオプティマイザーの使用」を参照してください。	2024 年 1 月 3 日
AmazonAthenaFullAccess - 既存のポリシーへの更新	Athena は、AWS Billing and Cost Management へのアクセスを提供するために pricing:GetProducts を追加しました。詳細については、「AWS Billing and Cost Management API リファレンス」の「GetProducts」を参照してください。	2023 年 1 月 25 日

変更	説明	日付
AmazonAthenaFullAccess - 既存のポリシーへの更新	CloudWatch メトリクス値を取得するため、Athena は `cloudwatch:GetMetricData` を追加しました。詳細については、「Amazon CloudWatch API リファレンス」の「GetMetricData」を参照してください。	2022 年 11 月 14 日
AmazonAthenaFullAccess および AWSQuicksightAthenaAccess – 既存のポリシーの更新	Athena が `s3:PutBucketPublicAccessBlock` を追加して、Athena によって作成されたバケットのパブリックアクセスのブロックを有効にしました。	2021 年 7 月 7 日
Athena が変更の追跡を開始	Athena が AWS 管理ポリシーの変更の追跡を開始しました。	2021 年 7 月 7 日

JDBC および ODBC 接続を介したアクセス

Athena や Amazon S3 バケットなどの AWS のサービス とリソースへのアクセス権を得るには、アプリケーションに JDBC または ODBC ドライバーの認証情報を提供します。JDBC または ODBC ドライバーを使用している場合は、「AWS 管理ポリシー: AWSQuicksightAthenaAccess」にリストされているすべてのアクションが IAM 許可ポリシーに含まれていることを確認してください。

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

認証方法

Athena JDBC および ODBC ドライバーは、以下の ID プロバイダーを含む SAML 2.0 ベースの認証をサポートしています。

- Active Directory Federation Services (AD FS)
- Azure Active Directory (AD)

- Okta
- PingFederate

詳細については、各ドライバーのインストールおよび設定ガイドを参照してください。これらのガイドは、JDBC および ODBC ドライバーページから PDF 形式でダウンロードできます。追加の関連情報については、以下を参照してください。

- Athena API へのフェデレーションアクセスの有効化
- Athena へのフェデレーションアクセスのための Lake Formation と Athena JDBC および ODBC ドライバーの使用
- ODBC、SAML 2.0、および Okta ID プロバイダを使用したシングルサインオンの設定

JDBC および ODBC ドライバーの最新バージョンとそのドキュメントについては、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

Amazon S3 へのアクセス権

アイデンティティベースのポリシー、バケットリソースポリシー、アクセスポイントポリシー、または上記の任意の組み合わせを使用して、Amazon S3 の場所へのアクセスを許可できます。アクターが Athena とやり取りすると、そのアクセス許可が Athena を通過して Athena がアクセスできる内容が決定されます。ユーザーが Athena でクエリを実行するには、Amazon S3 バケットにアクセスする許可が必要であることを意味します。

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

ポリシーで `aws:SourceIp` を設定すると、Athena は指定した IP アドレスを使用して Amazon S3 バケットにアクセスします。`aws:SourceVpc` 条件キーまたは `aws:SourceVpce` 条件キーに基づいて、Amazon S3 リソースへのアクセスを制限または許可することはできません。

> **Note**
>
> IAM アイデンティティセンター認証を使用する Athena ワークグループでは、信頼できる ID 伝達 ID を使用するように S3 アクセス許可を設定する必要があります。詳細については、

「Amazon Simple Storage Service ユーザーガイド」の「S3 Access Grants and directory identities」を参照してください。

Amazon S3 アクセスポイントとアクセスポイントエイリアス

Amazon S3 バケットに共有データセットがある場合、何百ものユースケースでアクセスを管理する単一のバケットポリシーを維持することが困難な場合があります。

Amazon S3 バケットアクセスポイントが、この問題の解決に役立ちます。バケットには複数のアクセスポイントがある場合があり、それぞれに異なる方法でバケットへのアクセスを制御するポリシーがあります。

作成したアクセスポイントごとに、Amazon S3 がアクセスポイントを表すエイリアスを生成します。エイリアスは Amazon S3 バケット名形式であるため、Athena での `CREATE TABLE` ステートメントの `LOCATION` 句でエイリアスを使用できます。Athena のバケットへのアクセスが、エイリアスが表すアクセスポイントのポリシーによって制御されます。

詳細については、Amazon S3 のテーブルの場所 および Amazon S3 ユーザーガイドの「アクセスポイントの使用」を参照してください。

CalledVia コンテキストキーの使用

セキュリティを強化するには、aws:CalledVia グローバル条件コンテキストキーを使用します。`aws:CalledVia` キーには、プリンシパルに代わってリクエストを実行したチェーン内の各サービスの順序付きリストが含まれます。`aws:CalledVia` コンテキストキーに Athena サービスプリンシパル名 `athena.amazonaws.com` を指定することで、リクエストを Athena から実行されたものだけに制限できます。詳細については、「Athena での CalledVia コンテキストキーの使用」を参照してください。

追加リソース

Amazon S3 のアクセス権を付与する方法の詳細と例については、以下のリソースを参照してください。

- 「Amazon S3 ユーザーガイド」の「チュートリアル例: アクセスの管理」。
- Amazon S3 バケット内のオブジェクトへのクロスアカウントアクセスを提供するにはどうすればよいですか？(AWS ナレッジセンター)
- Athena での Amazon S3 バケットへのクロスアカウントアクセス.

Athena での Amazon S3 バケットへのクロスアカウントアクセス

一般的な Amazon Athena シナリオでは、バケット所有者とは異なるアカウント内のユーザーにアクセス権を付与して、ユーザーがクエリを実行できるようにします。この場合は、アクセス権の付与にバケットポリシーを使用します。

> **Note**
>
> Athena からの AWS Glue データカタログへのクロスアカウントアクセスの詳細については、「AWS Glue データカタログへのクロスアカウントアクセス」を参照してください。

以下のバケットポリシー例は、バケット所有者によって作成されてバケット s3://DOC-EXAMPLE-BUCKET に適用されており、異なるアカウントであるアカウント 123456789123 内のすべてのユーザーにアクセス権を付与します。

```
{
   "Version": "2012-10-17",
   "Id": "MyPolicyID",
   "Statement": [
      {
          "Sid": "MyStatementSid",
          "Effect": "Allow",
          "Principal": {
             "AWS": "arn:aws:iam::123456789123:root"
          },
          "Action": [
             "s3:GetBucketLocation",
             "s3:GetObject",
             "s3:ListBucket",
             "s3:ListBucketMultipartUploads",
             "s3:ListMultipartUploadParts",
             "s3:AbortMultipartUpload"
          ],
          "Resource": [
             "arn:aws:s3:::DOC-EXAMPLE-BUCKET",
             "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
          ]
       }
    ]
 }
```

アカウントの特定のユーザーにアクセス権を付与するには、Principal キーを root から特定のユーザーを指定するキーに置き換えます。たとえば、ユーザープロファイル Dave にアクセス許可を付与するには、arn:aws:iam::123456789123:user/Dave に置き換えます。

カスタム AWS KMS キーで暗号化されたバケットへのクロスアカウントアクセス

カスタム AWS Key Management Service (AWS KMS) キーで暗号化された Amazon S3 バケットがある場合は、別の Amazon Web Services アカウントのユーザーに対してそのバケットへのアクセスを許可する必要がある場合があります。

アカウント A の AWS KMS暗号化されたバケットへのアクセスをアカウント B のユーザーに付与するには、次のアクセス許可が必要です。

- アカウント A のバケットポリシーは、アカウント B によって推定されるロールへのアクセスを許可する必要があります。
- アカウント A の AWS KMS キーポリシーは、アカウント B のユーザーによって推定されるロールへのアクセスを許可する必要があります。
- アカウント B によって推定される AWS Identity and Access Management (IAM) ロールは、アカウント A のバケットとキーの両方へのアクセスを許可する必要があります。

以下の手順では、これらのアクセス許可をそれぞれ付与する方法を示します。

アカウント A のバケットへのアクセスをアカウント B のユーザーに許可するには

- アカウント A で S3 バケットポリシーを確認し、アカウント B のアカウント ID からのアクセスを許可するステートメントがあることを確認します。

  例えば、次のバケットポリシーは、アカウント ID 111122223333 に対して s3:GetObject へのアクセスを許可します。

```
{
  "Id": "ExamplePolicy1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt1",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
```

```
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
      "Principal": {
        "AWS": [
          "111122223333"
        ]
      }
    }
  ]
}
```

アカウント A の AWS KMS キーポリシーからアカウント B のユーザーに対してアクセスを許可するには

1. アカウント A の AWS KMS キーポリシーで、アカウント B によって推定されるロールに次のアクションへのアクセス許可を付与します。

   - kms:Encrypt
   - kms:Decrypt
   - kms:ReEncrypt*
   - kms:GenerateDataKey*
   - kms:DescribeKey

   以下の例では、1 つの IAM ロールのみにキーへのアクセス権を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role_name"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

2. アカウント A から、AWS Management Console のポリシービューを使用してキーポリシーを確認します。
3. キーポリシーで、次のステートメントによってアカウント B がプリンシパルとしてリストされていることを確認します。

```
"Sid": "Allow use of the key"
```

4. "Sid": "Allow use of the key" ステートメントが存在しない場合は、以下のステップを実行します。

   a. コンソールのデフォルトビューを使用してキーポリシーを表示するように切り替えます。
   b. アカウント B のアカウント ID を、キーにアクセスできる外部アカウントとして追加します。

アカウント B によって推定される IAM ロールからアカウント A のバケットとキーへのアクセスを許可するには

1. アカウント B から、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. アカウント B のユーザーに関連付けられている IAM ロールを開きます。
3. IAM ロールに適用されている許可ポリシーのリストを確認します。
4. バケットへのアクセスを許可するポリシーが適用されていることを確認します。

   以下のステートメントの例は、バケット DOC-EXAMPLE-BUCKET での s3:GetObject オペレーションと s3:PutObject オペレーションに対するアクセス権を IAM ロールに付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt2",
      "Action": [
        "s3:GetObject",
```

```
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

5. キーへのアクセスを許可するポリシーが適用されていることを確認します。

> **Note**
>
> アカウント B によって推定される IAM ロールが管理者アクセス権を既に持っている場合は、ユーザーの IAM ポリシーからキーへのアクセス権を付与する必要はありません。

以下のステートメント例は、arn:aws:kms:us-west-2:123456789098:key/111aa2bb-333c-4d44-5555-a111bb2c33dd キーを使用するためのアクセス権を IAM ロールに付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt3",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:Encrypt",
        "kms:GenerateDataKey",
        "kms:ReEncrypt*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:kms:us-west-2:123456789098:key/111aa2bb-333c-4d44-5555-a111bb2c33dd"
    }
  ]
}
```

バケットオブジェクトへのクロスアカウントアクセス

バケットの所有アカウント (アカウント A) 以外のアカウント (アカウント C) によってアップロードされたオブジェクトには、クエリを実行するアカウント (アカウント B) への読み込みアクセスを許可する明示的なオブジェクトレベルの ACL が必要になる場合があります。この要件を回避するには、アカウント C がアカウント A のバケットにオブジェクトを配置する前に、アカウント A のロールを引き受ける必要があります。詳細については、「Amazon S3 バケット内のオブジェクトへのクロスアカウントアクセスを提供するには、どうしたらいいですか?」を参照してください。

AWS Glue Data Catalog のデータベースとテーブルへのきめ細かなアクセス

Amazon Athena で AWS Glue Data Catalog を使用している場合は、Athena で使用されるデータベースとテーブルの Data Catalog オブジェクトのリソースレベルのポリシーを定義できます。

> **Note**
>
> ここで言う「きめ細かなアクセス制御」という用語は、データベースとテーブルレベルのセキュリティを指します。列レベル、行レベル、およびセルレベルのセキュリティの詳細については、「Lake Formation でのデータフィルタリングとセルレベルのセキュリティ」を参照してください。

アイデンティティベースの IAM ポリシーにリソースレベルの許可を定義します。

> **Important**
>
> このセクションでは、アイデンティティベースの IAM ポリシーにおけるリソースレベルの許可について説明します。これらはリソースベースのポリシーとは異なります。相違点の詳細については、「IAM ユーザーガイド」の「アイデンティティベースのポリシーおよびリソースベースのポリシー」を参照してください。

これらのタスクについては、以下のトピックを参照してください。

このタスクを実行するには	以下のトピックを参照してください。
リソースへのきめ細かなアクセス権を定義す	「IAM ユーザーガイド」の「IAM ポリシーの作成」。

このタスクを実行するには	以下のトピックを参照してください。
る IAM ポリシーを作成する	
AWS Glue で使用する IAM アイデンティティベースポリシーについて説明します。	「AWS Glue デベロッパーガイド」の「アイデンティティベースのポリシー (IAM ポリシー)」。

このセクションの内容

- 制約事項
- AWS リージョン ごとのカタログとデータベースへの AWS Glue アクセス
- AWS Glue でのテーブルのパーティションとバージョン
- テーブルとデータベースのきめ細かなアクセス許可の例

制限事項

AWS Glue Data Catalog および Athena できめ細かなアクセスコントロールを使用する際は、以下の制限事項を考慮します。

- IAM アイデンティティセンターが有効な Athena ワークグループでは、IAM アイデンティティセンターの ID を使用するように Lake Formation を設定する必要があります。詳細情報については、「AWS Lake Formation デベロッパーガイドガイド」の「Integrating IAM Identity Center」を参照してください。
- データベースとテーブルにのみアクセスを制限することができます。きめ細かなアクセスコントロールはテーブルレベルで適用されるため、テーブル内の個々のパーティションへのアクセスを制限することはできません。詳細については、「AWS Glue でのテーブルのパーティションとバージョン」を参照してください。
- AWS Glue Data Catalog には `CATALOG`、`DATABASE`、`TABLE`、`FUNCTION` のリソースが含まれています。

> **Note**
>
> このリストによれば、Athena と AWS Glue Data Catalog に共通のリソースは各アカウントの TABLE、DATABASE、CATALOG です。Function は AWS Glue に固有です。Athena での削除アクションの場合、AWS Glue アクションへの許可を含める必要があります。「テーブルとデータベースのきめ細かなアクセス許可の例」を参照してください。

CATALOG は各アカウントのすべての DATABASES の祖先であり、各 DATABASE はそのすべての TABLES と FUNCTIONS の祖先であるという階層になっています。たとえば、アカウント内のカタログでデータベース db に属している table_test という名前のテーブルの場合、その祖先は db とアカウント内のカタログです。データベース db の場合、その祖先はアカウント内のカタログであり、その子孫はテーブルおよび関数です。リソースの階層構造に関する詳細については、AWS Glue デベロッパーガイドの「データカタログ内の ARN のリスト」を参照してください。

- CREATE DATABASE、CREATE TABLE、SHOW DATABASE、SHOW TABLE、または ALTER TABLE などの、リソースに対する削除以外の Athena アクションには、リソース (テーブルまたはデータベース) とそのリソースのすべての祖先でこのアクションを呼び出すための許可が必要です。例えば、テーブルの祖先は、それが属するデータベース、およびアカウントのカタログです。データベースの祖先はアカウントのカタログです。「テーブルとデータベースのきめ細かなアクセス許可の例」を参照してください。
- DROP DATABASE または DROP TABLE などの Athena での削除アクションには、データカタログ内のリソースのすべての祖先と子孫で削除アクションを呼び出すための許可も必要です。たとえば、データベースを削除するには、データベース、その祖先であるカタログ、その子孫であるすべてのテーブルとユーザー定義関数に対するアクセス許可が必要です。テーブルに子孫はありません。DROP TABLE を実行するには、テーブル、その属する先のデータベース、カタログに対して、このアクションを呼び出すためのアクセス許可が必要です。「テーブルとデータベースのきめ細かなアクセス許可の例」を参照してください。

AWS リージョン ごとのカタログとデータベースへの AWS Glue アクセス

Athena が AWS Glue と連携するには、AWS リージョン ごとに、データベースおよびアカウントの AWS Glue Data Catalog へのアクセス権限を付与するポリシーが必要です。データベースを作成するには、CreateDatabase 許可も必要です。次のポリシー例では、AWS リージョン、AWS アカウント ID、およびデータベース名を独自のものに置き換えます。

```
{
```

```
    "Sid": "DatabasePermissions",
    "Effect": "Allow",
    "Action": [
       "glue:GetDatabase",
       "glue:GetDatabases",
       "glue:CreateDatabase"
    ],
    "Resource": [
      "arn:aws:glue:us-east-1:123456789012:catalog",
      "arn:aws:glue:us-east-1:123456789012:database/default"
    ]
}
```

AWS Glue でのテーブルのパーティションとバージョン

AWS Glue では、テーブルがパーティションとバージョンを持つことができます。テーブルのバージョンとパーティションは、AWS Glue での独立したリソースとは見なされません。テーブルのバージョンとパーティションへのアクセスは、そのテーブルとその祖先リソースに対するアクセス許可を付与することで許可します。

きめ細かなアクセスコントロールを行う場合は、以下のアクセス許可を適用します。

- きめ細かなアクセスコントロールはテーブルレベルで適用されます。データベースとテーブルにのみアクセスを制限することができます。たとえば、パーティション分割されたテーブルへのアクセスを許可すると、そのアクセス権はそのテーブル内のすべてのパーティションに適用されます。テーブル内の個別パーティションへのアクセスを制限することはできません。

  > **Important**
  >
  > AWS Glue でパーティションに対してのアクションを実行するには、カタログ、データベース、テーブルレベルでパーティションアクションの許可が必要です。テーブル内のパーティションにアクセスできるだけでは十分ではありません。例えば、データベース myDB 内のテーブル myTable に対して GetPartitions を実行するには、カタログ、myDB データベース、および myTable リソースに対する glue:GetPartitions の許可を付与する必要があります。

- きめ細かなアクセスコントロールはテーブルバージョンには適用されません。パーティションと同様に、テーブルの以前のバージョンへのアクセス権は、AWS Glue でテーブルとそのテーブルの祖先に対してテーブルバージョン API へのアクセス権を介して付与されます。

AWS Glue アクションに対する許可については、「AWS Glue デベロッパーガイド」の「AWS Glue API アクセス許可: アクションとリソースのリファレンス」を参照してください。

テーブルとデータベースのきめ細かなアクセス許可の例

以下の表には、Athena のデータベースとテーブルに対するきめ細かなアクセス権を有効にするアイデンティティベースの IAM ポリシーの例がリストされています。これらの例から開始することをお勧めします。必要に応じて特定のデータベースやテーブルの個別アクションを許可または拒否するように調整します。

これらの例には、Athena と AWS Glue が連携できるようにするデータベースとカタログへのアクセス権限が含まれます。複数の AWS リージョンの場合は、データベースとカタログごとに同様のポリシーを含めます (リージョンごとに 1 行)。

例では、`example_db` データベースと `test` テーブルを独自のデータベース名とテーブル名に置き換えます。

DDL ステートメント	リソースへのアクセスを許可する IAM アクセスポリシーの例
ALTER DATABASE	`example_db` データベースのプロパティを変更できます。 `{ "Effect": "Allow", "Action": [ "glue:GetDatabase", "glue:UpdateDatabase" ], "Resource": [ "arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database / example_db " ] }`
CREATE DATABASE	`example_db` という名前のデータベースを作成できます。 `{ "Effect": "Allow", "Action": [ "glue:GetDatabase", "glue:CreateDatabase"`

DDL ステートメント	リソースへのアクセスを許可する IAM アクセスポリシーの例
	] "Resource": ["arn:aws:glue: *us-east-1* :*123456789012* :catalog", "arn:aws:glue: *us-east-1* :*123456789012* :database / *example_db* "] }

DDL ステートメント	リソースへのアクセスを許可する IAM アクセスポリシーの例
CREATE TABLE	test という名前のテーブルを example_db データベースに作成できます。

```
{
   "Sid": "DatabasePermissions",
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:GetDatabases"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db "
   ]
},
{
   "Sid": "TablePermissions",
   "Effect": "Allow",
   "Action": [
      "glue:GetTables",
      "glue:GetTable",
      "glue:GetPartitions",
      "glue:CreateTable"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db ",
     "arn:aws:glue: us-east-1 :123456789012 :table/example_d
b /test"
   ]
}
```

DDL ステートメント	リソースへのアクセスを許可する IAM アクセスポリシーの例

DROP DATABASE	すべてのテーブルを含む、example_db データベースを削除できます。
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:DeleteDatabase",
	"glue:GetTables",
	"glue:GetTable",
	"glue:DeleteTable"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database/ *example_db* ",
	"arn:aws:glue: *us-east-1* :*123456789012* :table/*example_db* /*",
	"arn:aws:glue: *us-east-1* :*123456789012* :userDefinedFunction/ *example_db* /*"
	]
	}

DDL ステートメント	リソースへのアクセスを許可する IAM アクセスポリシーの例

DROP TABLE	example_db データベースにある test という名前のパーティション分割されたテーブルを削除できます。テーブルにパーティションがない場合は、パーティションのアクションを含めません。
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:GetTable",
	"glue:DeleteTable",
	"glue:GetPartitions",
	"glue:GetPartition",
	"glue:DeletePartition"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database / *example_db* ",
	"arn:aws:glue: *us-east-1* :*123456789012* :table/*example_db* /*test*"
	]
	}

DDL ステートメント	リソースへのアクセスを許可する IAM アクセスポリシーの例

MSCK REPAIR TABLE	example_db データベースで Hive 互換パーティションを test という名前のテーブルに追加した後、カタログメタデータを更新できます。
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:CreateDatabase",
	"glue:GetTable",
	"glue:GetPartitions",
	"glue:GetPartition",
	"glue:BatchCreatePartition"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database/ *example_db* ",
	"arn:aws:glue: *us-east-1* :*123456789012* :table/*example_db* /*test*"
	]
	}

SHOW DATABASES	AWS Glue Data Catalog のすべてのデータベースをリストすることができます。
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:GetDatabases"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database/*"
	]
	}

DDL ステートメント	リソースへのアクセスを許可する IAM アクセスポリシーの例

SHOW TABLES	example_db データベースにあるすべてのテーブルをリストすることができます。
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:GetTables"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database / *example_db* ",
	"arn:aws:glue: *us-east-1* :*123456789012* :table/*example_db* /*"
	]
	}

AWS Glue データカタログへのクロスアカウントアクセス

Athena のクロスアカウント AWS Glue カタログ機能を使用して、所有するアカウント以外のアカウントから AWS Glue カタログを登録できます。AWS Glue に必要な IAM アクセス許可を設定して、カタログを Athena DataCatalog リソースとして登録したら、Athena を使用してクロスアカウントクエリを実行できます。Athena コンソールを使用して別のアカウントからカタログを登録する方法については、「別のアカウントからの AWS Glue Data Catalog の登録」を参照してください。

AWS Glue でのクロスアカウントアクセスの詳細については、「AWS Glue デベロッパーガイド」の「クロスアカウントアクセス許可の付与」を参照してください。

開始する前に

この機能は、既存の Athena DataCatalog リソース API と機能性を使用してクロスアカウントアクセスを有効にするため、開始する前に以下のリソースを読んでおくことが推奨されます。

- データソースへの接続 - Athena での AWS Glue、Hive、または Lambda データカタログソースの使用に関するトピックが含まれています。

- データカタログポリシーの例 – データカタログへのアクセスを制御するポリシーの作成方法が説明されています。
- Hive メタストアで AWS CLI を使用する - Hive メタストアで AWS CLI を使用する方法が説明されていますが、他のデータソースに該当するユースケースも含まれています。

考慮事項と制約事項

現在、Athena での AWS Glue カタログのクロスアカウントアクセスには、以下の制約事項があります。

- この機能は、Athena エンジンバージョン 2 以降がサポートされている AWS リージョン のみで使用できます。Athena エンジンバージョンの詳細については、「Athena エンジンのバージョニング」を参照してください。ワークグループのエンジンバージョンをアップグレードする方法について、「Athena エンジンバージョンの変更」を参照してください。
- 別のアカウントの AWS Glue Data Catalog をアカウントに登録するときは、特定のリージョンにある他のアカウントのデータのみにリンクされているリージョン `DataCatalog` リソースを作成します。
- 現在、クロスアカウント AWS Glue カタログを含む `CREATE VIEW` ステートメントはサポートされていません。
- AWS マネージドキーを使用して暗号化されたカタログを複数のアカウント全体でクエリすることはできません。複数のアカウント全体でクエリしたいカタログには、カスタマー管理キー (KMS_CMK) を代わりに使用してください。カスタマーマネージドキーと AWS マネージドキーの違いについては、「AWS Key Management Service デベロッパーガイド」の「カスタマーキーと AWS キー」を参照してください。

開始

次のシナリオでは、以下の例にあるように、「借用者」アカウント (666666666666) が「所有者」アカウント (999999999999) に属する AWS Glue カタログを参照する `SELECT` クエリを実行します。

```
SELECT * FROM ownerCatalog.tpch1000.customer
```

以下の手順にあるステップ 1a および 1b は、所有者と借用者の両方の観点から、所有者アカウントの AWS Glue リソースへのアクセス権を借用者アカウントに付与する方法を説明しています。この例は、データベース `tpch1000` とテーブル `customer` へのアクセス権を付与します。これらの例の名前は、要件に合わせて変更してください。

ステップ 1a: 所有者の AWS Glue リソースにアクセスするためのポリシーで借用者ロールを作成する

所有者アカウントの AWS Glue リソースにアクセスするためのポリシーで借用者アカウントロールを作成するには、AWS Identity and Access Management (IAM) コンソール、または IAM API を使用できます。以下の手順では、IAM コンソールを使用します。

所有者アカウントの AWS Glue リソースにアクセスするための借用者ロールとポリシーを作成する

1. 借用者アカウントから IAM コンソール (https://console.aws.amazon.com/iam/) にサインインします。
2. ナビゲーションペインの [アクセス管理] を展開し、[ポリシー] を選択します。
3. [Create policy] を選択します。
4. [ポリシーエディター] には [JSON] を選択します。
5. ポリシーエディタに以下のポリシーを入力してから、要件に応じて変更します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/tpch1000",
                "arn:aws:glue:us-east-1:999999999999:table/tpch1000/customer"
            ]
        }
    ]
}
```

6. [Next] を選択します。
7. [確認して作成] ページにある [ポリシー名] にポリシーの名前 (**CrossGluePolicyForBorrowerRole** など) を入力します。
8. [Create policy] を選択します。
9. ナビゲーションペインで Roles (ロール) を選択します。
10. [Create role] を選択します。

11. [信頼されたエンティティを選択] ページで [AWS アカウント] を選択してから、[次へ] を選択します。
12. [許可を追加] ページで、作成したポリシーの名前 (**CrossGluePolicyForBorrowerRole** など) を検索ボックスに入力します。
13. ポリシー名の横にあるチェックボックスにチェックを入れてから、[次へ] を選択します。
14. [Name, review, and create] (名前、レビュー、および作成) ページで、[Role name] (ロール名) にロールの名前 (例えば、**CrossGlueBorrowerRole**) を入力します。
15. [ロールの作成] を選択します。

ステップ 1b: 借用者に AWS Glue アクセス権を付与する所有者ポリシーを作成する

所有者アカウント (999999999999) から借用者のロールに AWS Glue アクセス権を付与するには、AWS Glue コンソール、または AWS Glue PutResourcePolicy API オペレーションを使用できます。以下の手順では、AWS Glue コンソールを使用します。

所有者から借用者アカウントに AWS Glue アクセス権を付与するには

1. 所有者アカウントから AWS Glue コンソール (https://console.aws.amazon.com/glue/) にサインインします。
2. ナビゲーションペインで、[データカタログ] を展開し、[カタログ設定] を選択します。
3. [Permissions] (許可) ボックスに、以下のようなポリシーを入力します。*rolename* には、ステップ 1a で借用者が作成したロール (**CrossGlueBorrowerRole** など) を入力します。権限の範囲を拡大する場合は、データベースとテーブルのリソースタイプ両方にワイルドカード文字 (*) を使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::666666666666:user/username",
                    "arn:aws:iam::666666666666:role/rolename"
                ]
            },
            "Action": "glue:*",
            "Resource": [
```

```
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/tpch1000",
                "arn:aws:glue:us-east-1:999999999999:table/tpch1000/customer"
            ]
        }
    ]
}
```

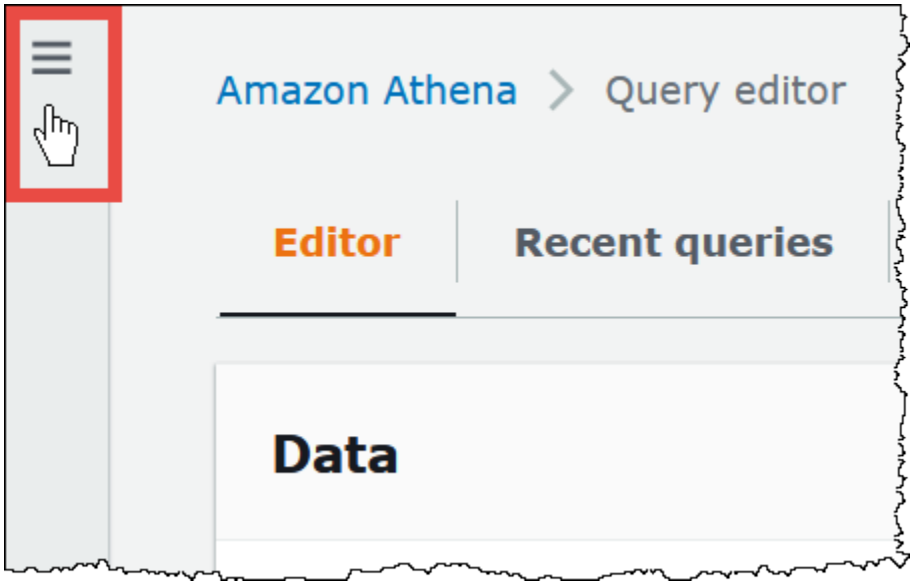
完了したら、AWS Glue API を使用してテスト用のクロスアカウントコールをいくつか実行し、許可が期待どおりに設定されているのを確認することをお勧めします。

ステップ 2: 借用者が所有者アカウントに属する AWS Glue Data Catalog を登録する

以下の手順では、Athena コンソールを使用して、所有者の Amazon Web Services アカウント内の AWS Glue Data Catalog をデータソースとして設定する方法を説明します。コンソールの代わりに API オペレーションを使用してカタログを登録する方法については、「API を使用した所有者アカウントに属する Athena データカタログの登録」(API を使用した所有者アカウントに属する Athena データカタログの登録) を参照してください。

別のアカウントに属する AWS Glue Data Catalog を登録する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。



3. [管理] を展開し、[データソース] を選択します。

4. コンソールの右上で、[Create data source] (データソースの作成) を選択します。
5. [データソースを選択] ページの [データソース] で、[S3 - AWS Glue Data Catalog] を選択し、[次へ] を選択します。
6. [Enter data source details] (データソースの詳細を入力) ページの [AWS Glue Data Catalog] (AWS Glue Data Catalog データカタログ) セクションにある [Choose an AWS Glue Data Catalog] (AWS Glue Data Catalog データカタログを選択) で、[AWS Glue Data Catalog in another account] (別のアカウントの AWS Glue Data Catalog データカタログ) を選択します。
7. [Data source details] (データソースの詳細) に、以下の情報を入力します。
   - [Data source name] (データソース名) – 他のアカウントにあるデータカタログを参照するために SQL クエリで使用する名前を入力します。
   - Description (説明) – (オプション) 他のアカウントにあるデータカタログの説明を入力します。
   - カタログ ID – データカタログが属する Amazon Web Services アカウントの 12 桁のアカウント ID を入力します。Amazon Web Services アカウント ID はカタログ ID です。
8. (オプション) [タグ] を展開し、データソースと関連付けるキーと値のペアを入力します。タグの詳細については、Athena リソースへのタグ付けを参照してください。
9. [Next] を選択します。
10. [Review and create] (確認と作成) ページで入力した情報を確認してから、[Create data source] (データソースの作成) を選択します。[Data source details] (データソースの詳細) ページに、登録したデータカタログのデータベースとタグがリストされます。
11. [Data sources] (データソース) を選択します。登録したデータカタログは、[Data source name] (データソース名) 列にリストされています。
12. データカタログに関する情報を表示または編集するには、カタログを選択してから、[Actions] (アクション)、[Edit] (編集) の順に選択します。
13. 新しいデータカタログを削除するには、カタログを選択してから、[Actions] (アクション)、[Delete] (削除する) の順に選択します。

ステップ 3: 借用者がクエリを送信する

以下の例にあるように、借用者が *catalog.database.table* 構文を使用して、カタログを参照するクエリを送信します。

```
SELECT * FROM ownerCatalog.tpch1000.customer
```

借用者は、完全修飾構文を使用する代わりに、カタログを QueryExecutionContext 経由で渡すことによって、コンテキスト的に指定することもできます。

その他の Amazon S3 許可

- 借用者アカウントが Athena クエリを使用して所有者アカウント内のテーブルに新しいデータを書き込む場合は、テーブルが所有者のアカウントに存在していても、Amazon S3 内のこのテーブルへのアクセス権が所有者に自動的に付与されることはありません。これは、別途設定されている場合を除き、借用者が Amazon S3 内のこの情報のオブジェクト所有者になっているためです。データに対するアクセス権を所有者に付与するには、追加のステップとして、オブジェクトに対する許可を適宜設定します。
- MSCK REPAIR TABLE などの特定のクロスアカウント DDL オペレーションには、Amazon S3 許可が必要です。例えば、所有者アカウントの S3 バケットにデータがある所有者アカウント内のテーブルに対して借用者アカウントがクロスアカウント `MSCK REPAIR` 操作を実行している場合、そのクエリを成功させるには、借用者が引き受けたロールに対して S3 バケットが許可を付与する必要があります。

バケットのアクセス許可の付与に関する詳細は、「Amazon Simple Storage Service ユーザーガイド」の「ACL の設定」を参照してください。

カタログの動的な使用

前提条件である登録ステップを行うことなく、クロスアカウント AWS Glue カタログに対するテストを素早く実行したいという場合があります。本書に前述されているとおり、必要な IAM 許可と Amazon S3 許可が正しく設定されている場合は、`DataCatalog` リソースオブジェクトを作成することなくクロスアカウントクエリを動的に実行することができます。

登録せずにカタログを明示的に参照するには、以下の例にある構文を使用します。

```
SELECT * FROM "glue:arn:aws:glue:us-east-1:999999999999:catalog".tpch1000.customer
```

「`glue:<arn>`」形式を使用します。*`<arn>`* は、使用する AWS Glue Data Catalog の ARN です。この例では、Athena がこの構文を使用して、アカウント 999999999999 の AWS Glue データカタログのために `DataCatalog` オブジェクトを別途作成したかのように、それを動的にポイントします。

動的カタログの使用に関する注意事項

動的カタログを使用するときは、以下の点に注意してください。

- 動的カタログの使用には、通常 Athena データカタログ API オペレーションに使用する IAM 許可が必要です。主な違いは、データカタログリソース名が `glue:*` 命名規則に従っていることです。
- カタログ ARN は、クエリが実行されているリージョンと同じリージョンに属している必要があります。
- DML クエリまたはビューで動的カタログを使用しているときは、それをエスケープされた二重引用符 (\") で囲みます。DDL クエリで動的カタログを使用しているときは、それをバックティック文字 (`) で囲みます。

API を使用した所有者アカウントに属する Athena データカタログの登録

ステップ 2 の説明に従って Athena コンソールを使用する代わりに、API オペレーションを使用して所有者アカウントに属するデータカタログを登録することができます。

Athena DataCatalog リソースの作成者には、Athena CreateDataCatalog API オペレーションを実行するための許可が必要です。要件によっては、追加の API オペレーションへのアクセス権が必要になる場合もあります。詳細については、「データカタログポリシーの例」を参照してください。

以下の `CreateDataCatalog` リクエストボディは、クロスアカウントアクセス用に AWS Glue カタログを登録します。

```
# Example CreateDataCatalog request to register a cross-account Glue catalog:
{
    "Description": "Cross-account Glue catalog",
    "Name": "ownerCatalog",
    "Parameters": {"catalog-id" : "999999999999"  # Owner's account ID
    },
    "Type": "GLUE"
}
```

以下のサンプルコードは、Java クライアントを使用して `DataCatalog` オブジェクトを作成します。

```
# Sample code to create the DataCatalog through Java client
CreateDataCatalogRequest request = new CreateDataCatalogRequest()
    .withName("ownerCatalog")
    .withType(DataCatalogType.GLUE)
    .withParameters(ImmutableMap.of("catalog-id", "999999999999"));
```

```
athenaClient.createDataCatalog(request);
```

これらのステップの後、借用者が ListDataCatalogs API オペレーションを呼び出すと、ownerCatalog が表示されます。

追加リソース

- 別のアカウントからの AWS Glue Data Catalog の登録
- 「AWS Prescriptive Guidance Patterns」ガイドの Amazon Athena を使用して、共有 AWS Glue Data Catalog へのクロスアカウントアクセスを設定します。
- AWS Big Data Blog の「Query cross-account AWS Glue Data Catalogs using Amazon Athena」
- 「AWS Glue デベロッパーガイド」の「クロスアカウントアクセス許可の付与」

Athena から AWS Glue Data Catalog の暗号化されたメタデータにアクセスする

Amazon Athena で AWS Glue Data Catalog を使用する場合は、AWS Glue コンソール、または API を使用して AWS Glue Data Catalog での暗号化を有効にできます。詳細については、「AWS Glue デベロッパーガイド」の「データカタログの暗号化」を参照してください。

AWS Glue Data Catalog が暗号化されている場合は、Athena へのアクセスに使用されるすべてのポリシーに以下のアクションを追加する必要があります。

```
{
 "Version": "2012-10-17",
 "Statement": {
 "Effect": "Allow",
     "Action": [
           "kms:GenerateDataKey",
           "kms:Decrypt",
           "kms:Encrypt"
       ],
     "Resource": "(arn of the key used to encrypt the catalog)"
   }
}
```

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

ワークグループとタグへのアクセス

ワークグループは、Athena で管理されているリソースです。したがって、ワークグループポリシーで workgroup を入力として受け取るアクションを使用する場合は、ワークグループの ARN を以下のように指定する必要があります。*workgroup-name* は実際のワークグループの名前です。

```
"Resource": [arn:aws:athena:region:AWSAcctID:workgroup/workgroup-name]
```

例えば、Amazon Web Services アカウント 123456789012 の us-west-2 リージョンにある test_workgroup という名前のワークグループの場合、以下の ARN を使用してワークグループをリソースとして指定します。

```
"Resource":["arn:aws:athena:us-east-2:123456789012:workgroup/test_workgroup"]
```

信頼できる ID 伝達 (TIP) が有効になっているワークグループにアクセスするには、Athena GetWorkGroup API アクションの応答によって返される IdentityCenterApplicationArn に IAM アイデンティティセンターのユーザーを割り当てる必要があります。

- ワークグループポリシーの一覧については、「the section called “ワークグループのポリシーの例”」を参照してください。
- ワークグループのタグベースのポリシーの一覧については、「タグベースの IAM アクセスコントロールポリシー」を参照してください。
- ワークグループ用の IAM ポリシーの作成に関する詳細については、「 ワークグループにアクセスするための IAM ポリシー」を参照してください。
- mazon Athena アクションの完全なリストについては、Amazon Athena API リファレンスの API アクション名を参照してください。
- IAM ポリシーの詳細については、「IAM ユーザーガイド」で「ビジュアルエディタでのポリシーの作成」を参照してください。

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

準備済みステートメントへのアクセスを許可する

このトピックでは、Amazon Athena の準備済みステートメント向けの IAM 許可について説明します。IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。

詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

準備済みステートメントの詳細については、「パラメータ化されたクエリの使用」を参照してください。

準備済みステートメントの作成、管理、および実行には、以下の IAM 許可が必要です。

```
athena:CreatePreparedStatement
athena:UpdatePreparedStatement
athena:GetPreparedStatement
athena:ListPreparedStatements
athena:DeletePreparedStatement
```

これらの許可は、以下の表の説明どおりに使用してください。

目的	使用する許可
PREPARE クエリを実行する	athena:StartQueryExecution athena:CreatePreparedStatement
PREPARE クエリを再度実行して既存の準備済みステートメントを更新する	athena:StartQueryExecution athena:UpdatePreparedStatement
EXECUTE クエリを実行する	athena:StartQueryExecution athena:GetPreparedStatement
DEALLOCATE PREPARE クエリを実行する	athena:StartQueryExecution athena:DeletePreparedStatement

例

以下の IAM ポリシー例は、指定されたアカウント ID とワークグループで準備済みステートメントを管理して実行するための許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "athena:StartQueryExecution",
                "athena:CreatePreparedStatement",
                "athena:UpdatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:DeletePreparedStatement",
                "athena:ListPreparedStatements"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/<workgroup-name>"
            ]
        }
    ]
}
```

Athena での CalledVia コンテキストキーの使用

プリンシパルが AWS に対してリクエストを実行すると、AWS は、リクエストを評価して承認するリクエストコンテキストにリクエスト情報を収集します。JSON ポリシーの Condition 要素を使用して、リクエストコンテキストのキーを、ポリシーで指定したキー値と比較できます。グローバル条件コンテキストキーは、aws: プレフィックスを持つ条件キーです。

aws:CalledVia コンテキストキー

aws:CalledVia グローバル条件コンテキストキーを使用して、ポリシー内のサービスと、IAM プリンシパル (ユーザーまたはロール) に代わってリクエストを実行したサービスを比較します。プリンシパルが AWS のサービス に対してリクエストを実行すると、そのサービスはプリンシパルの認証情報を使用して、他のサービスに対して後続のリクエストを実行することがあります。aws:CalledVia キーには、プリンシパルに代わってリクエストを実行したチェーン内の各サービスの順序付きリストが含まれます。

aws:CalledVia コンテキストキーにサービスプリンシパル名を指定することで、コンテキストキーを AWS のサービス 固有のものにすることができます。例えば、リクエストを Athena から実行されるものだけに制限するには、aws:CalledVia 条件キーを使用できます。Athena のポリシーで aws:CalledVia 条件キーを使用するには、以下の例にあるように、Athena サービスプリンシパル名 athena.amazonaws.com を指定します。

```
...
   "Condition": {
       "ForAnyValue:StringEquals": {
           "aws:CalledVia": "athena.amazonaws.com"
```

```
        }
    }
...
```

aws:CalledVia コンテキストキーを使用して、発信者が Athena からのリソース (Lambda 関数など) を呼び出す場合に、発信者のアクセス権がそのリソース限定であることを確実にすることができます。

> **Note**
>
> aws:CalledVia コンテキストキーは、信頼できる ID 伝達機能と互換性がありません。

Lambda 関数へのきめ細かなアクセス権のためにオプションの CalledVia コンテキストキーを追加する

Athena は、発信者がクエリに関連付けられた Lambda 関数を呼び出すには、lambda:InvokeFunction 許可を持っていることを必須としています。以下のステートメントは、Lambda 関数に対するきめ細かなアクセス許可を可能にして、ユーザーが Lambda 関数の呼び出しに Athena しか使用できないようにします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:OneAthenaLambdaFunction",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "athena.amazonaws.com"
                }
            }
        }
    ]
}
```

以下の例は、ユーザーが横串検索を実行して読み取ることができるようにするポリシーに対する、上記ステートメントの追加を示しています。これらのアクションの実行が許可されているプリンシパル

は、フェデレーティッドデータソースに関連付けられた Athena カタログを指定するクエリを実行できますが、関連付けられた Lambda 関数が Athena 経由で呼び出される場合を除き、この関数にはアクセスできません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "s3:PutObject",
                "s3:GetObject",
                "athena:StartQueryExecution",
                "s3:AbortMultipartUpload",
                "athena:StopQueryExecution",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/WorkGroupName",
                "arn:aws:s3:::MyQueryResultsBucket/*",
                "arn:aws:s3:::MyLambdaSpillBucket/MyLambdaSpillPrefix*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action":
                [
                "s3:ListBucket",
                "s3:GetBucketLocation"
                ],
            "Resource": "arn:aws:s3:::MyLambdaSpillBucket"
        },
```

```
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "athena.amazonaws.com"
                }
            }
        }
    ]
}
```

CalledVia 条件キーの詳細については、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。

外部 Hive メタストア用の Athena データコネクタへのアクセスを許可する

このトピックのアクセス許可ポリシーの例は、使用可能な必須アクションとそれらのアクションが許可されるリソースを示しています。同様のアクセス許可ポリシーを IAM ID にアタッチする前に、これらのポリシーを慎重に検討し、要件に従って変更してください。

- Example Policy to Allow an IAM Principal to Query Data Using Athena Data Connector for External Hive Metastore
- Example Policy to Allow an IAM Principal to Create an Athena Data Connector for External Hive Metastore

Example – IAM プリンシパルが、外部 Hive メタストア用の Athena データコネクタを使用してデータをクエリすることを許可する

以下のポリシーは、IAM プリンシパルに、Athena アクションへの完全なアクセス権を付与する AWS 管理ポリシー: AmazonAthenaFullAccess に加えてアタッチされるポリシーです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetLayerVersion",
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:layer:MyAthenaLambdaLayer:*"
            ]
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "arn:aws:s3:::MyLambdaSpillBucket/MyLambdaSpillLocation"
        }
    ]
}
```

許可の説明

実行可能なアクション	説明
`"s3:GetBucketLocation", "s3:GetObject", "s3:ListBucket", "s3:PutObject", "s3:ListMultipartUploadParts", "s3:AbortMultipartUpload"`	s3 アクションは、"arn:aws:s3::: *MyLambdaSpillBucket* /*MyLambdaSpillLocation* " として指定されたリソースの読み書きを許可します。ここでの *MyLambdaSpillLocation* は、呼び出される Lambda 関数の設定で指定されているスピルバケットを示します。*arn:aws:lambda:*: MyAWSAcct Id :layer:MyAthenaLambdaLayer :** リ

実行可能なアクション	説明
	ソース識別子は、デプロイ時の関数アーティファクトのサイズを削減するために、Lambda レイヤーを使用してカスタムランタイムの依存関係を作成する場合にのみ必要です。最後の位置にある * は、レイヤーバージョンのワイルドカードです。
`"lambda:GetFunction", "lambda:GetLayerVersion", "lambda:InvokeFunction"`	クエリが Resource ブロックで指定された AWS Lambda 関数を呼び出すことを許可します。例えば、*MyAthenaLambdaFunction* が呼び出される Lambda 関数の名前を指定する arn:aws:lambda:*: *MyAWSAcctId* :function: *MyAthenaLambdaFunction* などです。例に示すように、複数の関数を指定できます。

Example – IAM プリンシパルが外部 Hive メタストア用の Athena データコネクタを作成することを許可する

以下のポリシーは、IAM プリンシパルに、Athena アクションへの完全なアクセス権を付与する AWS 管理ポリシー: AmazonAthenaFullAccess に加えてアタッチされるポリシーです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "lambda:GetFunction",
                "lambda:ListFunctions",
                "lambda:GetLayerVersion",
                "lambda:InvokeFunction",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion",
                "lambda:UpdateFunctionConfiguration",
```

```
                "lambda:PutFunctionConcurrency",
                "lambda:DeleteFunctionConcurrency"
            ],
            "Resource": "arn:aws:lambda:*:111122223333:
 function: MyAthenaLambdaFunctionsPrefix*"
        }
    ]
}
```

アクセス許可の説明

クエリが Resource ブロックで指定された AWS Lambda 関数の AWS Lambda 関数を呼び出すことを許可します。例えば、*MyAthenaLambdaFunction* が呼び出される Lambda 関数の名前を指定する arn:aws:lambda:*:*MyAWSAcctId*:function:*MyAthenaLambdaFunction* などです。例に示すように、複数の関数を指定できます。

外部 Hive メタストアへの Lambda 関数アクセスを許可する

アカウントで Lambda 関数を呼び出すには、以下の許可を持つロールを作成する必要があります。

- AWSLambdaVPCAccessExecutionRole – 関数を VPC に接続する Elastic Network Interface を管理する AWS Lambda 実行ロールアクセス許可。利用可能なネットワークインターフェイスと IP アドレスが十分であることを確認します。
- AmazonAthenaFullAccess – AmazonAthenaFullAccess マネージドポリシーは、Athena への完全なアクセス権を付与します。
- Lambda 関数に Amazon S3 への書き込みを許可し、Athena に S3 からの読み取りを許可する S3 ポリシーです。

たとえば、次のポリシーは、スピル場所 s3:\\mybucket\spill のアクセス許可を定義します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutObject"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/spill"
            ]
        }
    ]
}
```

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

Lambda 関数の作成

アカウント内で Lambda 関数を作成するには、関数の開発許可、または AWSLambdaFullAccess ロールが必要です。詳細については、「AWS Lambda のアイデンティティベースの IAM ポリシー」を参照してください。

Athena は AWS Serverless Application Repository を使用して Lambda 関数を作成するため、Lambda 関数を作成するスーパーユーザーまたは管理者には、Athena の横串検索を許可する IAM ポリシーも必要です。

カタログ登録とメタデータ API オペレーション

カタログ登録 API およびメタデータ API オペレーションにアクセスするには、AmazonAthenaFullAccess マネージドポリシーを使用します。このポリシーを使用しない場合は、以下の API オペレーションを Athena ポリシーに追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListDataCatalogs",
                "athena:GetDataCatalog",
                "athena:CreateDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:GetDatabase",
                "athena:ListDatabases",
                "athena:GetTableMetadata",
                "athena:ListTableMetadata"
            ],
```

```
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Lambda のクロスリージョン呼び出し

Athena クエリを実行しているリージョン以外のリージョンで Lambda 関数を呼び出すには、Lambda 関数の完全な ARN を使用します。デフォルトで、Athena は同じリージョンで定義された Lambda 関数を呼び出します。Athena クエリを実行するリージョン以外のリージョンにある Hive メタストアにアクセスするための Lambda 関数を呼び出す必要がある場合は、Lambda 関数の完全な ARN を指定する必要があります。

例えば、欧州 (フランクフルト) リージョン `eu-central-1` のカタログ `ehms` を、米国東部 (バージニア北部) リージョンで以下の Lambda 関数を使用するように定義するとします。

```
arn:aws:lambda:us-east-1:111122223333:function:external-hms-service-new
```

この方法で完全な ARN を指定すると、Athena は `us-east-1` で `external-hms-service-new` Lambda 関数を呼び出して、`eu-central-1` から Hive メタストアデータを取得することができます。

> **Note**
>
> カタログ `ehms` は、Athena クエリを実行するリージョンと同じリージョンに登録する必要があります。

Lambda のクロスアカウント呼び出し

時折、別のアカウントから Hive メタストアへのアクセスが必要になる場合があります。例えば、Hive メタストアを実行するために、Athena クエリに使用するアカウントとは異なるアカウントから EMR クラスターを起動することがあります。異なるグループやチームが、VPC 内の異なるアカウントで Hive メタストアを実行することもできます。または、異なるグループやチームから異なる Hive メタストアのメタデータにアクセスすることもできます。

Athena は、クロスアカウントアクセスに対する AWS Lambda サポートを使用して、Hive メタストアのクロスアカウントアクセスを可能にします。

> **Note**
>
> Athena のクロスアカウントアクセスは、通常 Amazon S3 内のメタデータとデータの両方に対するクロスアカウントアクセスを意味することに注意してください。

以下のシナリオを想像してください。

- アカウント 111122223333 は、EMR クラスターで実行されている Hive メタストアにアクセスするために、Athena で us-east-1 の Lambda 関数 `external-hms-service-new` をセットアップします。
- アカウント 111122223333 は、アカウント 444455556666 に Hive メタストアデータへのアクセスを許可しようとしています。

アカウント 444455556666 に Lambda 関数 `external-hms-service-new` へのアクセス権を付与するため、アカウント 111122223333 は以下の AWS CLI `add-permission` コマンドを使用します。コマンドは、読みやすい形式にしてあります。

```
$ aws --profile perf-test lambda add-permission
      --function-name external-hms-service-new
      --region us-east-1
      --statement-id Id-ehms-invocation2
      --action "lambda:InvokeFunction"
      --principal arn:aws:iam::444455556666:user/perf1-test
{
    "Statement": "{\"Sid\":\"Id-ehms-invocation2\",
                  \"Effect\":\"Allow\",
                  \"Principal\":{\"AWS\":\"arn:aws:iam::444455556666:user/perf1-test
\"},
                  \"Action\":\"lambda:InvokeFunction\",
                  \"Resource\":\"arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new\"}"
}
```

Lambda 許可をチェックするには、以下の例にあるように `get-policy` コマンドを使用します。コマンドは、読みやすい形式にしてあります。

```
$ aws --profile perf-test lambda get-policy
      --function-name arn:aws:lambda:us-east-1:111122223333:function:external-hms-
service-new
```

```
      --region us-east-1
{
    "RevisionId": "711e93ea-9851-44c8-a09f-5f2a2829d40f",
    "Policy": "{\"Version\":\"2012-10-17\",
                \"Id\":\"default\",
                \"Statement\":[{\"Sid\":\"Id-ehms-invocation2\",
                                \"Effect\":\"Allow\",
                                \"Principal\":{\"AWS\":
\"arn:aws:iam::444455556666:user/perf1-test\"},
                                \"Action\":\"lambda:InvokeFunction\",
                                \"Resource\":\"arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new\"}]}"
}
```

許可を追加した後は、以下にあるように、カタログ ehms の定義時に us-east-1 で Lambda 関数の完全な ARN を使用できます。

```
arn:aws:lambda:us-east-1:111122223333:function:external-hms-service-new
```

クロスリージョン呼び出しの詳細については、このトピックの「Lambda のクロスリージョン呼び出し」を参照してください。

データに対するクロスアカウントアクセス許可の付与

Athena クエリを実行する前に、Amazon S3 内のデータへのクロスアカウントアクセス権を付与する必要があります。これには以下の 2 つの方法があります。

- 正規ユーザー ID で Amazon S3 バケットのアクセスコントロールリストポリシーを更新します。
- Amazon S3 バケットポリシーに対するクロスアカウントアクセス権を追加します。

例えば、アカウント 111122223333 の Amazon S3 バケットポリシーに以下のポリシーを追加して、アカウント 444455556666 が指定された Amazon S3 の場所からデータを読み取ることを許可します。

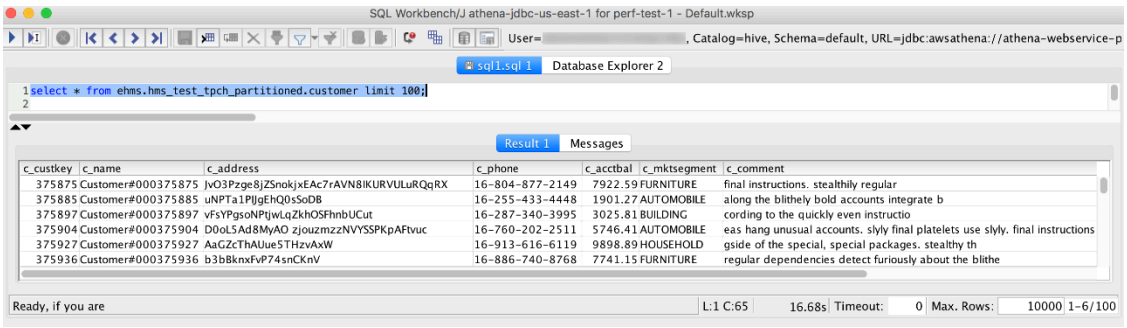
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1234567890123",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "arn:aws:iam::444455556666:user/perf1-test"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::athena-test/lambda/dataset/*"
        }
    ]
}
```

> **Note**
>
> データだけでなく、Amazon S3 のスピルの場所へのクロスアカウントアクセス権も付与する必要がある場合があります。レスポンスオブジェクトのサイズが指定されたしきい値を超えると、Lambda 関数は余分なデータをスピルの場所にスピルします。サンプルポリシーについては、このトピックの最初を参照してください。

現在の例では、444455556666, にクロスアカウントアクセスが許可された後、444455556666 が独自の account のカタログ ehms を使用して、アカウント 111122223333 で定義されたテーブルをクエリできます。

次の例では、SQLワークベンチプロファイル perf-test-1 は、アカウント 444455556666 用です。このクエリは、カタログ ehms を使用して、アカウント 111122223333 の Hive メタストアと Amazon S3 データにアクセスします。



Athena Federated Query を許可する IAM 許可ポリシーの例

このトピックのアクセス許可ポリシーの例は、使用可能な必須アクションとそれらのアクションが許可されるリソースを示しています。これらのポリシーを IAM ID にアタッチする前に、これらのポリシーを慎重に検討し、要件に従って変更してください。

IAM アイデンティティへのポリシーのアタッチに関する詳細については、「IAM ユーザーガイド」の「IAM ID のアクセス許可の追加および削除」を参照してください。

- Example policy to allow an IAM principal to run and return results using Athena Federated Query
- Example Policy to Allow an IAM Principal to Create a Data Source Connector

Example – IAM プリンシパルが Athena Federated Query を使用し、実行して結果を返すことを許可する

以下のアイデンティティベースの許可ポリシーは、ユーザーまたは他の IAM プリンシパルが Athena の横串検索を使用するために必要なアクションを許可します。これらのアクションの実行が許可されているプリンシパルは、フェデレーティッドデータソースに関連付けられた Athena カタログを指定するクエリを実行できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Athena",
            "Effect": "Allow",
            "Action": [
                "athena:GetDataCatalog",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "athena:GetWorkGroup",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/WorkgroupName",
                "arn:aws:athena:aws_region:111122223333:datacatalog/DataCatalogName"
            ]
        },
        {
            "Sid": "ListAthenaWorkGroups",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        },
        {
            "Sid": "Lambda",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:OneAthenaLambdaFunction",
```

```
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction"
            ]
        },
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListMultipartUploadParts",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::MyLambdaSpillBucket",
                "arn:aws:s3:::MyLambdaSpillBucket/*",
                "arn:aws:s3:::MyQueryResultsBucket",
                "arn:aws:s3:::MyQueryResultsBucket/*"
            ]
        }
    ]
}
```

許可の説明

実行可能なアクション	説明
`"athena:GetQueryExecution", "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StartQueryExecution", "athena:StopQueryExecution"`	横串検索の実行に必要な Athena の許可です。
`"athena:GetDataCatalog", "athena:GetQueryExecution," "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StartQueryExecution", "athena:StopQueryExecution"`	フェデレーティッドビュークエリの実行に必要な Athena の権限です。ビューには `GetDataCatalog` アクションが必要です。

実行可能なアクション	説明
`"lambda:InvokeFunction"`	クエリが Resource ブロックで指定された AWS Lambda 関数の AWS Lambda 関数を呼び出すことを許可します。例えば、*MyAthenaLambdaFunction* が呼び出される Lambda 関数の名前を指定する arn:aws:lambda:*: *MyAWSAcctId* :function: *MyAthenaLambdaFunction* などです。例にあるように、複数の関数を指定できます。

実行可能なアクション	説明

`"s3:AbortMultipartUpload",`
`"s3:GetBucketLocation",`
`"s3:GetObject",`
`"s3:ListBucket",`
`"s3:ListMultipartUploadParts",`
`"s3:PutObject"`
	StartQueryExecution を実行する IAM プリンシパルのクエリ出力バケットにアクセスするには、s3:ListBucket 許可と s3:GetBucketLocation 許可が必要です。
	
	s3:PutObject 、s3:ListMultipartUploadParts 、および s3:AbortMultipartUpload は、arn:aws:s3::: *MyQueryResultsBucket* /* リソース識別子 (*MyQueryResultsBucket* は Athena クエリ結果バケット) が指定するクエリ結果バケットのすべてのサブフォルダへのクエリ結果の書き込みを許可します。詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。
	
	s3:GetObject は、arn:aws:s3::: *MyQueryResultsBucket* として指定されたリソースのクエリ結果とクエリ履歴の読み取りを許可します。ここでの *MyQueryResultsBucket* は、Athena クエリ結果バケットです。
	
	s3:GetObject も、"arn:aws:s3::: *MyLambdaSpillBucket* */MyLambdaSpillPrefix* *" として指定されたリソースからの読み取りを許可します。ここでの *MyLambdaSpillPrefix* は、呼び出される Lambda 関数の設定で指定されます。

Example – IAM プリンシパルにデータソースコネクタの作成を許可する

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:ListVersionsByFunction",
                "iam:CreateRole",
                "lambda:GetFunctionConfiguration",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "lambda:PutFunctionConcurrency",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "lambda:ListTags",
                "iam:ListAttachedRolePolicies",
                "iam:DeleteRolePolicy",
                "lambda:DeleteFunction",
                "lambda:GetAlias",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetPolicy",
                "lambda:InvokeFunction",
                "lambda:GetFunction",
                "lambda:ListAliases",
                "lambda:UpdateFunctionConfiguration",
                "iam:DeleteRole",
                "lambda:UpdateFunctionCode",
                "s3:GetObject",
                "lambda:AddPermission",
                "iam:UpdateRole",
                "lambda:DeleteFunctionConcurrency",
                "lambda:RemovePermission",
                "iam:GetRolePolicy",
                "lambda:GetPolicy"
            ],
            "Resource": [

  "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunctionsPrefix*",
                "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
                "arn:aws:iam::*:role/RoleName",
                "arn:aws:iam::111122223333:policy/*"
            ]
        },
```

```
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateUploadBucket",
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:ListExports",
                "cloudformation:ListStacks",
                "cloudformation:ListImports",
                "lambda:ListFunctions",
                "iam:ListRoles",
                "lambda:GetAccountSettings",
                "ec2:DescribeSecurityGroups",
                "cloudformation:EstimateTemplateCost",
                "ec2:DescribeVpcs",
                "lambda:ListEventSourceMappings",
                "cloudformation:DescribeAccountLimits",
                "ec2:DescribeSubnets",
                "cloudformation:CreateStackSet",
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "cloudformation:*",
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:111122223333:stack/
serverlessrepo-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:*:transform/Serverless-*",
                "arn:aws:cloudformation:*:111122223333:stackset/aws-serverless-
repository-MyCFStackPrefix*:*",
                "arn:aws:cloudformation:*:111122223333:stackset/
serverlessrepo-MyCFStackPrefix*:*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "serverlessrepo:*",
            "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
```

```
        }
    ]
}
```

許可の説明

実行可能なアクション	説明

`"lambda:CreateFunction",`
`"lambda:ListVersionsByFunction",`
`"lambda:GetFunctionConfiguration",`
`"lambda:PutFunctionConcurrency",`
`"lambda:ListTags",`
`"lambda:DeleteFunction",`
`"lambda:GetAlias",`
`"lambda:InvokeFunction",`
`"lambda:GetFunction",`
`"lambda:ListAliases",`
`"lambda:UpdateFunctionConfigur ation",`
`"lambda:UpdateFunctionCode",`
`"lambda:AddPermission",`
`"lambda:DeleteFunctionConcurrency",`
`"lambda:RemovePermission",`
`"lambda:GetPolicy"`
`"lambda:GetAccountSettings",`
`"lambda:ListFunctions",`
`"lambda:ListEventSourceMappings",`
	リソースとしてリストされている Lambda 関数の作成と管理を許可します。この例では、名前のプレフィックスがリソース識別子 `arn:aws:lambda:*:` *`MyAWSAcct Id`* `:function:` *`MyAthenaLambdaFunc tionsPrefix`* `*` で使用されます。ここでの *`MyAthenaLambdaFunctionsPrefix`* は、Lambda 関数グループの名前で使用される共有プレフィックスで、これによって Lambda 関数をリソースとして個別に指定する必要がなくなります。1 つ、または複数の Lambda 関数リソースを指定できます。

`"s3:GetObject"`	リソース識別子 AWS Serverless Applicati on Repository で指定された、`arn:aws:s 3:::awsserverlessrepo-chang esets-` *`1iiv3xa62ln3m`* `/*` が必要なバケットの読み取りを許可します。このバケットは、アカウントに固有のものである場合があります。
`"cloudformation:*"`	リソース *`MyCFStackPrefix`* によって指定されている AWS CloudFormation スタックの作成と管理を許可します。これらのスタックおよびスタックセットは、AWS Serverless

実行可能なアクション	説明
	Application Repository がコネクタと UDF をデプロイする方法です。
`"serverlessrepo:*"`	リソース ID AWS Serverless Application Repository によって指定された `arn:aws:serverlessrepo:*:*:applications/*` でアプリケーションの検索、表示、公開、および更新を許可します。

Amazon Athena ユーザー定義関数 (UDF) を許可する IAM 許可ポリシーの例

このトピックのアクセス許可ポリシーの例は、使用可能な必須アクションとそれらのアクションが許可されるリソースを示しています。同様のアクセス許可ポリシーを IAM ID にアタッチする前に、これらのポリシーを慎重に検討し、要件に従って変更してください。

- Example Policy to Allow an IAM Principal to Run and Return Queries that Contain an Athena UDF Statement
- Example Policy to Allow an IAM Principal to Create an Athena UDF

Example - IAM プリンシパルが Athena UDF ステートメントを含むクエリを実行して返すことを許可する

以下のアイデンティティベースの許可ポリシーは、ユーザーまたは他の IAM プリンシパルが Athena UDF ステートメントを使用するクエリを実行するために必要なアクションを許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "lambda:InvokeFunction",
                "athena:GetQueryResults",
                "s3:ListMultipartUploadParts",
                "athena:GetWorkGroup",
```

```
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "athena:StopQueryExecution",
                "athena:GetQueryExecution",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:athena:*:MyAWSAcctId:workgroup/MyAthenaWorkGroup",
                "arn:aws:s3:::MyQueryResultsBucket/*",
                "arn:aws:lambda:*:MyAWSAcctId:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:MyAWSAcctId:function:AnotherAthenaLambdaFunction"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        }
    ]
}
```

許可の説明

実行可能なアクション	説明
`"athena:StartQueryExecution", "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StopQueryExecution", "athena:GetQueryExecution",`	MyAthenaWorkGroup ワークグループでのクエリの実行に必要な Athena 許可です。
`"s3:PutObject", "s3:GetObject", "s3:AbortMultipartUpload"`	s3:PutObject と s3:AbortMultipartUpload は、arn:aws:s3::: *MyQueryResultsBucket* /* リソース識別子によって指定されたクエリ結果バケットのすべてのサブフォルダへのクエリ結果の書き込みを許可します。ここでの *MyQueryResultsBucket* は、Athena クエリ結果バケットです。詳細に

実行可能なアクション	説明
	ついては、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。 s3:GetObject は、arn:aws:s3::: *MyQueryResultsBucket* として指定されたリソースのクエリ結果とクエリ履歴の読み取りを許可します。ここでの *MyQueryResultsBucket* は、Athena クエリ結果バケットです。詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。 s3:GetObject も、"arn:aws:s3::: *MyLambdaSpillBucket* */ *MyLambdaSpillPrefix* *" として指定されたリソースからの読み取りを許可します。ここでの *MyLambdaSpillPrefix* は、呼び出される Lambda 関数の設定で指定されます。
"lambda:InvokeFunction"	クエリが Resource ブロックで指定された AWS Lambda 関数を呼び出すことを許可します。例えば、*MyAthenaLambdaFunction* が呼び出される Lambda 関数の名前を指定する arn:aws:lambda:*: *MyAWSAcctId* :function: *MyAthenaLambdaFunction* などです。例に示すように、複数の関数を指定できます。

Example – IAM プリンシパルに Athena UDF の作成を許可する

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
```

```
            "Action": [
                "lambda:CreateFunction",
                "lambda:ListVersionsByFunction",
                "iam:CreateRole",
                "lambda:GetFunctionConfiguration",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "lambda:PutFunctionConcurrency",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "lambda:ListTags",
                "iam:ListAttachedRolePolicies",
                "iam:DeleteRolePolicy",
                "lambda:DeleteFunction",
                "lambda:GetAlias",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetPolicy",
                "lambda:InvokeFunction",
                "lambda:GetFunction",
                "lambda:ListAliases",
                "lambda:UpdateFunctionConfiguration",
                "iam:DeleteRole",
                "lambda:UpdateFunctionCode",
                "s3:GetObject",
                "lambda:AddPermission",
                "iam:UpdateRole",
                "lambda:DeleteFunctionConcurrency",
                "lambda:RemovePermission",
                "iam:GetRolePolicy",
                "lambda:GetPolicy"
            ],
            "Resource": [

 "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunctionsPrefix*",
                "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
                "arn:aws:iam::*:role/RoleName",
                "arn:aws:iam::111122223333:policy/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
```

```
                "cloudformation:CreateUploadBucket",
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:ListExports",
                "cloudformation:ListStacks",
                "cloudformation:ListImports",
                "lambda:ListFunctions",
                "iam:ListRoles",
                "lambda:GetAccountSettings",
                "ec2:DescribeSecurityGroups",
                "cloudformation:EstimateTemplateCost",
                "ec2:DescribeVpcs",
                "lambda:ListEventSourceMappings",
                "cloudformation:DescribeAccountLimits",
                "ec2:DescribeSubnets",
                "cloudformation:CreateStackSet",
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "cloudformation:*",
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:111122223333:stack/
serverlessrepo-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:*:transform/Serverless-*",
                "arn:aws:cloudformation:*:111122223333:stackset/aws-serverless-
repository-MyCFStackPrefix*:*",
                "arn:aws:cloudformation:*:111122223333:stackset/
serverlessrepo-MyCFStackPrefix*:*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "serverlessrepo:*",
            "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
        }
    ]
}
```

許可の説明

実行可能なアクション	説明
`"lambda:CreateFunction", "lambda:ListVersionsByFunction", "lambda:GetFunctionConfiguration", "lambda:PutFunctionConcurrency", "lambda:ListTags", "lambda:DeleteFunction", "lambda:GetAlias", "lambda:InvokeFunction", "lambda:GetFunction", "lambda:ListAliases", "lambda:UpdateFunctionConfiguration", "lambda:UpdateFunctionCode", "lambda:AddPermission", "lambda:DeleteFunctionConcurrency", "lambda:RemovePermission", "lambda:GetPolicy" "lambda:GetAccountSettings", "lambda:ListFunctions", "lambda:ListEventSourceMappings",`	リソースとしてリストされている Lambda 関数の作成と管理を許可します。この例では、名前のプレフィックスがリソース識別子 `arn:aws:lambda:*:` *MyAWSAcctId* `:function:` *MyAthenaLambdaFunctionsPrefix* `*` で使用されます。ここでの *MyAthenaLambdaFunctionsPrefix* は、Lambda 関数グループの名前で使用される共有プレフィックスで、これによって Lambda 関数をリソースとして個別に指定する必要がなくなります。1 つ、または複数の Lambda 関数リソースを指定できます。
`"s3:GetObject"`	リソース識別子 AWS Serverless Application Repository で指定された、`arn:aws:s3:::awsserverlessrepo-changesets-` *1iiv3xa62ln3m* `/*` が必要なバケットの読み取りを許可します。
`"cloudformation:*"`	リソース *MyCFStackPrefix* によって指定されている AWS CloudFormation スタックの作成と管理を許可します。これらのスタックおよびスタックセットは、AWS Serverless Application Repository がコネクタと UDF をデプロイする方法です。
`"serverlessrepo:*"`	リソース ID AWS Serverless Application Repository によって指定された `arn:aws:s`

実行可能なアクション	説明
	erverlessrepo:*:*:applications/* でアプリケーションの検索、表示、公開、および更新を許可します。

Athena での ML へのアクセスの許可

Athena ML クエリを実行する IAM プリンシパルには、使用する Sagemaker エンドポイントに対する sagemaker:invokeEndpoint アクションの実行が許可されている必要があります。ユーザー ID にアタッチされた ID ベースのアクセス許可ポリシーに、次のようなポリシーステートメントを含めます。さらに、Athena アクションへの完全なアクセス権を付与する AWS 管理ポリシー: AmazonAthenaFullAccess、またはアクションのサブセットを許可する、変更されたインラインポリシーをアタッチします。

例 の arn:aws:sagemaker:*region*:*AWSAcctID*:*ModelEndpoint* を、クエリで使用するモデルエンドポイントの ARN または ARN に置き換えます。詳細については、「サービス承認リファレンス」の「SageMaker 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。

```
{
            "Effect": "Allow",
            "Action": [
                "sagemaker:invokeEndpoint"
            ],
            "Resource": "arn:aws:sagemaker:us-west-2:123456789012:workteam/public-
crowd/default"
}
```

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

Athena API へのフェデレーションアクセスの有効化

このセクションでは、組織内のユーザーまたはクライアントアプリケーションによる Amazon Athena API オペレーションの呼び出しを許可するフェデレーションアクセスについて説明します。この場合、組織のユーザーは Athena に直接アクセスできません。代わりに、AWS 外部の Microsoft Active Directory でユーザーの認証情報を管理します。Active Directory は SAML 2.0 (Security Assertion Markup Language 2.0) をサポートしています。

このシナリオでのユーザーの認証には、Active Directory Federation Services (ADFS) 3.0 にアクセスし、クライアントアプリケーションによる Athena API オペレーションの呼び出しを有効にするために、SAML 2.0 をサポートする JDBC または ODBC ドライバーを使用します。

AWS での SAML 2.0 のサポートに関する詳細については、「IAM ユーザーガイド」の「SAML 2.0 フェデレーションについて」を参照してください。

> **Note**
>
> Athena API へのフェデレーションアクセスは、特定のタイプの ID プロバイダー (IdP)、および Windows Server の一部である Active Directory Federation Service (ADFS 3.0) に対してサポートされています。フェデレーションアクセスは、IAM アイデンティティセンターの信頼できる ID 伝達機能と互換性がありません。アクセスは、SAML 2.0 をサポートするバージョンの JDBC や ODBC ドライバーを通じて確立されます。詳細については、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

トピック

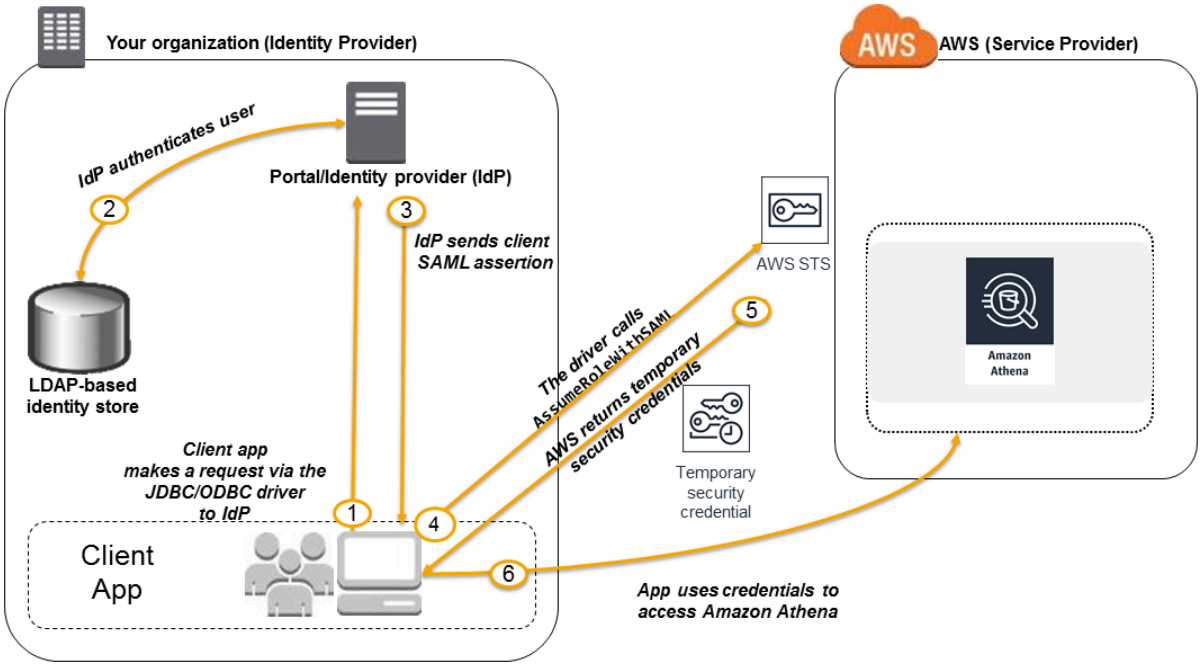
開始する前に

開始する前に、次の前提条件を完了します。

- 組織内で、IdP として ADFS 3.0 をインストールして設定します。
- Athena へのアクセスに使用されるクライアントで、最新バージョンの JDBC または ODBC ドライバーをインストールして設定します。そのドライバーには、SAML 2.0 と互換性のあるフェデレーションアクセスのサポートが含まれている必要があります。詳細については、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

アーキテクチャ図

次の図は、このプロセスを示したものです。



1. 組織のユーザーが、クライアントアプリケーションと JDBC または ODBC ドライバーを使用して、組織の IdP に認証をリクエストします。IdP は ADFS 3.0 です。
2. IdP がユーザーを Active Directory に対して認証します。これは組織の ID ストアです。
3. IdP はユーザーに関する情報を使用して SAML アサーションを構築し、JDBC または ODBC ドライバーを介してクライアントアプリケーションにアサーションを送信します。
4. JDBC または ODBC ドライバーは、AWS Security Token Service AssumeRoleWithSAML API オペレーションを呼び出し、以下のパラメータを渡します。
   - SAML プロバイダーの ARN
   - 引き受けるロールの ARN。
   - IdP からの SAML アサーション

   詳細については、「AWS Security Token Service API リファレンス」の「AssumeRoleWithSAML」を参照してください。
5. JDBC または ODBC ドライバーを介したクライアントアプリケーションへの API レスポンスには、一時的なセキュリティ認証情報が含まれています。

6. クライアントアプリケーションは、一時的なセキュリティ認証情報を使用して Athena API オペレーションを呼び出すことで、ユーザーが Athena API オペレーションにアクセスできるようにします。

手順: Athena API への SAML ベースのフェデレーションアクセス

この手順で、組織の IdP と AWS アカウント間の信頼が確立され、Amazon Athena API オペレーションへの SAML ベースのフェデレーションアクセスが可能になります。

Athena API へのフェデレーションアクセスを有効にするには、以下を実行します。

1. 組織内で、IdP のサービスプロバイダー (SP) として AWS を登録します。このプロセスは、証明書利用者の信頼と呼ばれます。詳細については、「IAM ユーザーガイド」で「証明書利用者の信頼によって SAML 2.0 IdP を設定する」を参照してください。このタスクの一部として、次のステップを実行します。

   a. URL https://signin.aws.amazon.com/static/saml-metadata.xml からサンプル SAML メタデータドキュメントを取得します。

   b. 組織の IdP (ADFS) で、AWS への ID プロバイダーとして IdP を記述する同等のメタデータ XML ファイルを生成します。メタデータファイルには、発行元名、作成日、有効期限、AWS が組織からの認証レスポンス (アサーション) を検証するために使用するキーを含める必要があります。

2. IAM コンソールで、SAML ID プロバイダーのエンティティを作成します。詳細については、「IAM ユーザーガイド」の「SAML ID プロバイダーの作成」を参照してください。このステップの一部として、以下の操作を行います。

   a. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

   b. この手順のステップ 1 で IdP (ADFS) によって生成された SAML メタデータドキュメントをアップロードします。

3. IAM コンソールで、IdP のために 1 つ、または複数の IAM 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。このステップの一部として、以下の操作を行います。

   - ロールのアクセス権限ポリシーで、AWS で組織のユーザーが実行できるアクションをリストします。
   - ロールの信頼ポリシーで、この手順のステップ 2 でプリンシパルとして作成した SAML プロバイダーエンティティを設定します。

これにより、組織と AWS 間で信頼関係が確立されます。

4. 組織の IdP (ADFS) で、組織のユーザーまたはグループを IAM ロールにマップするアサーションを定義します。IAM ロールへのユーザーとグループのマッピングは、クレームルールとしても知られています。組織内の異なるユーザーとグループは、異なる IAM ロールにマップされている場合があることに注意してください。

   ADFS でマッピングを設定する方法については、ブログ記事「Windows Active Directory、ADFS、SAML 2.0 を使用した AWS へのフェデレーションの有効化」を参照してください。

5. JDBC または ODBC ドライバーと SAML 2.0 のサポートをインストールして設定します。詳細については、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

6. アプリケーションから JDBC または ODBC ドライバーへの接続文字列を指定します。アプリケーションが使用する接続文字列の詳細については、JDBC を使用した Amazon Athena への接続 および ODBC を使用した Amazon Athena への接続 トピックから PDF 形式でダウンロードできる「JDBC Driver Installation and Configuration Guide」(JDBC ドライバーのインストールおよび設定ガイド) の「Using the Active Directory Federation Services (ADFS) Credentials Provider」(Active Directory Federation Services (ADFS) 認証情報プロバイダの使用)、または「ODBC Driver Installation and Configuration Guide」(ODBC ドライバーのインストールおよび設定ガイド) の同様のトピックを参照してください。

   以下に示しているのは、ドライバーに対する接続文字列の設定の概要です。

   1. `AwsCredentialsProviderClass configuration` で、ADFS IdP を介した SAML 2.0 ベースの認証を使用するように `com.simba.athena.iamsupport.plugin.AdfsCredentialsProvider` を設定します。
   2. `idp_host` で、ADFS IdP サーバーのホスト名を指定します。
   3. `idp_port` で、SAML アサーションリクエストで ADFS IdP がリッスンするポート番号を指定します。
   4. `UID` および `PWD` で、AD ドメインユーザーの認証情報を提供します。Windows でドライバーを使用する場合、`UID` および `PWD` が指定されていない場合、ドライバーは Windows マシンにログインしているユーザーの認証情報の取得を試みます。

5. 必要に応じて、`ssl_insecure` を `true` に設定します。この場合、ドライバーは ADFS IdP サーバーの SSL 証明書の信頼性を確認しません。ADFS IdP の SSL 証明書がドライバーによって信頼されるように設定されていない場合は、`true` に設定する必要があります。
6. (この手順のステップ 4 で説明しているように) Active Directory ドメインまたはグループの 1 つ以上の IAM ロールへのマッピングを有効にするには、JDBC または ODBC 接続の `preferred_role` で、ドライバー接続のために引き受ける IAM ロール (ARN) を指定します。`preferred_role` の指定はオプションであり、そのロールが、クレームルールに登録された最初のロールではない場合に役立ちます。

この手順の結果として、以下のアクションが実行されます。

1. アーキテクチャ図のステップ 4 にあるように、JDBC または ODBC ドライバーが AWS STS AssumeRoleWithSAML API を呼び出し、それにアサーションを渡します。
2. AWS は、ロールを引き受けるリクエストは SAML プロバイダーエンティティで参照される IdP からのリクエストであることを確認します。
3. リクエストが成功すると、AWS STS AssumeRoleWithSAML API オペレーションから一時的セキュリティ認証情報一式が返され、クライアントアプリケーションではこれを使用して Athena に対して署名付きリクエストを作成します。

   これで、アプリケーションが現在のユーザーに関する情報を取得し、プログラム的に Athena にアクセスできるようになりました。

Athena でのロギングとモニタリング

インシデントを検知し、インシデントの発生時にアラートを受け取って、それらに対応するために、これらのオプションを Amazon Athena で使用します。

- AWS CloudTrail を使用して Athena をモニタリングする — AWS CloudTrail は、ユーザー、ロール、または AWS のサービス が Athena で実行したアクションの記録を提供します。これは、Athena コンソールからのコール、および Athena API オペレーションへのコードコールをイベントとしてキャプチャするので、Athena に対して実行されたリクエスト、リクエスト実行元の IP アドレス、リクエストの実行者、リクエストの実行時、およびその他の詳細を判断できます。詳細については、「AWS CloudTrail を使用した Amazon Athena API コールのログ記録」を参照してください。

Athena を使用して、Athena だけでなく他の AWS のサービス の CloudTrail ログファイルをクエリすることもできます。詳細については、「AWS CloudTrail ログのクエリ」を参照してください。

- CloudTrail と Amazon QuickSight を使用して Athena の使用状況を監視する – Amazon QuickSight は、クラウドを原動力とするフルマネージド型のビジネスインテリジェンスサービスで、組織がどのデバイスからでもアクセスできるインタラクティブなダッシュボードを作成できます。Athena の使用状況をモニタリングするために CloudTrail と Amazon QuickSight を使用するソリューションの例については、「AWS Big·Data·Blog」(ビッグデータブログ) の記事「Realtor.com が Amazon Athena の使用状況を AWS CloudTrail と Amazon QuickSight を使用してモニタリングする方法」を参照してください。
- Athena で EventBridge を使用する - AWS Amazon EventBridge はリソースの変更を記述したシステムイベントのストリームをほぼリアルタイムに配信します。EventBridge は、運用上の変更が生じると同時にそれらを認識して対応し、環境に応答するためのメッセージを送信する、機能をアクティブ化する、変更を行う、および状態情報を収集することによって、必要に即した是正措置を講じます。イベントは、ベストエフォートベースで発生します。詳細については、「Amazon EventBridge ユーザーガイド」の「Getting started with Amazon EventBridge」を参照してください。
- ワークグループを使用してユーザー、チーム、アプリケーション、またはワークロードを分離し、クエリの制限を設定して、クエリのコストを制御する – Amazon CloudWatch でクエリ関連のメトリクスを表示する、スキャンされるデータの量に対する制限を設定することでクエリのコストを制御する、しきい値を作成し、これらのしきい値を超過したときに Amazon SNS アラームなどのアクションをトリガーできます。大まかな手順については、「ワークグループのセットアップ」を参照してください。特定のワークグループへのアクセスを制御するには、リソースレベルの IAM 許可を使用します。詳細については、クエリを実行するためのワークグループの使用およびCloudWatch のメトリクスとイベントを使用したコストの管理とクエリのモニタリングを参照してください。

トピック

- AWS CloudTrail を使用した Amazon Athena API コールのログ記録

AWS CloudTrail を使用した Amazon Athena API コールのログ記録

Athena は AWS CloudTrail と統合されており、これにより、ユーザー、ロール、または AWS のサービス によって Athena で実行されたアクションの記録を提供しています。

CloudTrail は、Athena に対するすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、Athena コンソールのコールと、Athena API オペレーションへのコードコールが含まれます。証跡を作成する場合は、Athena のイベントを含めた CloudTrail イベントの Amazon S3 バケットへ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、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。

CloudTrail によって収集された情報を使用して、Athena に対して実行されたリクエスト、リクエスト実行元の IP アドレス、リクエストの実行者、リクエストの実行時、およびその他の詳細を判断することができます。

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。

Athena を使用して、Athena 自体および他の AWS のサービス から、CloudTrail ログファイルをクエリすることができます。詳細については、「AWS CloudTrail ログのクエリ」、「Hive JSON SerDe」、および AWS ビッグデータブログ記事「Amazon Athena で CTAS ステートメントを使用して、コストを削減し、パフォーマンスを向上させる」(この記事では、CloudTrail を使用して Athena の使用状況に対するインサイトを提供しています) を参照してください。

CloudTrail 内の Athena 情報

CloudTrail は、アカウント作成時に Amazon Web Services アカウントで有効になります。Athena でアクティビティが発生すると、そのアクティビティは AWS のその他のサービスイベントと共に CloudTrail イベントの [イベント履歴] に記録されます。最近のイベントは、Amazon Web Services アカウントで表示、検索、ダウンロードできます。詳細については、「Viewing events with CloudTrail event history」(CloudTrail イベント履歴でのイベントの表示) を参照してください。

Athena のイベントなど、Amazon Web Services 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、CloudTrail ログで収集されたイベントデータをより詳細に分析し、それを基にアクションを取るために他の AWS のサービス を設定できます。詳細については、次を参照してください:

- 「証跡作成の概要」
- 「CloudTrail がサポートされているサービスと統合」
- CloudTrail の Amazon SNS 通知の設定
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

Athena では、すべてのアクションが CloudTrail によってログに記録されます。詳細については、「Amazon Athena API リファレンス」を参照してください。例えば、StartQueryExecution アクションと GetQueryResults アクションをコールすると、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、リクエストの生成者に関する情報が含まれます。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、ルート認証情報と AWS Identity and Access Management (IAM) 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またはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービス によって送信されたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

Athena 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するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意ソースからの単一リクエストを表し、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の情報を含みます。CloudTrail ログファイルは、パブリック API 呼び出し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。

> **Note**
>
> 機密情報の意図しない漏洩を防止するには、`StartQueryExecution` と `CreateNamedQuery` の両方のログの `queryString` のエントリ値を `***OMITTED***` に設定します。これは仕様です。実際のクエリ文字列にアクセスするには、Athena GetQueryExecution API を使用して CloudTrail ログから `responseElements.queryExecutionId` の値を渡すことができます。

次の例は、以下に関する CloudTrail ログエントリを示しています。

- StartQueryExecution (成功)
- StartQueryExecution (失敗)
- CreateNamedQuery

StartQueryExecution (成功)


```
{
 "eventVersion":"1.05",
 "userIdentity":{
    "type":"IAMUser",
    "principalId":"EXAMPLE_PRINCIPAL_ID",
    "arn":"arn:aws:iam::123456789012:user/johndoe",
    "accountId":"123456789012",
    "accessKeyId":"EXAMPLE_KEY_ID",
    "userName":"johndoe"
 },
 "eventTime":"2017-05-04T00:23:55Z",
 "eventSource":"athena.amazonaws.com",
 "eventName":"StartQueryExecution",
 "awsRegion":"us-east-1",
 "sourceIPAddress":"77.88.999.69",
 "userAgent":"aws-internal/3",
 "requestParameters":{
    "clientRequestToken":"16bc6e70-f972-4260-b18a-db1b623cb35c",
    "resultConfiguration":{
       "outputLocation":"s3://DOC-EXAMPLE-BUCKET/test/"
    },
    "queryString":"***OMITTED***"
 },
 "responseElements":{
    "queryExecutionId":"b621c254-74e0-48e3-9630-78ed857782f9"
 },
 "requestID":"f5039b01-305f-11e7-b146-c3fc56a7dc7a",
 "eventID":"c97cf8c8-6112-467a-8777-53bb38f83fd5",
 "eventType":"AwsApiCall",
 "recipientAccountId":"123456789012"
}
```


StartQueryExecution (失敗)


```
{
 "eventVersion":"1.05",
 "userIdentity":{
  "type":"IAMUser",
  "principalId":"EXAMPLE_PRINCIPAL_ID",
  "arn":"arn:aws:iam::123456789012:user/johndoe",
  "accountId":"123456789012",
```

```
  "accessKeyId":"EXAMPLE_KEY_ID",
  "userName":"johndoe"
  },
 "eventTime":"2017-05-04T00:21:57Z",
 "eventSource":"athena.amazonaws.com",
 "eventName":"StartQueryExecution",
 "awsRegion":"us-east-1",
 "sourceIPAddress":"77.88.999.69",
 "userAgent":"aws-internal/3",
 "errorCode":"InvalidRequestException",
 "errorMessage":"Invalid result configuration. Should specify either output location or
 result configuration",
 "requestParameters":{
  "clientRequestToken":"ca0e965f-d6d8-4277-8257-814a57f57446",
  "queryString":"***OMITTED***"
  },
 "responseElements":null,
 "requestID":"aefbc057-305f-11e7-9f39-bbc56d5d161e",
 "eventID":"6e1fc69b-d076-477e-8dec-024ee51488c4",
 "eventType":"AwsApiCall",
 "recipientAccountId":"123456789012"
}
```

CreateNamedQuery

```
{
  "eventVersion":"1.05",
  "userIdentity":{
     "type":"IAMUser",
     "principalId":"EXAMPLE_PRINCIPAL_ID",
     "arn":"arn:aws:iam::123456789012:user/johndoe",
     "accountId":"123456789012",
     "accessKeyId":"EXAMPLE_KEY_ID",
     "userName":"johndoe"
  },
  "eventTime":"2017-05-16T22:00:58Z",
  "eventSource":"athena.amazonaws.com",
  "eventName":"CreateNamedQuery",
  "awsRegion":"us-west-2",
  "sourceIPAddress":"77.88.999.69",
  "userAgent":"aws-cli/1.11.85 Python/2.7.10 Darwin/16.6.0 botocore/1.5.48",
  "requestParameters":{
     "name":"johndoetest",
```

```
      "queryString":"***OMITTED***",
      "database":"default",
      "clientRequestToken":"fc1ad880-69ee-4df0-bb0f-1770d9a539b1"
      },
   "responseElements":{
      "namedQueryId":"cdd0fe29-4787-4263-9188-a9c8db29f2d6"
      },
   "requestID":"2487dd96-3a83-11e7-8f67-c9de5ac76512",
   "eventID":"15e3d3b5-6c3b-4c7c-bc0b-36a8dd95227b",
   "eventType":"AwsApiCall",
   "recipientAccountId":"123456789012"
},
```


Amazon Athena のコンプライアンス検証

Amazon Athena のセキュリティとコンプライアンスは、AWS のさまざまなコンプライアンスプログラムの一環として、サードパーティー監査機関によって評価されます。これらのプログラムには、SOC、PCI、FedRAMP などがあります。

特定のコンプライアンスプログラムの対象範囲内となる AWS のサービス のリストについては、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」を参照してください。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。

AWS Artifact 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。詳細については、「AWS Artifact のレポートのダウンロード」を参照してください。

Athena を使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性や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。AWS では、コンプライアンスに役立つ以下のリソースを提供しています。

- 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焦点を当てたベースライン環境を AWS にデプロイするための手順を示します。
- 「Amazon Web Services での HIPAA のセキュリティとコンプライアンスに向けた構築」 – このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法を説明しています。
- AWS コンプライアンスのリソース – このワークブックとガイドのコレクションは、お客様の業界や場所に適用される場合があります。
- AWS Config — この AWS のサービス では、自社プラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態が評価されます。

- AWS Security Hub — この AWS のサービス では、AWS 内のセキュリティ状態が包括的に示されており、セキュリティ業界の標準およびベストプラクティスへの準拠の確認に役立ちます。

Athena 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。AWS リージョン には、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立・隔離されたアベイラビリティーゾーンがあります。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に比べて、可用性、耐障害性、および拡張性に優れています。

AWS リージョン と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。

AWS のグローバルインフラストラクチャに加えて、Athena 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うえで役立つ複数の機能を提供しています。

Athena はサーバーレスであるため、セットアップまたは管理するインフラストラクチャはありません。Athena は高い可用性を備えており、複数のアベイラビリティーゾーンにまたがるコンピューティングリソースを使用してクエリを実行して、特定のアベイラビリティーゾーンが到達不可能であれば、クエリを自動で適切にルーティングします。Athena は、その基盤となるデータストアとして Amazon S3 を使用しているため、データの可用性と耐久性が向上します。Amazon S3 は重要なデータを保存するための耐久性に優れたインフラストラクチャを提供し、オブジェクトの 99.999999999% の耐久性を実現するように設計されています。データは複数の施設間で冗長化され、各施設で複数のデバイスに保存されます。

Athena で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 Athena は AWS グローバルネットワークセキュリティで保護されています。AWS セキュリティサービスと AWS がインフラストラクチャを保護する方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

Athena には、AWS が公開した API コールを使用してネットワーク経由でアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2 は必須で TLS 1.3 がお勧めです。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。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、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキー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、一時的な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。

Athena オペレーションへのアクセスを制限するには、IAM ポリシーを使用します。IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

Athena のマネージドポリシーは使いやすく、サービスの進化に伴って必要なアクションで自動更新されます。カスタマーマネージドポリシーとインラインポリシーは、さらにきめ細かな Athena アクションをポリシー内に指定することで、ポリシーを微調整することを可能にします。データの Amazon S3 の場所に対する適切なアクセス権を付与します。Amazon S3 へのアクセス権を付与する方法の詳細とシナリオについては、「Amazon Simple Storage Service デベロッパーガイド」の「チュートリアル例: アクセスの管理」を参照してください。許可する Amazon S3 アクションの詳細と例については、「クロスアカウントアクセス」のバケットポリシー例を参照してください。

トピック

- インターフェイス VPC エンドポイントを使用して Amazon Athena に接続する

インターフェイス VPC エンドポイントを使用して Amazon Athena に接続する

VPC のセキュリティ体制を改善するには、Virtual Private Cloud (VPC) のインターフェイス VPC エンドポイント (AWS PrivateLink) と AWS Glue VPC エンドポイントを使用します。インターフェイス VPC エンドポイントは、VPC 内からどの送信先に到達できるかを制御できるようにすることで、セキュリティを強化します。各 VPC エンドポイントは、VPC サブネット内のプライベート IP アドレスを持つ 1 つ以上の Elastic Network Interface (ENI) で表されます。

インターフェイス VPC エンドポイントは VPC を Athena に直接接続します。その際、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 接続を使用しません。VPC 内のインスタンスが Athena API と通信するためにパブリック IP アドレスは必要ありません。

VPC 経由で Athena を使用するには、VPC 内にあるインスタンスから接続するか、Amazon Virtual Private Network (VPN) または AWS Direct Connect を使用してプライベートネットワークを VPC に接続する必要があります。Amazon VPN については、「Amazon Virtual Private Cloud ユーザーガイド」の「VPN 接続」を参照してください。AWS Direct Connect の詳細については、「AWS Direct Connect ユーザーガイド」の「接続を作成する」を参照してください。

Athena は、Amazon VPC と Athena の両方が利用可能なすべての AWS リージョン で VPC エンドポイントをサポートしています。

インターフェイス VPC エンドポイントを作成し、AWS Management Console か AWS Command Line Interface (AWS CLI) コマンドのいずれかを使用して、Athena に接続できます。詳細については、「インターフェイスエンドポイントの作成」を参照してください。

インターフェイス VPC エンドポイントを作成した後、エンドポイントのプライベート DNS ホスト名を有効にすると、デフォルトの Athena エンドポイント (https://athena.*Region*.amazonaws.com) が VPC エンドポイントに解決されます。

プライベート DNS ホスト名を有効にしない場合は、Amazon VPC が以下の形式で使用できる DNS エンドポイント名を提供します。

```
VPC_Endpoint_ID.athena.Region.vpce.amazonaws.com
```

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

Athena は、VPC 内にあるすべての API アクションへの呼び出しをサポートします。

Athena の VPC エンドポイントポリシーを作成する

Athena の Amazon VPC エンドポイントに対するポリシーを作成して、以下のような制限を指定することができます。

- [プリンシパル] – アクションを実行できるプリンシパル。
- [アクション] – 実行可能なアクション。
- [リソース] – このアクションを実行できるリソース。
- [信頼できる D のみ] — `aws:PrincipalOrgId` 条件を使用して、AWS 組織の一部である認証情報のみにアクセスを制限します。これにより、意図しないプリンシパルによるアクセスを防ぐことができます。

- [信頼できるリソースのみ] — aws:ResourceOrgId 条件を使用して、意図しないリソースへのアクセスを防ぎます。
- [信頼できる ID とリソースのみ] — VPC エンドポイント用のポリシーを組み合わせて、意図しないプリンシパルやリソースへのアクセスを防ぎます。

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントによるサービスへのアクセスの制御」および、AWS ホワイトペーパー「AWS におけるデータ境界の構築」の「付録 2 — VPC エンドポイントポリシーの例」を参照してください。

Example – VPC エンドポイントポリシー

次の例では、組織リソースへの組織 ID によるリクエストを許可し、AWS サービスプリンシパルによるリクエストを許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsByOrgsIdentitiesToOrgsResources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "my-org-id",
                    "aws:ResourceOrgID": "my-org-id"
                }
            }
        },
        {
            "Sid": "AllowRequestsByAWSServicePrincipals",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "Bool": {
```

```
                    "aws:PrincipalIsAWSService": "true"
                }
            }
        }
    ]
}
```

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

共有サブネット

自分と共有されているサブネットで VPC エンドポイントを作成、説明、変更、または削除することはできません。ただし、VPC エンドポイントを使用することはできます。VPC 共有の詳細については、「Amazon VPC ユーザーガイド」の「VPC を他のアカウントと共有する」を参照してください。

Athena での設定と脆弱性分析

Athena はサーバーレスであるため、お客様が設定や管理を行うインフラストラクチャはありません。AWS がゲストオペレーティングシステム (OS) やデータベースへのパッチの適用、ファイアウォールの設定、災害対策などの基本的なセキュリティタスクを処理します。これらの手順は適切な第三者によって確認され、証明されています。詳細については、以下の AWS リソースを参照してください。

- 責任共有モデル
- セキュリティ、アイデンティティ、コンプライアンスのベストプラクティス

Athena を使用して AWS Lake Formation に登録されたデータをクエリする

AWS Lake Formation では、Amazon S3 に保存されているデータの読み込みに Athena のクエリを使用するときに、データベース、テーブル、および列レベルでのアクセスポリシーを定義し、実施することができます。Lake Formation は、Amazon S3 に保存されているデータに対して、承認およびガバナンスレイヤーを提供します。Lake Formation の許可階層を使用して、データベース、テーブル、および列などのデータカタログオブジェクトを読み込む許可を付与または取り消すことができます。Lake Formation は許可の管理を簡素化し、データにきめ細かなアクセスコントロール (FGAC) を実装できるようにします。

Athena を使用して、Lake Formation に登録されているデータと、Lake Formation に登録されていないデータの両方をクエリできます。

Lake Formation の許可は、Lake Formation に登録された Amazon S3 の場所からのソースデータのクエリに Athena を使用する場合に適用されます。Lake Formation の許可は、登録された Amazon S3 のデータの場所をポイントするデータベースとテーブルを作成するときにも適用されます。Lake Formation を使用して登録されたデータで Athena を使用するには、Athena が AWS Glue Data Catalog を使用するように設定されている必要があります。

Lake Formation の許可は、Amazon S3 にオブジェクトを書き込むときには適用されず、Amazon S3 に保存されているデータや、Lake Formation に登録されていないメタデータをクエリするときにも適用されません。Amazon S3 のソースデータと、Lake Formation に登録されていないメタデータへのアクセスは、Amazon S3 および AWS Glue アクションの IAM アクセス許可ポリシーによって決定されます。Amazon S3 にある Athena のクエリ結果の場所を Lake Formation に登録することはできず、Amazon S3 の IAM 許可ポリシーがアクセスを制御します。さらに、Lake Formation の許可は、Athena のクエリ履歴には適用されません。クエリ履歴へのアクセスは、Athena ワークグループを使用して制御できます。

Lake Formation の詳細については、「AWS Lake Formation デベロッパーガイド」の「Lake Formation のよくある質問」を参照してください。

トピック

- Athena が Lake Formation に登録されたデータにアクセスする方法
- Athena を使用して Lake Formation に登録されたデータをクエリするときの考慮事項と制限
- Lake Formation と Athena ユーザー許可の管理
- 既存のデータベースとテーブルへの Lake Formation 許可の適用
- Athena へのフェデレーションアクセスのための Lake Formation と Athena JDBC および ODBC ドライバーの使用

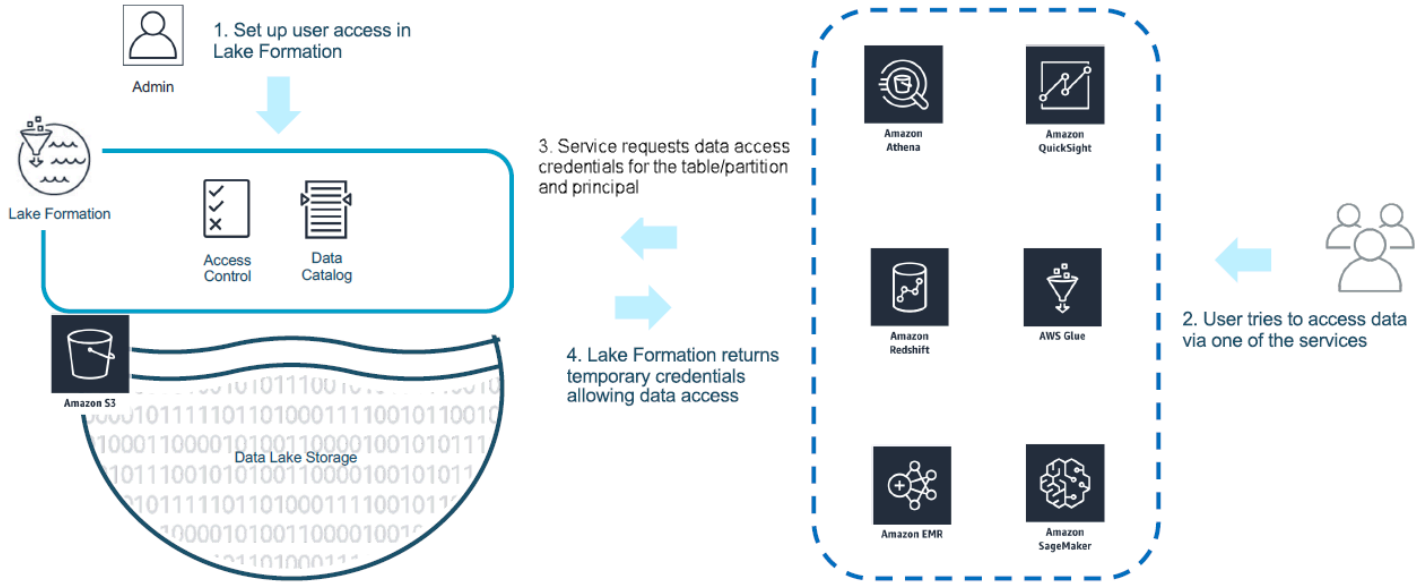
Athena が Lake Formation に登録されたデータにアクセスする方法

このセクションで説明するアクセスワークフローが適用されるのは、Lake Formation に登録された Amazon S3 の場所とメタデータオブジェクトに対して Athena クエリを実行している場合のみです。詳細については、「AWS Lake Formation デベロッパーガイド」の「データレイクの登録」を参照してください。Lake Formation 管理者は、データの登録に加えて、データカタログ内のメタデータ、および Amazon S3 のデータの場所へのアクセス権を付与または取り消す Lake Formation 許可

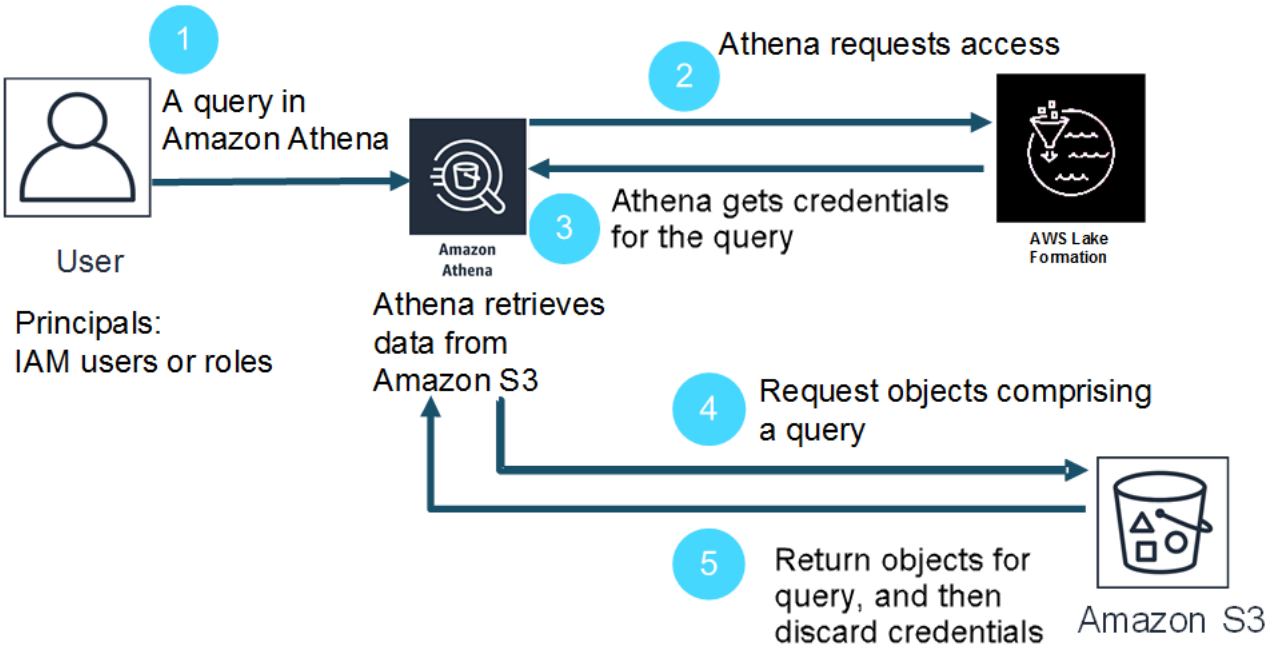
を適用します。詳細については、「AWS Lake Formation デベロッパーガイド」の「メタデータとデータに対するセキュリティとアクセスコントロール」を参照してください。

Lake Formation は、Athena プリンシパル (ユーザー、グループ、またはロール) が Lake Formation を使用して登録されたデータに対するクエリを実行するたびに、データベース、テーブル、および Amazon S3 の場所に対する、そのクエリに見合った適切な Lake Formation 許可がプリンシパルにあることを検証します。プリンシパルにアクセス権がある場合、Lake Formation は一時的な認証情報を Athena に供給し、クエリが実行されます。

以下の図は、上記のフローを説明するものです。



以下の図は、Simple Storage Service (Amazon S3) の場所が Lake Formation に登録されているときのテーブルに対する仮説上の SELECT クエリについて、Athena での認証情報の供給がクエリごとにどのように機能するかを示しています。



1. プリンシパルが Athena で SELECT クエリを実行します。
2. Athena がクエリを分析し、Lake Formation の許可をチェックして、テーブルとテーブルの列に対するアクセス権がプリンシパルに付与されているかどうかを確認します。
3. プリンシパルにアクセス権がある場合、Athena が Lake Formation からの認証情報をリクエストします。プリンシパルにアクセス権がない場合は、Athena がアクセス拒否エラーを発行します。
4. Lake Formation が、Amazon S3 からデータを読み込むときに使用する認証情報と、許可される列のリストを Athena に発行します。
5. Athena が Lake Formation の一時的な認証情報を使用して、Amazon S3 からデータをクエリします。クエリが完了すると、Athena が認証情報を破棄します。

Athena を使用して Lake Formation に登録されたデータをクエリするときの考慮事項と制限

Athena を使用して Lake Formation に登録されたデータをクエリするときは、以下の点を考慮します。詳細については、「AWS Lake Formation デベロッパーガイド」の「AWS Lake Formation の既知の問題」を参照してください。

考慮事項と制約事項

- Avro とカスタム SerDe を使用する一部の状況で、許可されていないユーザーに列メタデータが表示される

- ビューに対する Lake Formation 許可の使用
- Lake Formation のきめ細かなアクセス制御と Athena ワークグループ
- Lake Formation に登録されていない Amazon S3 の Athena クエリ結果の場所
- Athena Workgroups を使用してクエリ履歴へのアクセスを制限する
- データカタログへのクロスアカウントアクセス
- Lake Formation 登録済みの CSE-KMS で暗号化された Amazon S3 の場所
- Lake Formation に登録されているパーティションされたデータの場所がテーブルのサブディレクトリ内にある必要がある
- Create Table As Select (CTAS) クエリに Simple Storage Service (Amazon S3) の書き込み許可が必要になる
- デフォルトデータベースに DESCRIBE 許可が必要になる

Avro とカスタム SerDe を使用する一部の状況で、許可されていないユーザーに列メタデータが表示される

Lake Formation の列レベルの承認は、ユーザーが Lake Formation の許可がない列のデータにアクセスできないようにします。ただし、特定の状況では、ユーザーがデータに対する許可がない列を含めた、テーブル内のすべての列を説明するメタデータにアクセスできます。

これは、Apache Avro ストレージ形式を使用するか、SerDe 定義と共にテーブルプロパティにテーブルスキーマが定義されているカスタムシリアライザー/デシリアライザー (SerDe) を使用して、列メタデータがテーブルのテーブルプロパティに保存されたときに発生します。Athena で Lake Formation を使用するときは、Lake Formation に登録するテーブルプロパティの内容を確認し、可能な場合はテーブルプロパティに保存される情報を制限して、機密のメタデータがユーザーに表示されないようにすることをお勧めします。

ビューに対する Lake Formation 許可の使用

Lake Formation に登録されたデータについて、Athena ユーザーが `VIEW` を作成できるのは、`VIEW` の基礎であるテーブル、列、および Simple Storage Service (Amazon S3) のソースデータの場所に対する Lake Formation の許可がある場合のみです。`VIEW` が Athena で作成されたら、Lake Formation 許可を `VIEW` に適用できます。列レベルのアクセス許可は、`VIEW` では使用できません。`VIEW` に対する Lake Formation の許可はあるが、ビューの基礎であるテーブルと列に対する許可はないというユーザーは、データのクエリに `VIEW` を使用できません。ただし、この許可の組み合わせを持つユーザーは、`DESCRIBE VIEW`、`SHOW CREATE VIEW`、および `SHOW COLUMNS` を使用

して VIEW メタデータを表示できます。このため、各 VIEW に対する Lake Formation 許可は、基盤となるテーブルの許可と合致させておくようにしてください。テーブルで定義されているセルフィルターは、そのテーブルの VIEW に適用されません。リソースリンク名は、元となるアカウントのリソースと同じ名前である必要があります。クロスアカウント設定でビューを操作するとき、他にも制限があります。アカウント間での共有ビューの許可のセットアップの詳細については、「データカタログへのクロスアカウントアクセス」を参照してください。

Lake Formation のきめ細かなアクセス制御と Athena ワークグループ

同じ Athena ワークグループのユーザーは、Lake Formation のきめ細かなアクセス制御によりワークグループがアクセスできるように設定したデータを見ることができます。Lake Formation でのきめ細かいアクセス制御の使用について詳しくは、「AWS ビッグデータブログ」の「AWS Lake Formation を使用したきめ細かなアクセス制御の管理」を参照してください。

Lake Formation に登録されていない Amazon S3 の Athena クエリ結果の場所

Amazon S3 にある Athena のクエリ結果の場所を Lake Formation に登録することはできません。Lake Formation 許可は、これらの場所へのアクセスを制限しません。アクセスを制限しない限り、Athena ユーザーは、データに対する Lake Formation の許可がなくてもクエリ結果ファイルとメタデータにアクセスできます。この問題を回避するには、ワークグループを使用してクエリ結果の場所を指定し、ワークグループのメンバーシップを Lake Formation の許可と合致させることが推奨されます。その後、IAM 許可ポリシーを使用して、クエリ結果の場所へのアクセスを制限できます。クエリ結果の詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。

Athena Workgroups を使用してクエリ履歴へのアクセスを制限する

Athena クエリ履歴は、保存されたクエリと完全なクエリ文字列のリストを公開します。ワークグループを使用してクエリ履歴へのアクセスを分離しない限り、Lake Formation のデータをクエリする権限がない Athena ユーザーが、列名や選択条件などを含めた、そのデータに対して実行されるクエリ文字列を表示することができます。ワークグループを使用してクエリ履歴を分離し、Athena ワークグループのメンバーシップを Lake Formation 許可に合致させてアクセスを制限することが推奨されます。詳細については、「ワークグループを使用してクエリのアクセスとコストを制御する」を参照してください。

データカタログへのクロスアカウントアクセス

別のアカウントのデータカタログにアクセスするには、Athena のクロスアカウント AWS Glue 機能を使用するか、Lake Formation でクロスアカウントアクセスをセットアップします。

データカタログへの Athena クロスアカウントアクセス

Athena のクロスアカウント AWS Glue カタログ機能を使用して、アカウントにカタログを登録できます。この機能は Athena エンジンバージョン 2 以降でのみ使用でき、アカウント間の同一リージョンでの使用に制限されています。詳細については、「別のアカウントからの AWS Glue Data Catalog の登録」を参照してください。

共有するデータカタログのリソースポリシーが AWS Glue で設定されている場合、次の例のように AWS Resource Access Manager へのアクセスを許可し、アカウント B へ許可を付与して、アカウント A のデータカタログを使用するように、更新する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "ram.amazonaws.com"
        },
        "Action": "glue:ShareResource",
        "Resource": [
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:table/*/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:database/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:catalog"
        ]
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::<ACCOUNT-B>:root"
        },
        "Action": "glue:*",
        "Resource": [
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:table/*/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:database/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:catalog"
        ]
    }
    ]
}
```

詳細については、「AWS Glue データカタログへのクロスアカウントアクセス」を参照してください。

Lake Formation でのクロスアカウントアクセスのセットアップ

AWS Lake Formation では、単一のアカウントを使用して中央データカタログを管理できます。この機能を使用して、データカタログメタデータと基盤となるデータにクロスアカウントアクセスを実装します。例えば、所有者アカウントは、テーブルに対する SELECT 許可を別の (受信者) アカウントに付与できます。

共有のデータベースまたはテーブルを Athena クエリエディタに表示するには、Lake Formation で共有のデータベースまたはテーブルへのリソースリンクを作成します。Lake Formation の受信者アカウントが所有者のテーブルをクエリすると、CloudTrail が受信者アカウントと所有者アカウントの両方のログにデータアクセスイベントを追加します。

共有ビューの場合は、次の点に注意してください。

- クエリは、ソーステーブルまたはビューではなく、ターゲットリソースリンクで実行され、出力がターゲットアカウントと共有されます。
- ビューだけを共有するだけでは不十分です。ビューの作成に関与するすべてのテーブルは、クロスアカウント共有の一部である必要があります。
- 共有リソースで作成されたリソースリンクの名前は、所有者アカウントのリソースの名前と一致する必要があります。名前が一致しない場合、「Failed analyzing stored view 'awsdatacatalog.*my-lf-resource-link*.*my-lf-view*': line 3:3: Schema *schema_name* does not exist」のようなエラーメッセージが表示されます。

Lake Formation でのクロスアカウントアクセスの詳細については、「AWS Lake Formation デベロッパーガイド」の以下のリソースを参照してください。

クロスアカウントアクセス

Lake Formation でのリソースリンクの仕組み

CloudTrail のクロスアカウントロギング

Lake Formation 登録済みの CSE-KMS で暗号化された Amazon S3 の場所

以下の特性を持つ Apache Iceberg などの Open Table Format (OTF) テーブルを Athena でクエリすることはできません。

- テーブルが Lake Formation 登録済みの Amazon S3 の場所に基づいている。
- Amazon S3 内のオブジェクトがクライアント側の暗号化 (CSE) を使用して暗号化されている。
- 暗号化が AWS KMS カスタマーマネージドキー (CSE_KMS) を使用している。

CSE_KMS キーで暗号化されている OTF 以外のテーブルをクエリするには、CSE 暗号化に使用している AWS KMS キーのポリシーに、以下のブロックを追加します。*<KMS_KEY_ARN>* は、データを暗号化する AWS KMS キーの ARN です。*<IAM-ROLE-ARN>* は、Lake Formation で Amazon S3 を登録する IAM ロールの ARN です。

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "*"
    },
    "Action": "kms:Decrypt",
    "Resource": "<KMS-KEY-ARN>",
    "Condition": {
        "ArnLike": {
            "aws:PrincipalArn": "<IAM-ROLE-ARN>"
        }
    }
}
```

Lake Formation に登録されているパーティションされたデータの場所がテーブルのサブディレクトリ内にある必要がある

Lake Formation に登録されているパーティションされたテーブルは、パーティションされたデータが Amazon S3 にあるテーブルのサブディレクトリであるディレクトリに置かれている必要があります。例えば、場所が `s3://DOC-EXAMPLE-BUCKET/mytable` で、パーティションが `s3://DOC-EXAMPLE-BUCKET/mytable/dt=2019-07-11`、`s3://DOC-EXAMPLE-BUCKET/mytable/dt=2019-07-12` などにあるテーブルは、Lake Formation に登録し、Athena を使用してクエリできます。反対に、場所が `s3://DOC-EXAMPLE-BUCKET/mytable` で、パーティションが `s3://DOC-EXAMPLE-BUCKET/dt=2019-07-11`、`s3://DOC-EXAMPLE-BUCKET/dt=2019-07-12` などにあるテーブルを Lake Formation に登録することはできません。このようなパーティションは `s3://DOC-EXAMPLE-BUCKET/mytable` のサブディレクトリではなく、それらを Athenaから読み込むこともできないからです。

Create Table As Select (CTAS) クエリに Simple Storage Service (Amazon S3) の書き込み許可が必要になる

Create Table As Statements (CTAS) には、テーブルの Amazon S3 の場所への書き込みアクセス権が必要です。Lake Formation に登録されたデータに対して CTAS クエリを実行するには、Athena ユーザーに、データの場所を読み込むための適切な Lake Formation 許可に加えて、テーブルの

Amazon S3 の場所に書き込むための IAM 許可が必要です。詳細については、「クエリ結果からのテーブルの作成 (CTAS)」を参照してください。

デフォルトデータベースに DESCRIBE 許可が必要になる

`default` データベースには Lake Formation の DESCRIBE 許可が必要です。以下の AWS CLI コマンドの例では、AWS アカウント `111122223333` のユーザー `datalake_user1` に、`default` データベースに対する `DESCRIBE` アクセス許可を付与しています。

```
aws lakeformation grant-permissions --principal
 DataLakePrincipalIdentifier=arn:aws:iam::111122223333:user/datalake_user1 --
permissions "DESCRIBE" --resource '{ "Database": {"Name":"default"}}
```

詳細については、「AWS Lake Formation デベロッパーガイド」の「Lake Formation のアクセス許可リファレンス」を参照してください。

Lake Formation と Athena ユーザー許可の管理

Lake Formation は、Lake Formation に登録されている Amazon S3 データストアをクエリするための認証情報を供給します。以前に IAM ポリシーを使用して Amazon S3 にあるデータの場所の読み込みを許可または拒否した場合は、その代わりに Lake Formation 許可を使用できます。ただし、他の IAM 許可は引き続き必要です。

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、IAM ユーザーガイドの「IAM でのセキュリティベストプラクティス」を参照してください。

以下のセクションでは、Lake Formation に登録されたデータのクエリに Athena を使用するうえで必要な許可を要約しています。詳細については、「AWS Lake Formation デベロッパーガイド」の「AWS Lake Formation でのセキュリティ」を参照してください。

許可の概要

- Lake Formation と Athena に対するアイデンティティベースの許可
- Athena のクエリ結果の場所に対する Simple Storage Service (Amazon S3) の許可
- 履歴をクエリする Athena ワークグループのメンバーシップ
- データに対する Lake Formation 許可
- Simple Storage Service (Amazon S3) の場所に書き込むための IAM 許可
- 暗号化されたデータ、メタデータ、および Athena のクエリ結果に対する許可
- 外部アカウントの Amazon S3 バケットに対するリソースベースの許可 (オプション)

Lake Formation と Athena に対するアイデンティティベースの許可

Lake Formation に登録されたデータのクエリに Athena を使用するユーザーには、`lakeformation:GetDataAccess` アクションを許可する IAM 許可ポリシーが必要です。AWS 管理ポリシー: AmazonAthenaFullAccessでは、このアクションが許可されます。インラインポリシーを使用する場合は、このアクションを許可するように許可ポリシーを更新してください。

Lake Formation では、データレイク管理者が、データベースやテーブルなどのメタデータオブジェクトの作成、他のユーザーへの Lake Formation 許可の付与、および新しい Amazon S3 の場所の登録を行う許可を持っています。新しい場所を登録するには、Lake Formation のためのサービスリンクロールに対する許可が必要です。詳細については、「AWS Lake Formation デベロッパーガイド」の「データレイク管理者の作成」および「Lake Formation のサービスにリンクしたロールのアクセス許可」を参照してください。

Lake Formation ユーザーは、データレイク管理者によって付与された Lake Formation 権限に基づいて、データベース、テーブル、テーブル列、および基盤となる Amazon S3 データストアのクエリに Athena を使用できます。ユーザーは、データベースやテーブルを作成したり、Lake Formation に新しい Amazon S3 の場所を登録したりすることはできません。詳細については、「AWS Lake Formation デベロッパーガイド」の「データレイクユーザーの作成」を参照してください。

Athena では、アイデンティティベースのアクセス許可ポリシー (Athena ワークグループ用を含む) が、引き続き Amazon Web Services アカウントユーザーによる Athena アクションへのアクセスを制御します。さらに、Athena ドライバーで利用できる SAML ベースの認証を通じて、フェデレーションアクセスが提供される場合もあります。詳細については、「ワークグループを使用してクエリのアクセスとコストを制御する」、「 ワークグループにアクセスするための IAM ポリシー」、および「Athena API へのフェデレーションアクセスの有効化」を参照してください。

詳細については、「AWS Lake Formation デベロッパーガイド」の「Lake Formation へのアクセス許可の付与」を参照してください。

Athena のクエリ結果の場所に対する Simple Storage Service (Amazon S3) の許可

Amazon S3 にある Athena のクエリ結果の場所を Lake Formation に登録することはできません。Lake Formation 許可は、これらの場所へのアクセスを制限しません。アクセスを制限しない限り、Athena ユーザーは、データに対する Lake Formation の許可がなくてもクエリ結果ファイルとメタデータにアクセスできます。この問題を回避するには、ワークグループを使用してクエリ結果の場所を指定し、ワークグループのメンバーシップを Lake Formation の許可と合致させることが推奨されます。その後、IAM 許可ポリシーを使用して、クエリ結果の場所へのアクセスを制限できます。クエリ結果の詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。

履歴をクエリする Athena ワークグループのメンバーシップ

Athena クエリ履歴は、保存されたクエリと完全なクエリ文字列のリストを公開します。ワークグループを使用してクエリ履歴へのアクセスを分離しない限り、Lake Formation のデータをクエリする権限がない Athena ユーザーが、列名や選択条件などを含めた、そのデータに対して実行されるクエリ文字列を表示することができます。ワークグループを使用してクエリ履歴を分離し、Athena ワークグループのメンバーシップを Lake Formation 許可に合致させてアクセスを制限することが推奨されます。詳細については、「ワークグループを使用してクエリのアクセスとコストを制御する」を参照してください。

データに対する Lake Formation 許可

Athena ユーザーには、Lake Formation を使用するための基本的な許可に加えて、クエリするリソースにアクセスするための Lake Formation 許可が必要です。これらの許可は、Lake Formation 管理者によって付与および管理されます。詳細については、「AWS Lake Formation デベロッパーガイド」の「メタデータとデータに対するセキュリティとアクセスコントロール」を参照してください。

Simple Storage Service (Amazon S3) の場所に書き込むための IAM 許可

Amazon S3 に対する Lake Formation 許可には、Amazon S3 に書き込む機能が含まれていません。Create Table As Statements (CTAS) には、テーブルの Amazon S3 の場所への書き込みアクセス権が必要です。Lake Formation に登録されたデータに対して CTAS クエリを実行するには、Athena ユーザーに、データの場所を読み込むための適切な Lake Formation 許可に加えて、テーブルの Amazon S3 の場所に書き込むための IAM 許可が必要です。詳細については、「クエリ結果からのテーブルの作成 (CTAS)」を参照してください。

暗号化されたデータ、メタデータ、および Athena のクエリ結果に対する許可

Amazon S3 にある基盤となるソースデータと、Lake Formation に登録されたデータカタログ内のメタデータは暗号化することができます。Lake Formation に登録されたデータをクエリするために Athena を使用しているときに、Athena がクエリ結果の暗号化を処理する方法に変更はありません。詳細については、「Amazon S3 に保存された Athena のクエリ結果の暗号化」を参照してください。

- ソースデータの暗号化 — Amazon S3 データの場所にあるソースデータの暗号化がサポートされています。Lake Formation に登録されている暗号化された Amazon S3 の場所をクエリする Athena ユーザーには、データを暗号化および復号化するための許可が必要です。要件の詳細については、「サポートされる Amazon S3 の暗号化オプション」および「Amazon S3 の暗号化されたデータに対する許可」を参照してください。

- メタデータの暗号化 – データカタログ内のメタデータの暗号化がサポートされています。Athena を使用するプリンシパルの場合、アイデンティティベースのポリシーで、メタデータを暗号化するために使用されるキーの `"kms:GenerateDataKey"`、`"kms:Decrypt"`、および `"kms:Encrypt"` アクションが許可されている必要があります。詳細については、「AWS Glue デベロッパーガイド」の「Data Catalog の暗号化」と、「Athena から AWS Glue Data Catalog の暗号化されたメタデータにアクセスする」を参照してください。

外部アカウントの Amazon S3 バケットに対するリソースベースの許可 (オプション)

別のアカウントにある Amazon S3 のデータの場所をクエリするには、リソースベースの IAM ポリシー (バケットポリシー) がその場所へのアクセスを許可する必要があります。詳細については、「Athena での Amazon S3 バケットへのクロスアカウントアクセス」を参照してください。

別のアカウントにあるデータカタログへのアクセスについては、「データカタログへの Athena クロスアカウントアクセス」を参照してください。

既存のデータベースとテーブルへの Lake Formation 許可の適用

Athena を初めて使用しており、クエリデータへのアクセスを設定するために Lake Formation を使用する場合は、ユーザーが Amazon S3 データを読み込み、メタデータを作成できるように IAM ポリシーを設定する必要はありません。許可を管理には Lake Formation を使用できます。

Lake Formation へのデータの登録と IAM 許可ポリシーの更新は必須ではありません。データが Lake Formation に登録されていない場合、Amazon S3 (および、該当する場合、AWS Glue) における適切なアクセス許可を持つ Athena ユーザーは、Lake Formation に登録されていないデータを引き続きクエリできます。

Lake Formation に登録されていないデータをクエリする既存の Athena ユーザーがいる場合は、Amazon S3 (および、該当する場合、AWS Glue Data Catalog) の IAM アクセス許可を更新して、Lake Formation アクセス許可を使用してユーザーアクセスを一元的に管理できます。Amazon S3 の データの場所を読み込む許可については、リソースベースおよびアイデンティティベースのポリシーを更新して Amazon S3 許可を変更できます。メタデータへのアクセスについては、AWS Glue できめ細かなアクセスコントロールのためのリソースレベルのポリシーを設定した場合、その代わりに Lake Formation のアクセス許可を使用してアクセスを管理できます。

詳細については、「AWS Glue Data Catalog のデータベースとテーブルへのきめ細かなアクセス」と、「AWS Lake Formation デベロッパーガイド」の「AWS Glue データアクセス許可の AWS Lake Formation モデルへのアップグレード」を参照してください。

Athena へのフェデレーションアクセスのための Lake Formation と Athena JDBC および ODBC ドライバーの使用

Athena JDBC および ODBC ドライバーは、Okta および Microsoft Active Directory Federation Services (AD FS) ID プロバイダーを使用する Athena での SAML 2.0 ベースのフェデレーションをサポートします。Amazon Athena をAWS Lake Formation に統合することで、企業の認証情報を使用した Athena への SAML ベースの認証が可能になります。Lake Formation と AWS Identity and Access Management (IAM) を使用すると、SAML ユーザーが利用できるデータに対して、きめ細かな列レベルのアクセスコントロールを維持できます。Athena JDBC および ODBC ドライバーでは、ツールまたはプログラムによるアクセスにフェデレーションアクセスを使用できます。

Lake Formation によって制御されているデータソースへのアクセスに Athena を使用するには、ID プロバイダー (IdP) と AWS Identity and Access Management (IAM) ロールを設定して、SAML 2.0 ベースのフェデレーションを有効にする必要があります。詳細なステップについては、「チュートリアル: Lake Formation および JDBC を使用した Okta ユーザーの Athena へのフェデレーションアクセスの設定」を参照してください。

前提条件

フェデレーションアクセスに Amazon Athena と Lake Formation を使用するには、以下の要件を満たす必要があります。

- Okta または Microsoft Active Directory Federation Services (AD FS) などの既存の SAML ベースの ID プロバイダーを使用して企業 ID を管理している。
- AWS Glue Data Catalog をメタデータストアとして使用します。
- AWS Glue Data Catalog 内のデータベース、テーブル、および列にアクセスをするためのアクセス許可を Lake Formation で定義し、管理します。詳細については、「AWS Lake Formation デベロッパーガイド」を参照してください。
- Athena JDBC ドライバーのバージョン 2.0.14 以降、または Athena ODBC ドライバーのバージョン 1.1.3 以降を使用している。

考慮事項と制約事項

Athena JDBC または ODBC ドライバーと Lake Formation を使用して Athena へのフェデレーションアクセスを設定するときは、以下の点に留意してください。

- 現在、Athena JDBC および ODBC ドライバーは、Okta、Microsoft Active Directory Federation Services (AD FS) および、Azure AD アイデンティティプロバイダーをサポートしていま

す。Athena JDBC ドライバーには、他の ID プロバイダーを使用するために拡張できる汎用 SAML クラスがありますが、他の ID プロバイダー (IdP) を Athena で使用できるようにするカスタム拡張のサポートは限られている場合があります。

- JDBC ドライバーと ODBC ドライバーを使用するフェデレーションアクセスは、IAM アイデンティティセンターの信頼できる ID 伝達機能と互換性がありません。
- 現在、Athena での IdP および SAML の使用に対するサポートの設定に Athena コンソールを使用することはできません。このサポートを設定するには、サードパーティー ID プロバイダー、Lake Formation コンソールと IAM 管理コンソール、および JDBC または ODBC ドライバークライアントを使用してください。
- Lake Formation と Athena での使用のために ID プロバイダーと SAML を設定する前に、SAML 2.0 の仕様と、それが ID プロバイダーと連携する仕組みを理解しておく必要があります。
- SAML プロバイダーと Athena JDBC および ODBC ドライバーはサードパーティー提供のものであるため、それらの使用に関連する問題に対する AWS によるサポートは限られている場合があります。

トピック

- チュートリアル: Lake Formation および JDBC を使用した Okta ユーザーの Athena へのフェデレーションアクセスの設定

チュートリアル: Lake Formation および JDBC を使用した Okta ユーザーの Athena へのフェデレーションアクセスの設定

このチュートリアルでは、Okta、AWS Lake Formation、AWS Identity and Access Management アクセス許可、および Athena JDBC ドライバーを設定して、Athena の SAML ベースのフェデレーション使用を有効にする方法を説明します。Lake Formation は、Athena で利用できるデータに対するきめ細かなアクセスコントロールを SAML ベースのユーザーに提供します。この設定を行うため、チュートリアルでは Okta デベロッパーコンソール、AWS IAM コンソール、Lake Formation コンソール、および SQL Workbench/J ツールを使用します。

前提条件

このチュートリアルでは、以下が実行済みであることを前提としています。

- Amazon Web Services アカウントが作成されている。アカウントを作成するには、Amazon Web Services のホームページにアクセスしてください。
- Simple Storage Service (Amazon S3) に Athena のクエリ結果の場所がセットアップされている。

- Lake Formation に Amazon S3 データバケットの場所が登録されている。
- Amazon S3 のデータにポイントするデータベースとテーブルが AWS Glue データカタログで定義されている。
  - テーブルをまだ定義していない場合は、アクセスするデータについて AWS Glue クローラを実行するか、Athena を使用して、データベースと 1 つ以上のテーブルを定義します。
  - このチュートリアルでは、Registry of open data on AWS で利用できる NYC Taxi trips dataset に基づくテーブルを使用します。このチュートリアルでは、データベース名 `tripdb` とテーブル名 `nyctaxi` を使用します。

チュートリアルのステップ

ステップ 1: Okta アカウントを作成する

このチュートリアルでは、Okta を SAML ベースの ID プロバイダーとして使用します。Okta アカウントがまだない場合は、無料のアカウントを作成できます。Okta アカウントは、SAML 認証用の Okta アプリケーションを作成するために必要です。

Okta アカウントを作成する

1. Okta を使用するには、Okta Developer サインアップページに移動して、無料の Okta トライアルアカウントを作成します。デベロッパー版のサービスは、Okta が developer.okta.com/pricing で指定する制限まで無料です。
2. アクティベーション E メールを受け取ったら、アカウントをアクティブにします。

Okta ドメイン名が割り当てられます。このドメイン名は、参照用に保存しておきます。後ほど、Athena に接続する JDBC 文字列でこのドメイン名 (*<okta-idp-domain>*) を使用します。

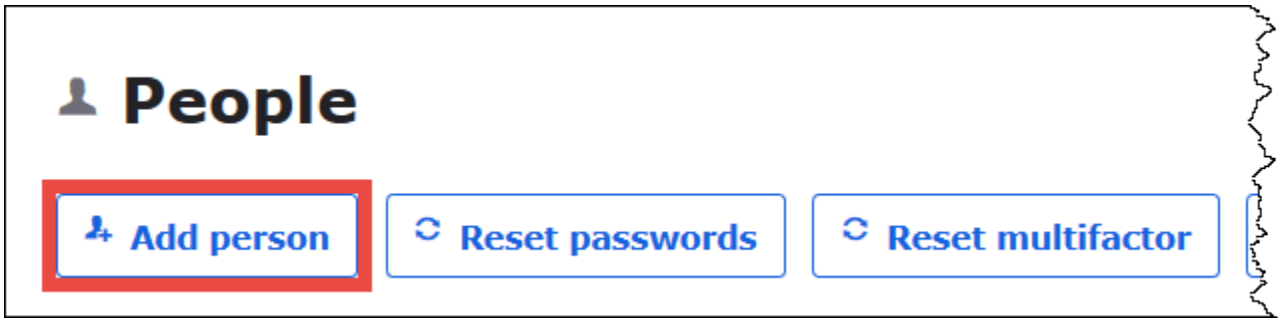
ステップ 2: Okta にユーザーとグループを追加する

このステップでは、Okta コンソールを使用して以下のタスクを実行します。

- 2 人の Okta ユーザーを作成する。
- 2 つの Okta グループを作成する。
- 各 Okta グループに Okta ユーザーを 1 人追加する。

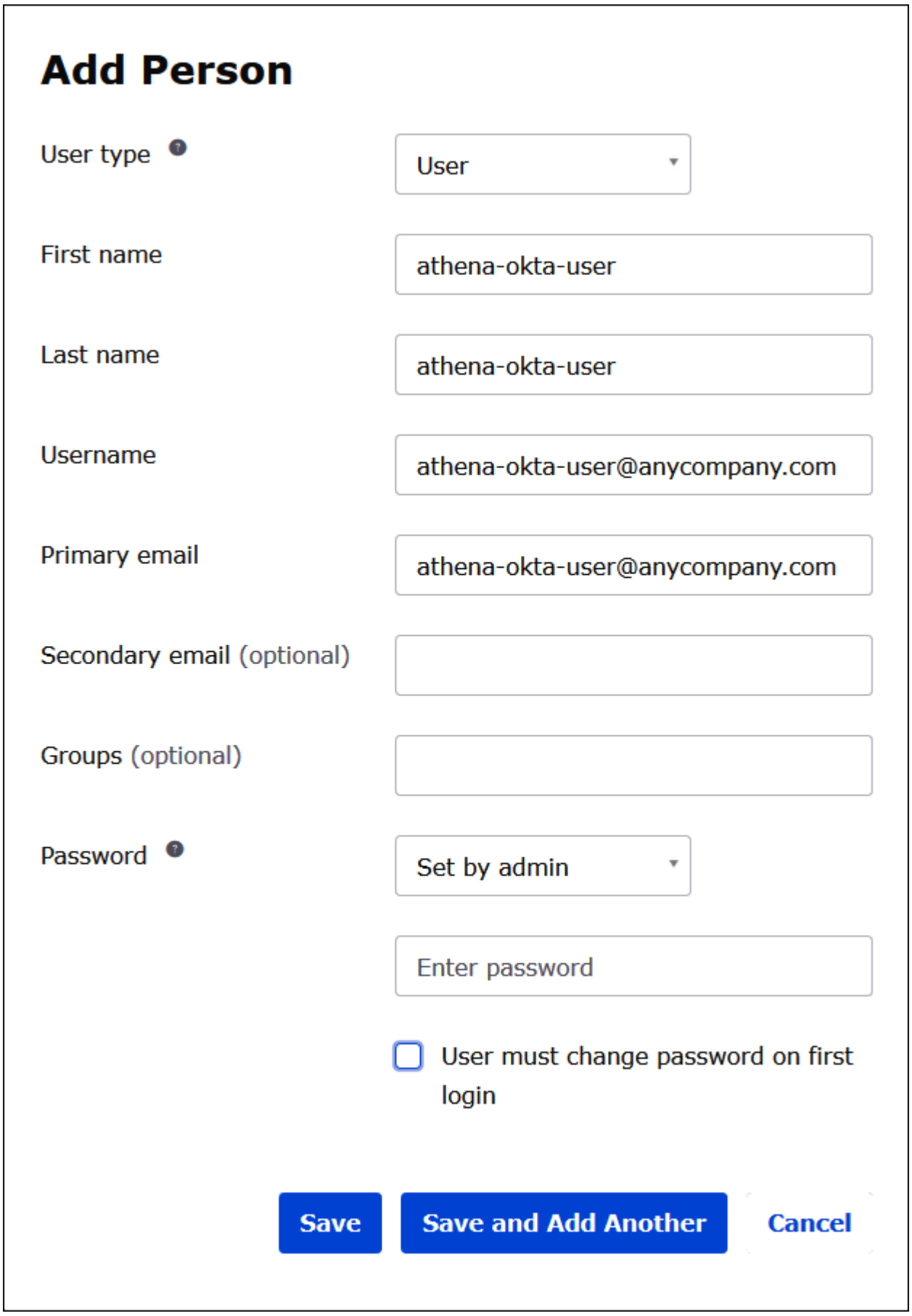
Okta にユーザーを追加する

1. Okta アカウントをアクティブにしたら、割り当てられた Okta ドメインに管理ユーザーとしてログインします。
2. 左側のナビゲーションペインで、[ディレクトリ] 、[ピープル] の順に選択します。
3. [Add Person] (人を追加する) を選択して、JDBC ドライバー経由で Athena にアクセスする新しいユーザーを追加します。

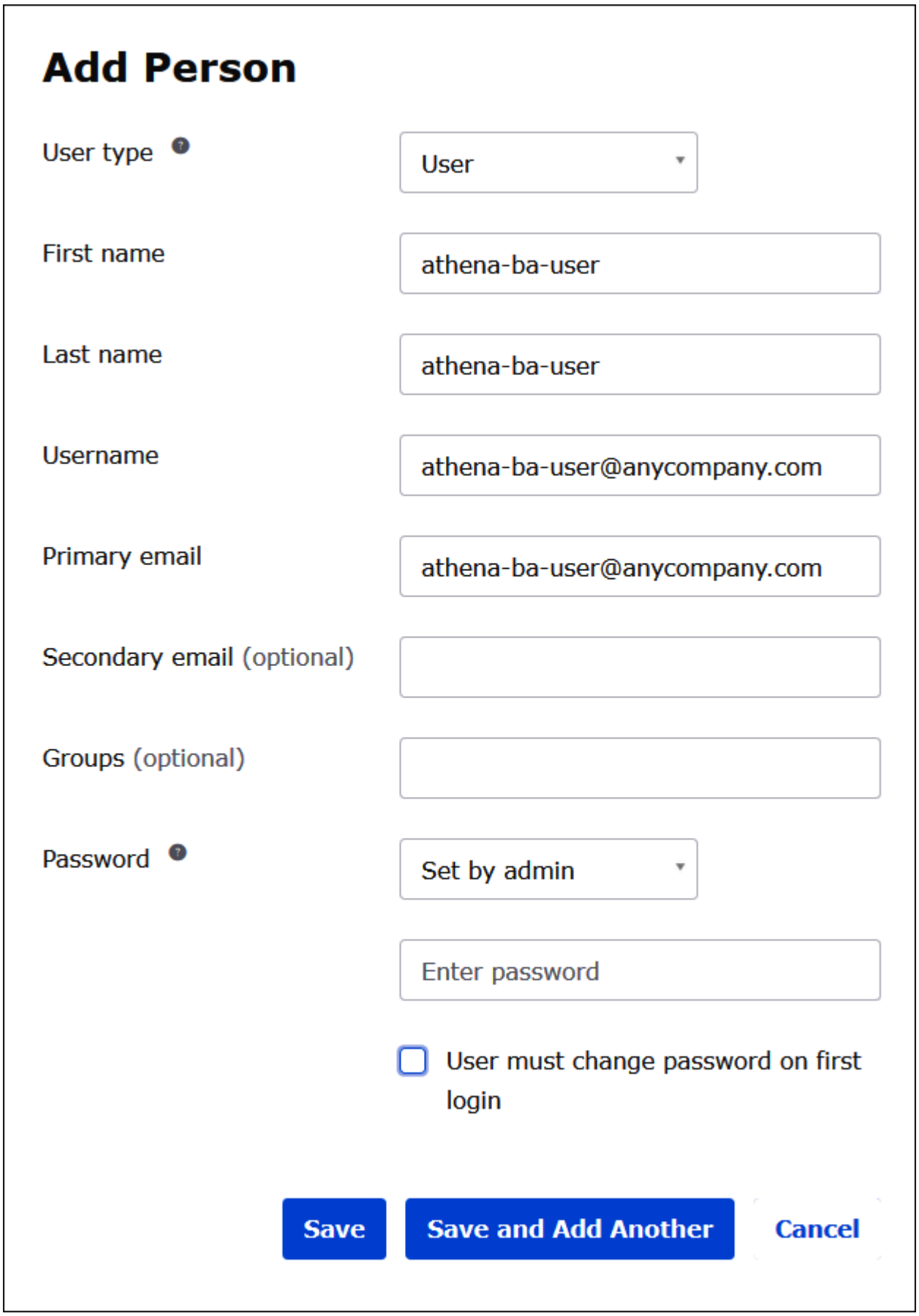


4. [Add Person] (人を追加する) ダイアログボックスで、必要な情報を入力します。
   - [First name] (名) と [Last name] (姓) に値を入力します。このチュートリアルでは *athena-okta-user* を使用します。
   - [Username] (ユーザー名) と [Primary email] (プライマリ E メール) を入力します。このチュートリアルでは *athena-okta-user@anycompany.com* を使用します。
   - [Password] (パスワード) には [Set by admin] (管理者が設定) を選択し、パスワードを入力します。このチュートリアルでは [User must change password on first login] (ユーザーは最初

の口グイン時にパスワードを変更する必要があります) オプションのチェックをオフにしますが、ユーザーによってはセキュリティ要件が異なる場合があります。


Add Person
User type
User
First name
athena-okta-user
Last name
athena-okta-user
Username
athena-okta-user@anycompany.com
Primary email
athena-okta-user@anycompany.com
Secondary email (optional)
Groups (optional)
Password
Set by admin
Enter password
User must change password on first login
Save
Save and Add Another
Cancel

5. [Save and Add Another] (保存して別の人を追加する) を選択します。
6. 別のユーザーの情報を入力します。この例では、ビジネスアナリストユーザーの *athena-ba-user@anycompany.com* を追加します。

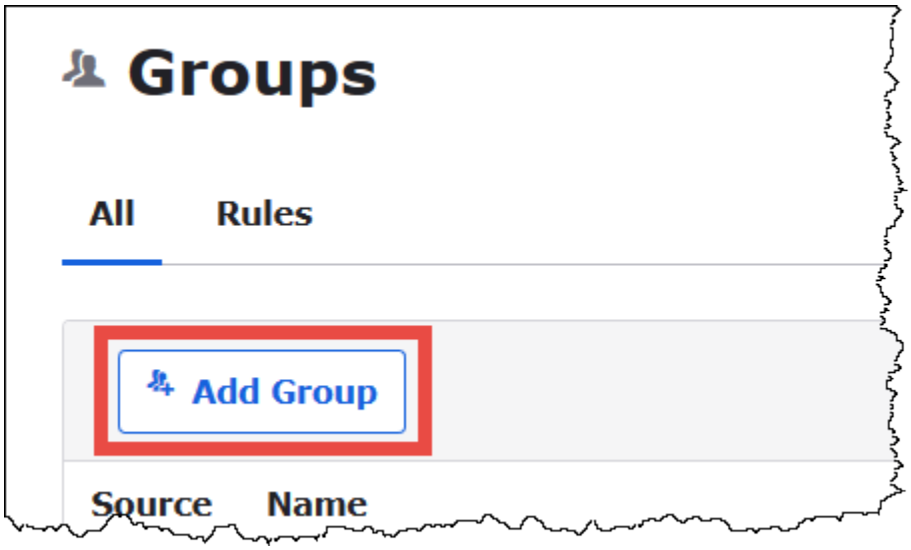

Add Person
User type
User
First name
athena-ba-user
Last name
athena-ba-user
Username
athena-ba-user@anycompany.com
Primary email
athena-ba-user@anycompany.com
Secondary email (optional)
Groups (optional)
Password
Set by admin
Enter password
User must change password on first login
Save
Save and Add Another
Cancel

7. [Save] (保存) を選択します。

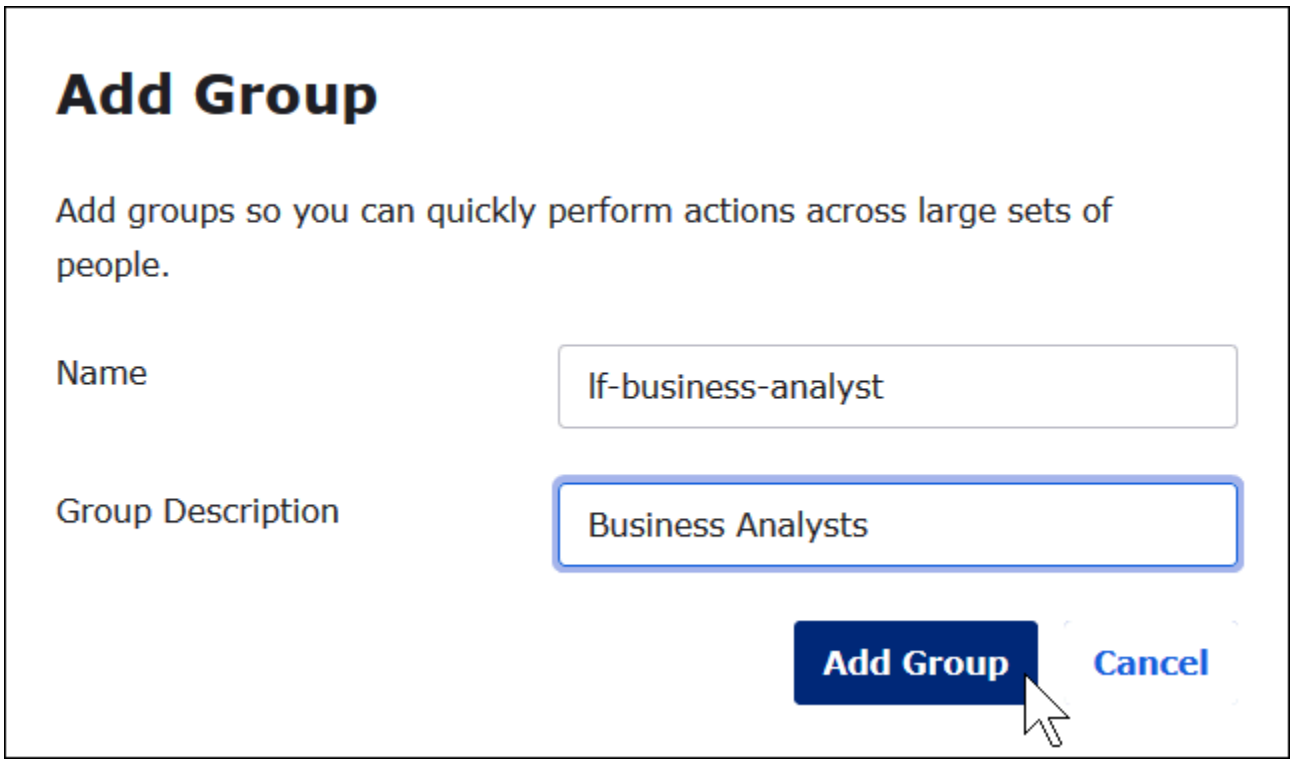
以下の手順では、「Business Analysts」グループと「Developer」グループを追加することによって、Athena JDBC ドライバー経由で 2 つの Okta グループのアクセス権を提供します。

Okta グループを追加する

1. Okta ナビゲーションペインで、[ディレクトリ] 、[グループ] の順に選択します。
2. [Groups] (グループ) ページで [Add Group] (グループの追加) をクリックします。



3. [Add Group] (グループの追加) ダイアログボックスで、必要な情報を入力します。
   - [Name] (名前) には *lf-business-analyst* を入力します。
   - [Group Description] (グループの説明) には *Business Analysts* を入力します。

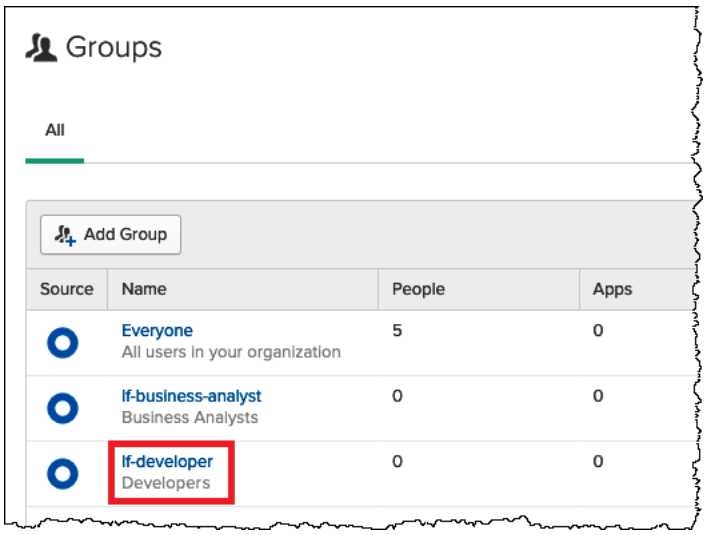


4. [Add Group] (グループの追加) を選択します。
5. [Groups] (グループ) ページで、もう一度 [Add Group] (グループの追加) を選択します。今回は、Developer グループの情報を入力します。
6. 必要な情報を入力します。
   - [Name] (名前) には *lf-developer* を入力します。
   - [Group Description] (グループの説明) には *[Developers]* を入力します。
7. [Add Group] (グループの追加) を選択します。

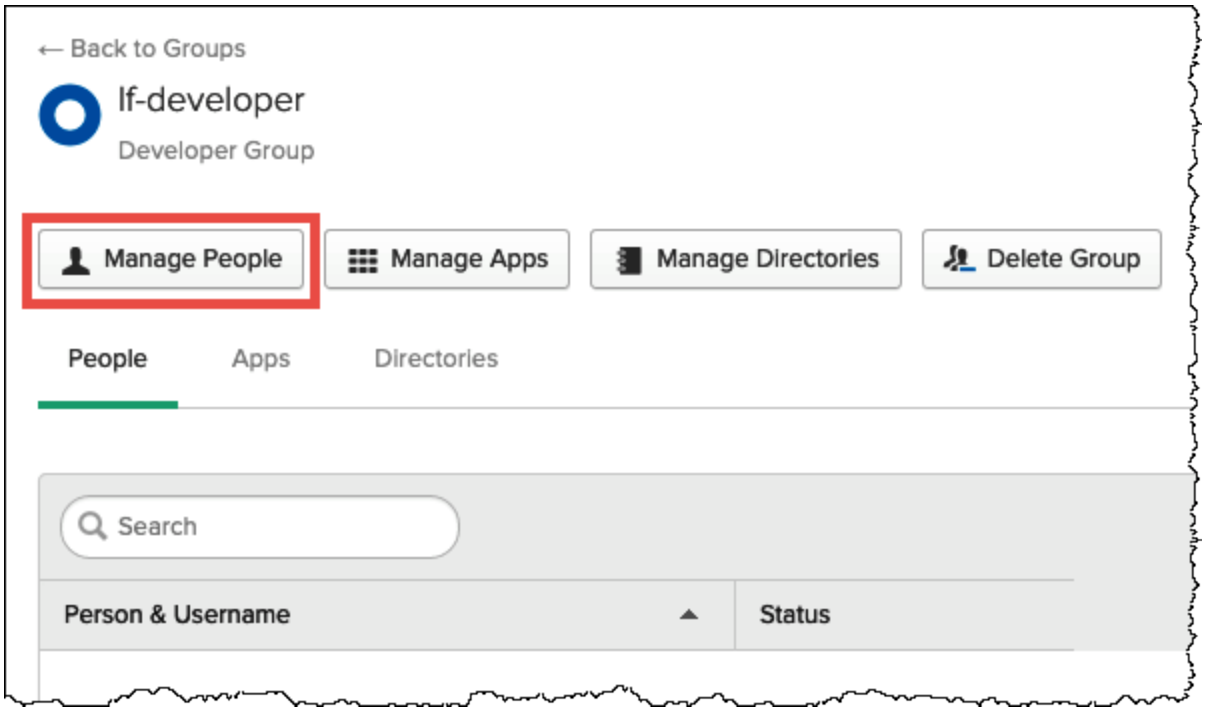
2 人のユーザーと 2 つのグループを作成したところで、各グループにユーザーを追加する準備が整いました。

ユーザーをグループに追加する

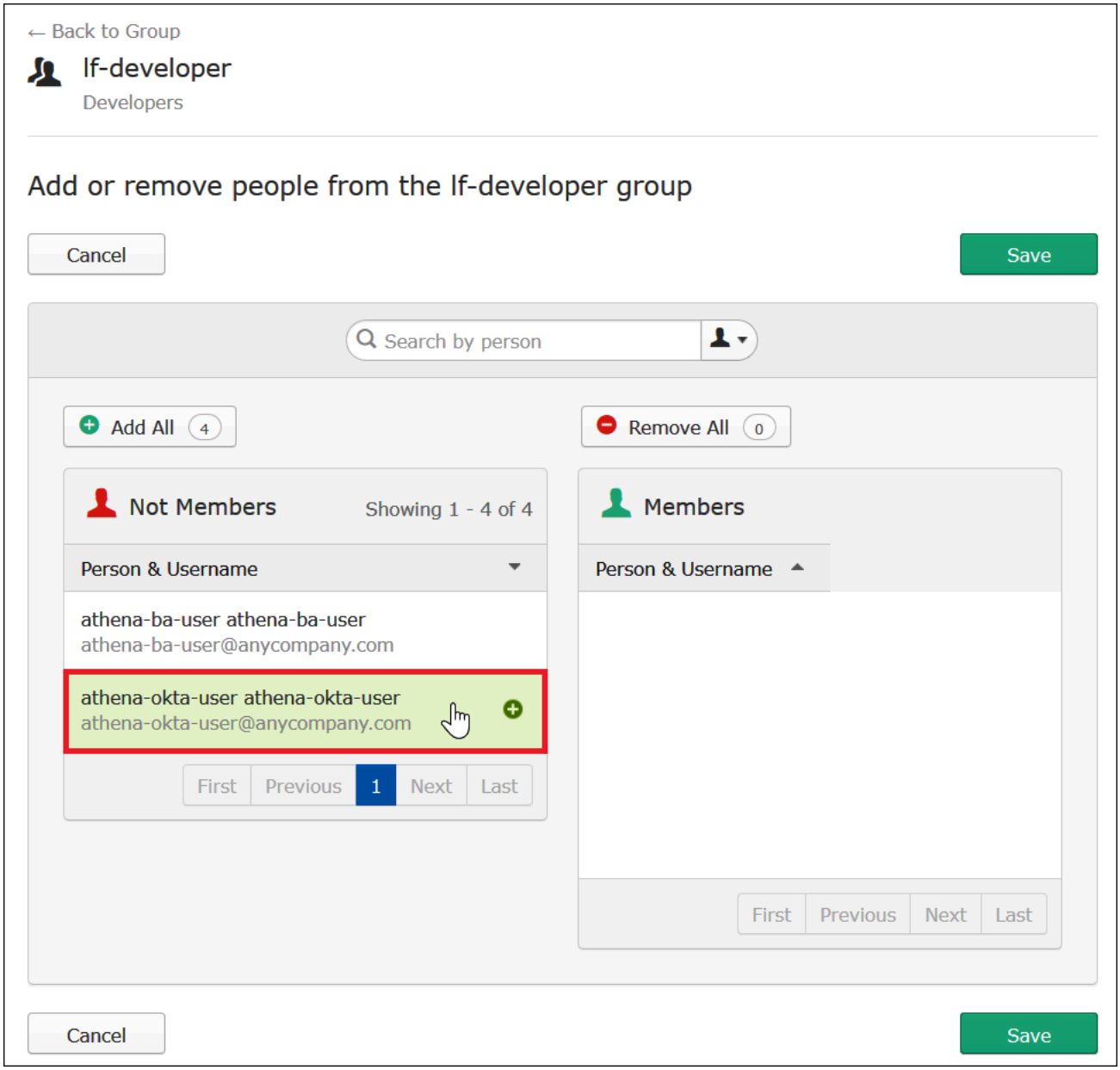
1. [Groups] (グループ) ページで、先ほど作成した lf-developer グループを選択します。このグループに、デベロッパーとして作成した Okta ユーザーの 1 人を追加します。



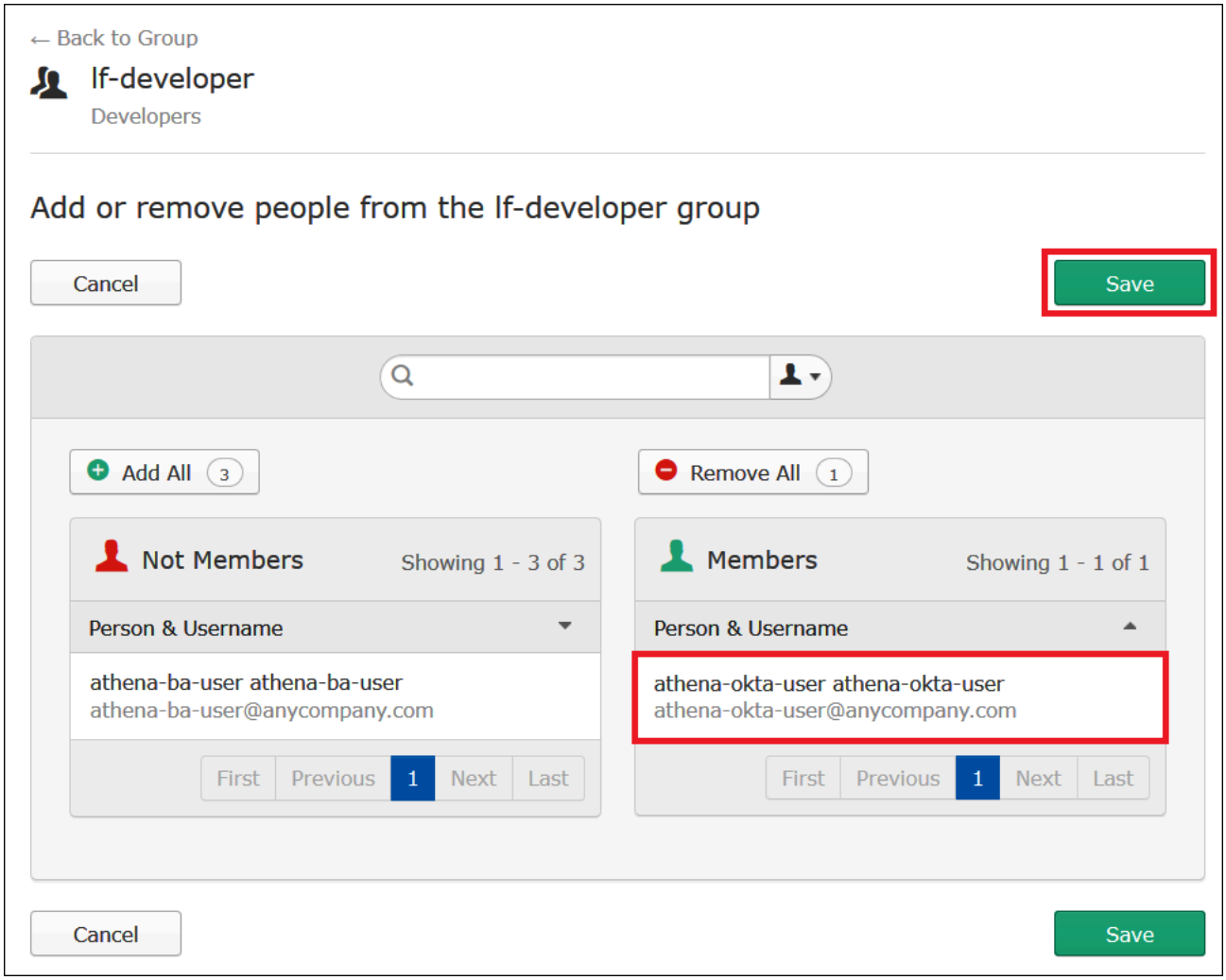
2. [Manage People] (ピープルの管理) をクリックします。



3. [Not Members] (非メンバー) リストで athena-okta-user を選択します。

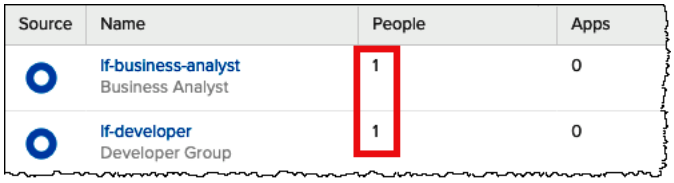


このユーザーのエントリが、左側の [Not Members] (非メンバー) リストから右側の [Members] (メンバー) リストに移動します。



4. [Save] (保存) を選択します。
5. [グループに戻る] をクリック、または [ディレクトリ] 、[グループ] の順に選択します。
6. lf-business-analyst グループを選択します。
7. [Manage People] (ピープルの管理) をクリックします。
8. lf-business-analyst グループの [Members] (メンバー) リストに athena-ba-user を追加して、[Save] (保存) をクリックします。
9. [グループに戻る] をクリック、または [ディレクトリ]、[グループ] の順に選択します。

これで、各グループに 1 人の Okta ユーザーがいることが [Groups] (グループ) ページに表示されるようになりました。



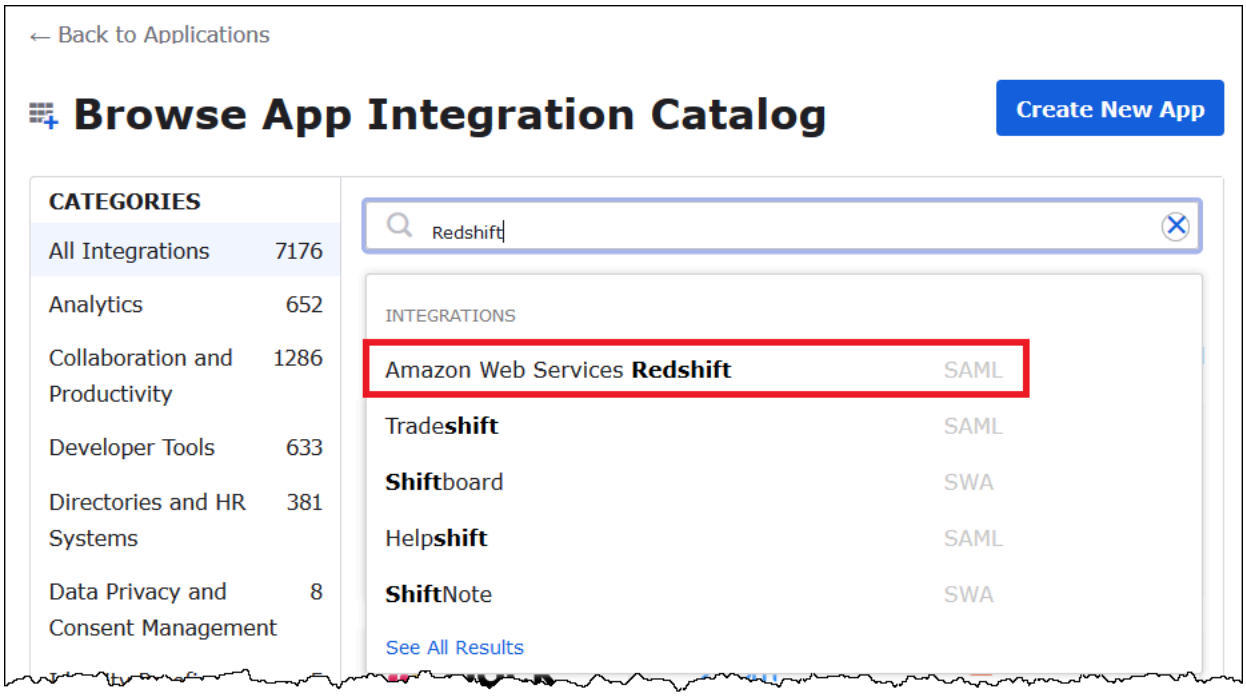
ステップ 3: SAML 認証用の Okta アプリケーションをセットアップする

このステップでは、Okta Developer コンソールを使用して、以下のタスクを実行します。

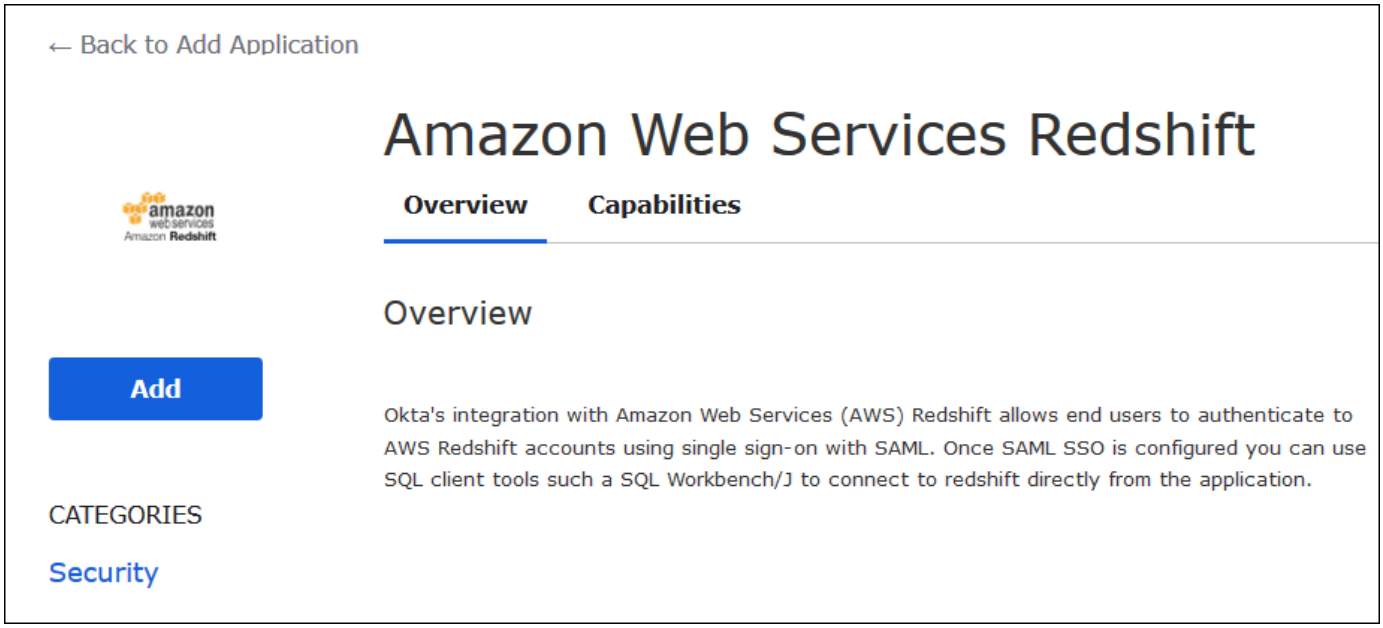
- AWS で使用するための SAML アプリケーションを追加する。
- アプリケーションを Okta ユーザーに割り当てる。
- アプリケーションを Okta グループに割り当てる。
- 後ほど AWS で使用するために、結果として得た ID プロバイダーメタデータをダウンロードする。

SAML 認証用のアプリケーションを追加する

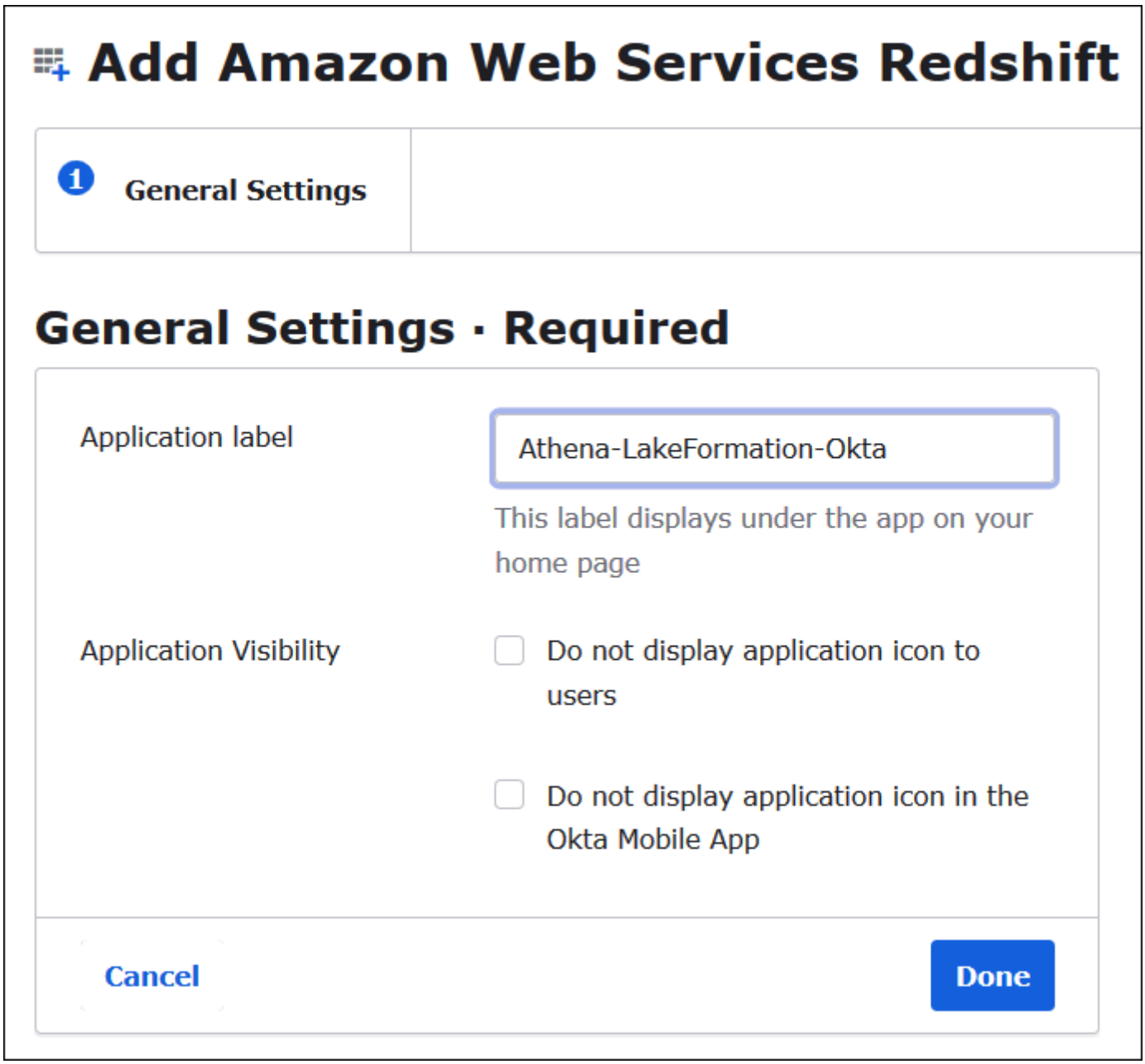
1. Okta ナビゲーションペインで、[アプリケーション] 、[アプリケーション] の順に選択して、Athena に SAML 認証用の Okta アプリケーションを設定できるようにします。
2. [アプリケーションカタログを参照] をクリックします。
3. 検索ボックスに「**`Redshift`**」と入力します。
4. [Amazon Web Services Redshift] を選択します。このチュートリアルの Okta アプリケーションは、Amazon Redshift の既存の SAML 統合を使用します。



5. [Amazon Web Services Redshift] ページで [Add] (追加) をクリックして、Amazon Redshift 用の SAML ベースのアプリケーションを作成します。



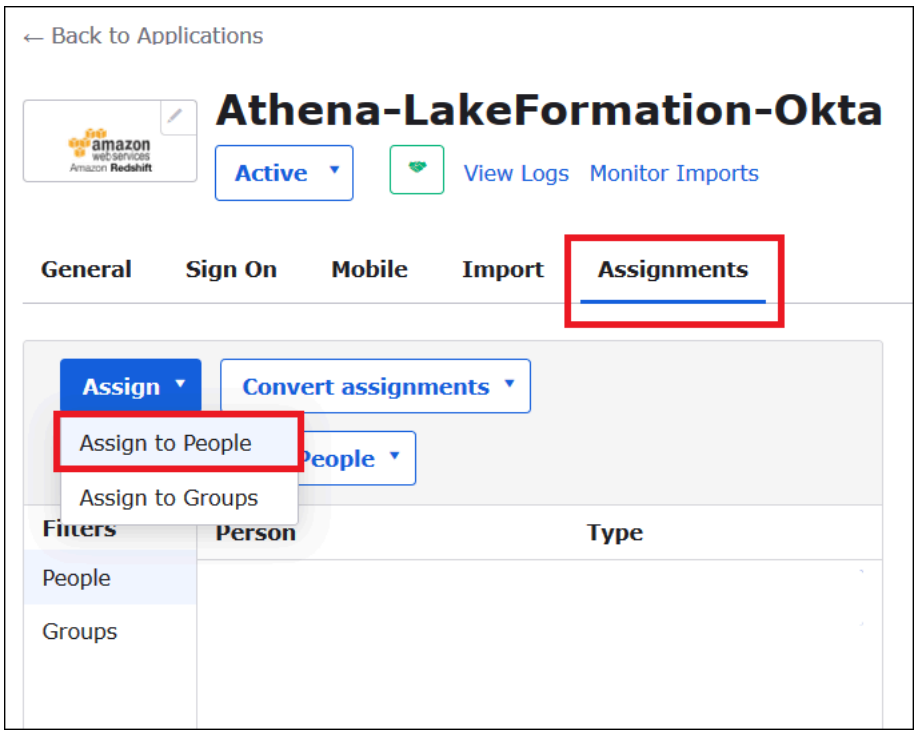
6. [Application label] (アプリケーションラベル) に `Athena-LakeFormation-Okta` を入力し、[Done] (完了) を選択します。



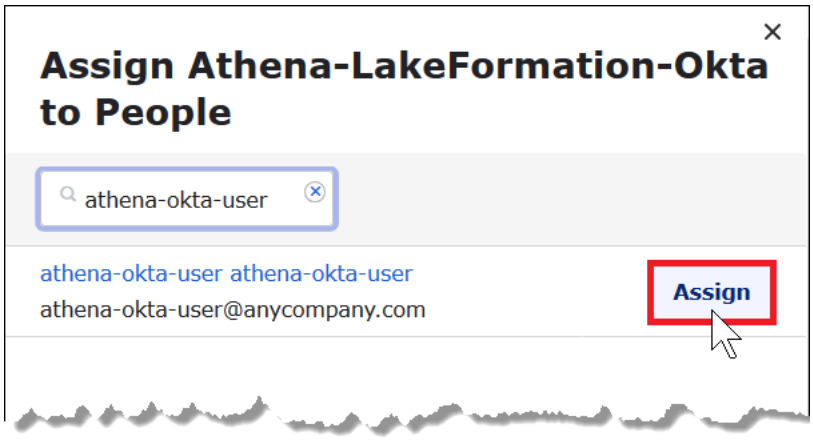
これで Okta アプリケーションが作成されたので、それを作成したユーザーとグループに割り当てることができます。

アプリケーションをユーザーとグループに割り当てる

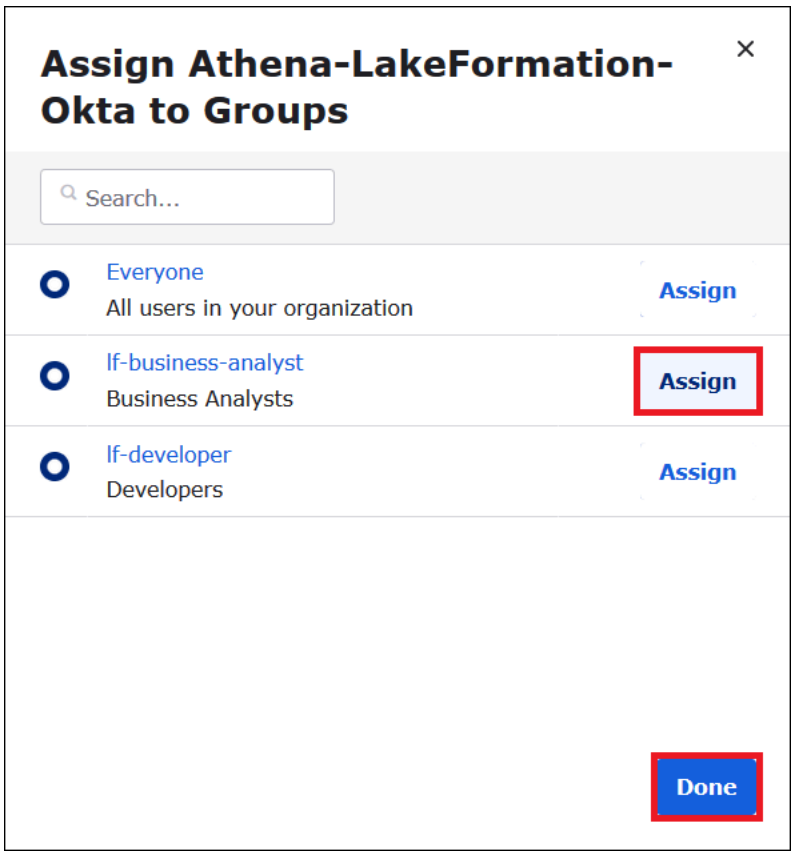
1. [アプリケーション] ページで、Athena-LakeFormation-Okta アプリケーションを選択します。
2. [割り当て] タブで、[割り当てる] 、[ピープルに割り当てる] を順に選択します。



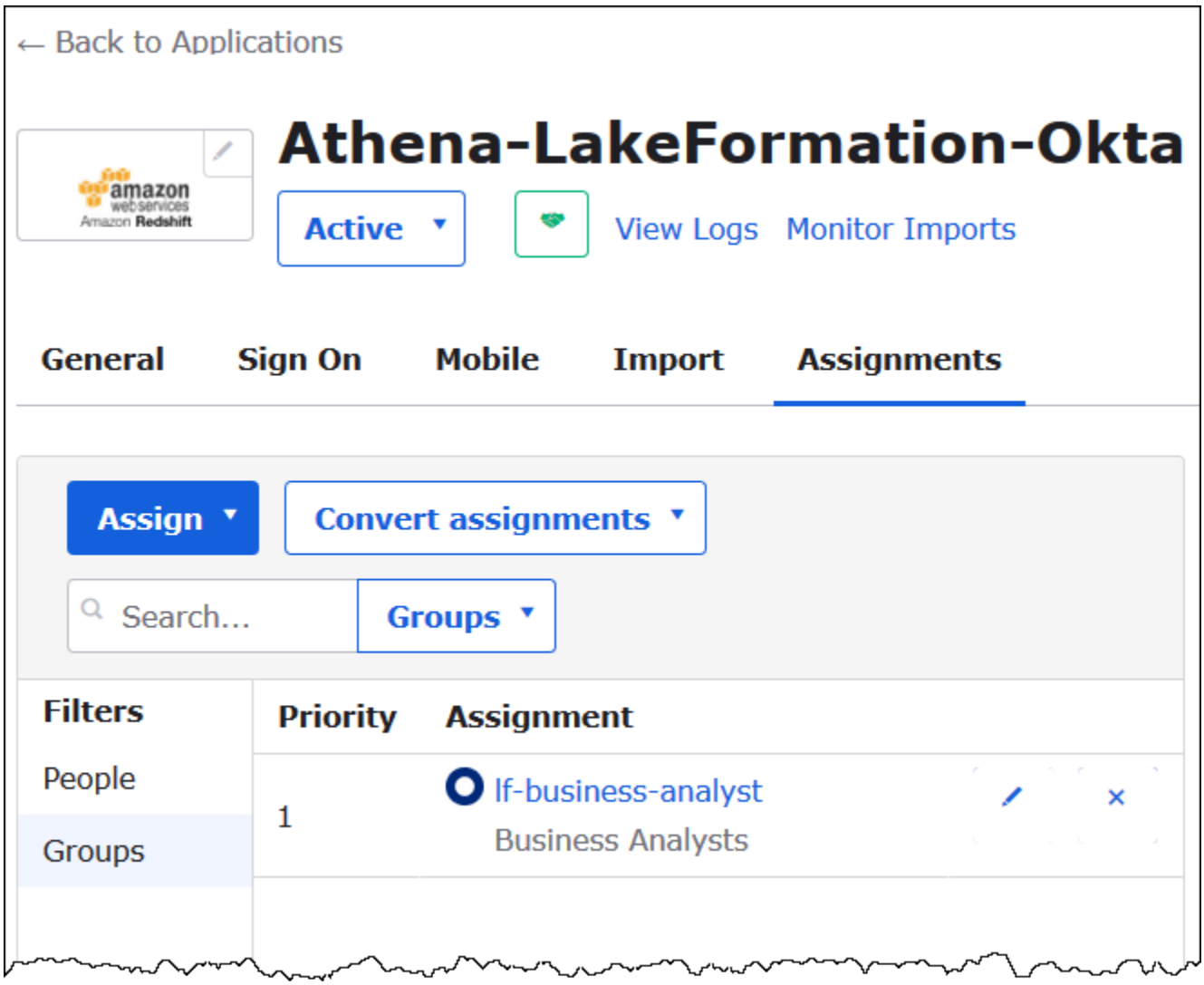
3. [Assign Athena-LakeFormation-Okta to People] (Athena-LakeFormation-Okta をピープルに割り当てる) ダイアログボックスで、先ほど作成した athena-okta-user ユーザーを見つけます。
4. [Assign] (割り当てる) を選択してユーザーをアプリケーションに割り当てます。



5. [Save and Go Back] (保存して戻る) を選択します。
6. [完了] をクリックします。
7. Athena-LakeFormation-Okta アプリケーションの [Assignments] (割り当て) タブで、[Assign] (割り当てる) を選択し、[Assign to Groups] (グループに割り当てる) を選択します。
8. lf-business-analyst の [Assign] (割り当てる) を選択して、Athena-LakeFormation-Okta アプリケーションを lf-business-analyst グループに割り当ててから、[Done] (完了) を選択します。



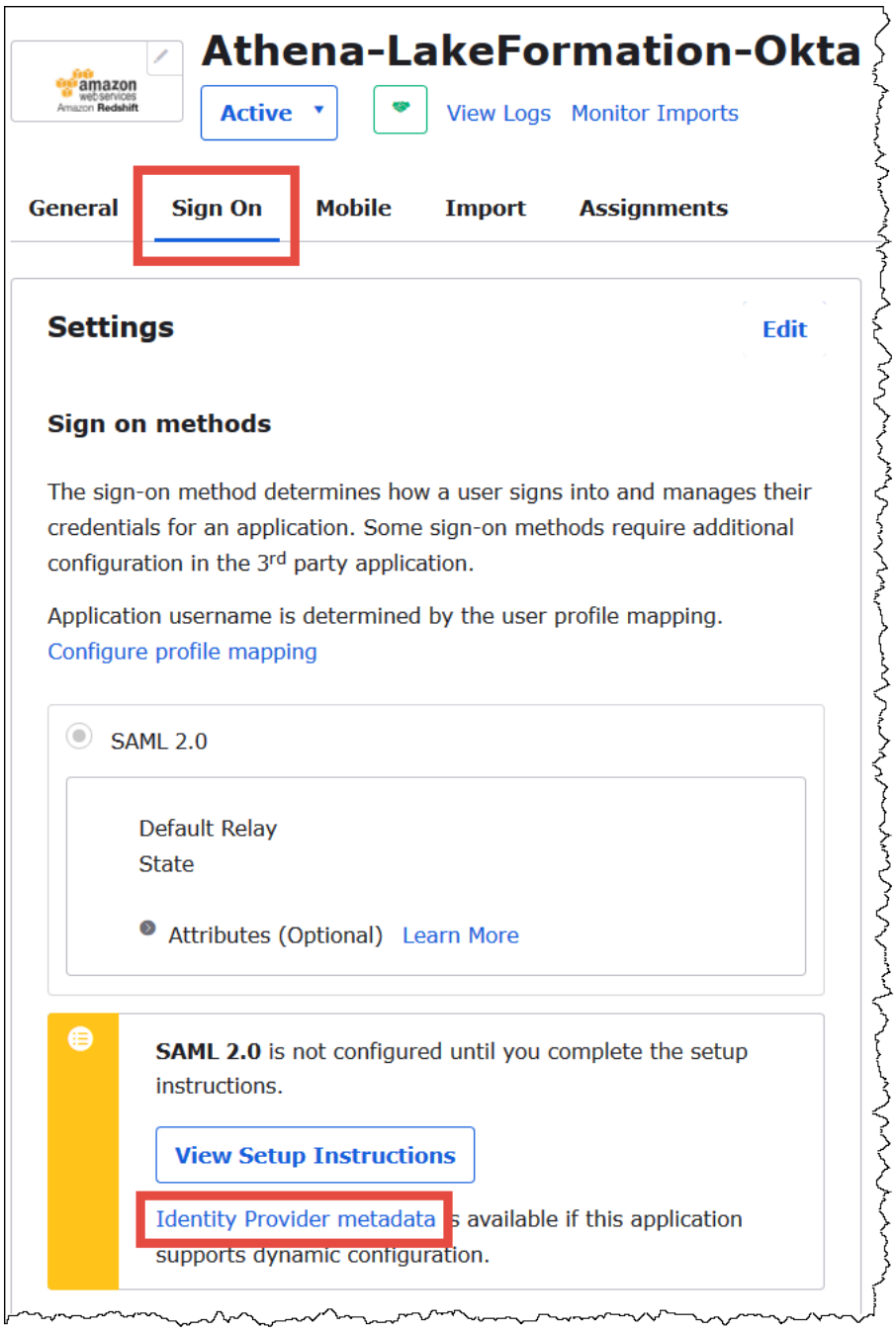
アプリケーションのグループのリストにこのグループが表示されます。



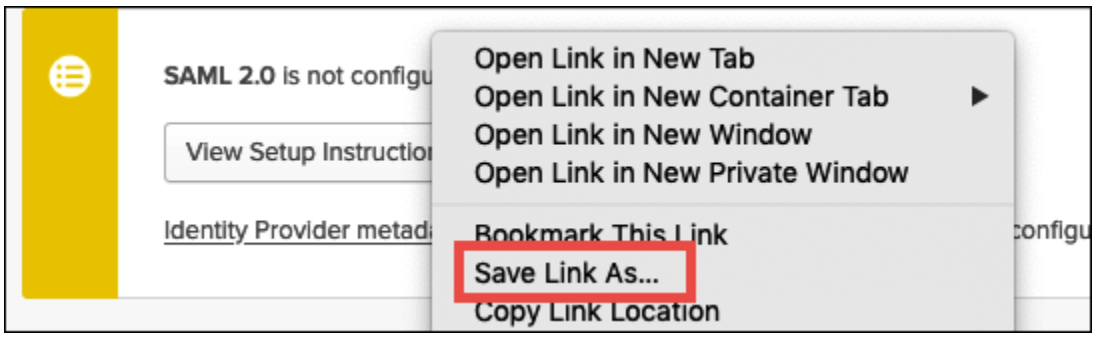
これで、AWS での使用のために ID プロバイダーアプリケーションのメタデータをダウンロードする準備ができました。

アプリケーションのメタデータをダウンロードする

1. Okta アプリケーションの [Sign On] (サインオン) タブを選択してから、[Identity Provider metadata] (ID プロバイダーのメタデータ) を右クリックします。



2. [Save Link As] (名前を付けてリンクを保存する) を選択して、XML 形式になっている ID プロバイダーのメタデータをファイルに保存します。これには、分かりやすい名前を付けてください (`Athena-LakeFormation-idp-metadata.xml` など)。



ステップ 4: AWS SAML ID プロバイダーと Lake Formation アクセス IAM ロールを作成する

このステップでは、AWS Identity and Access Management (IAM) コンソールを使用して、以下のタスクを実行します。

- AWS 用の ID プロバイダーを作成する。
- Lake Formation アクセス用の IAM ロールを作成する。
- AmazonAthenaFullAccess マネージドポリシーをロールに追加する。
- Lake Formation および AWS Glue 用のポリシーをロールに追加する。
- Athena のクエリ結果用のポリシーをロールに追加する。

AWS SAML ID プロバイダーを作成する

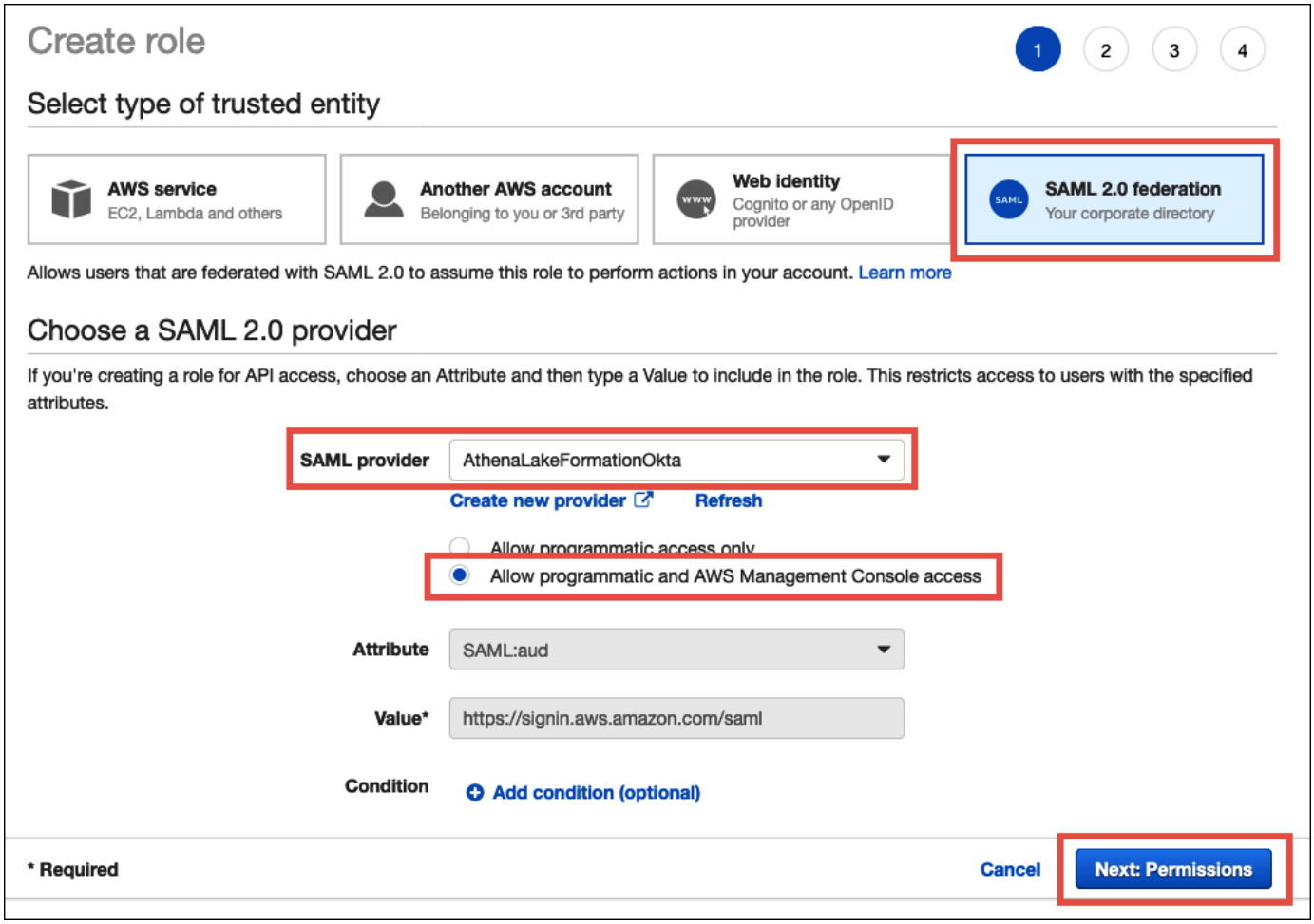
1. Amazon Web Services アカウントコンソールに Amazon Web Services アカウント管理者としてサインインし、IAM コンソール (https://console.aws.amazon.com/iam/) に移動します。
2. ナビゲーションペインで、[ID プロバイダー] を選択し、[プロバイダーを追加] をクリックします。
3. [プロバイダーの設定] 画面で、次の情報を入力します。
   - [プロバイダーのタイプ] では [SAML] を選択します。
   - [プロバイダー名] では `AthenaLakeFormationOkta` を入力します。
   - [メタデータドキュメント] では、[ファイルを選択] オプションを使用して、ダウンロードした ID プロバイダー (IdP) メタデータ XML ファイルをアップロードします。
4. [プロバイダーを追加] をクリックします。

次に、AWS Lake Formation アクセス用の IAM ロールを作成します。このロールには、2 つのインラインポリシーを追加します。1 つのポリシーで Lake Formation と AWS Glue API へのアクセス許可

を提供できます。もう 1 つのポリシーは、Athena と、Amazon S3 にある Athena のクエリ結果の場所へのアクセス権を提供します。

AWS Lake Formation にアクセスするための IAM ロールを作成するには

1. IAM コンソールのナビゲーションペインで、[Roles] (ロール)、[Create role] (ロールの作成) の順にクリックします。
2. [Create role] (ロールの作成) ページで、以下のステップを実行します。



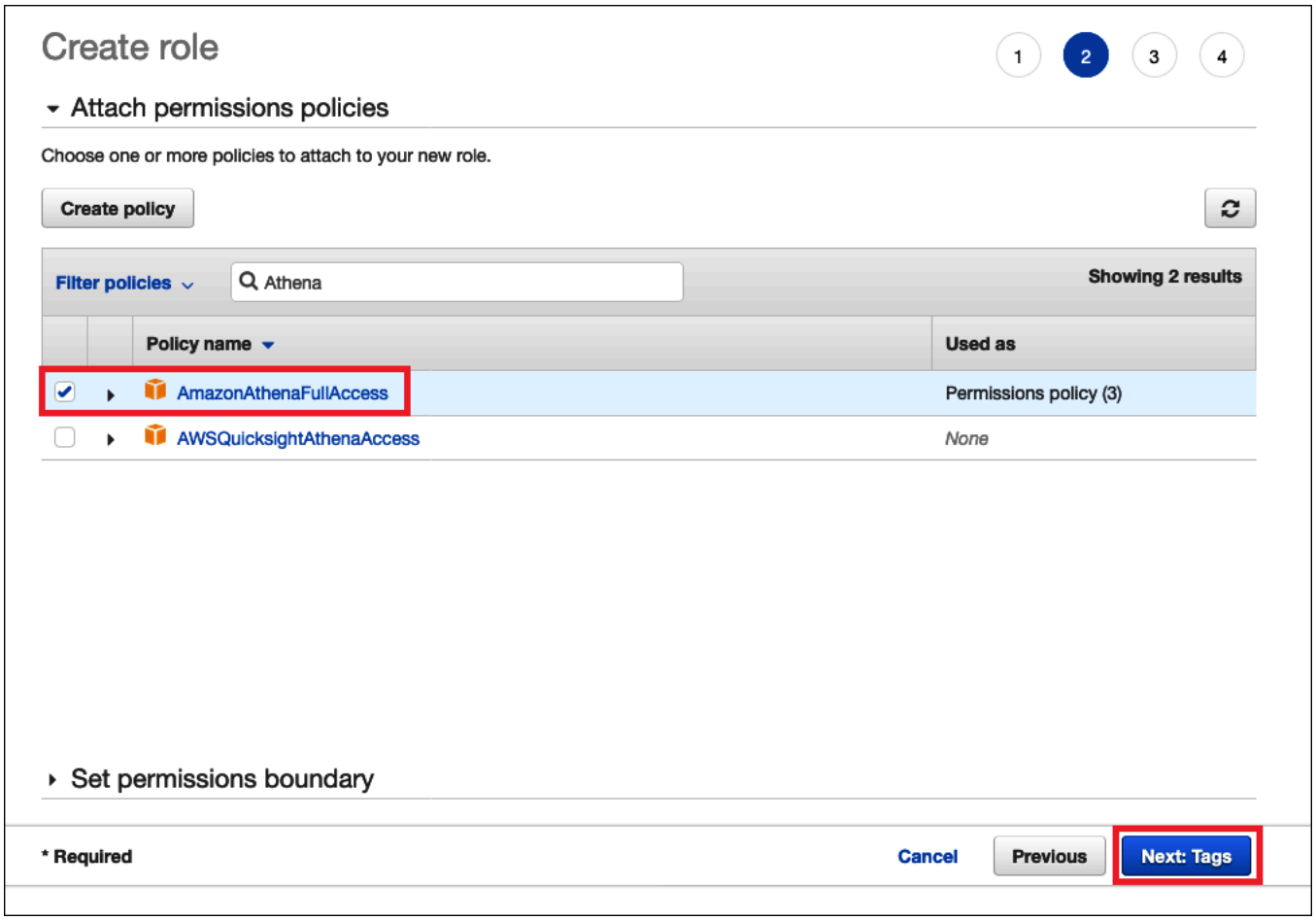
a. [Select type of trusted entity] (信頼されたエンティティのタイプを選択) で、[SAML 2.0 Federation] (SAML 2.0 フェデレーション) を選択します。

b. [SAML provider] (SAML プロバイダー) には AthenaLakeFormationOkta を選択します。

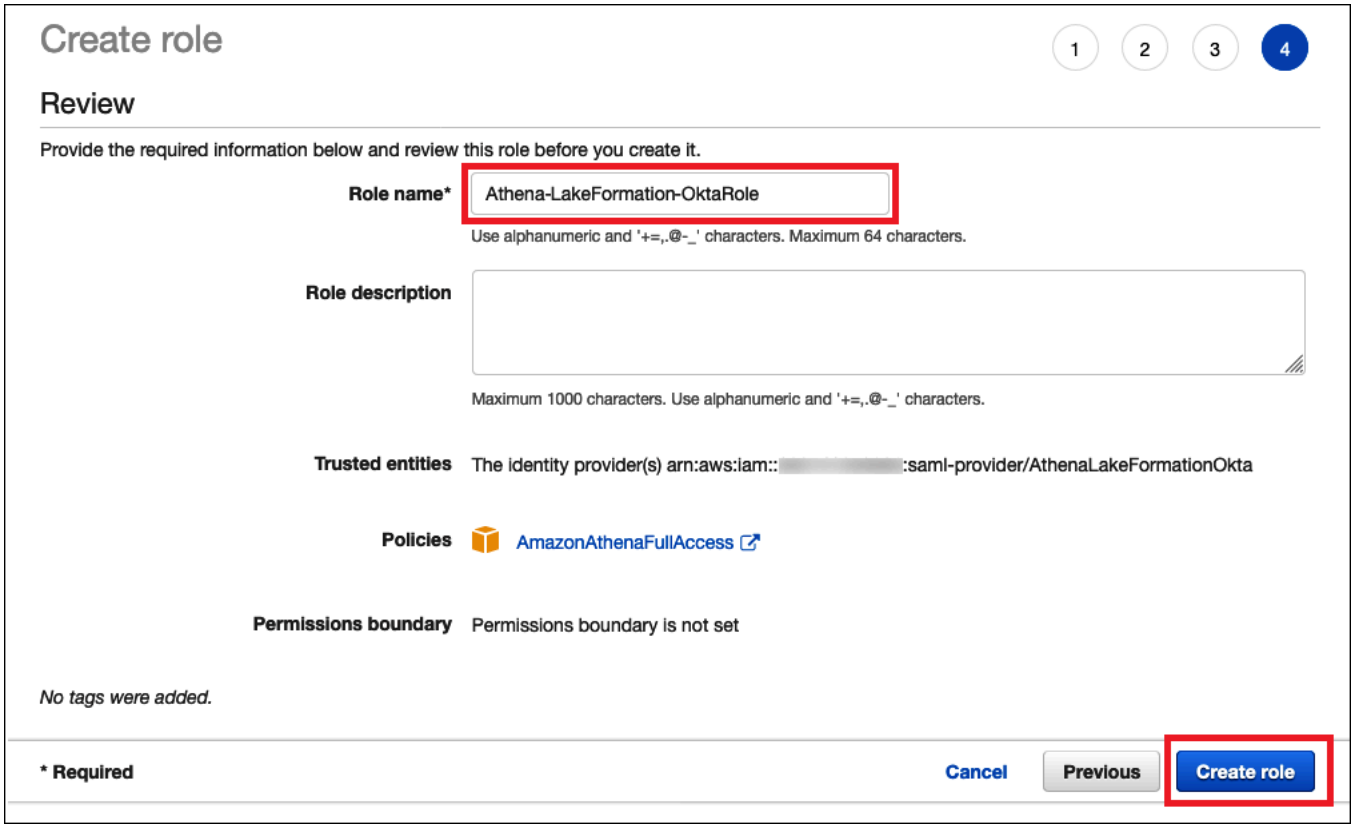
c. [SAML プロバイダー] で、[プログラムによるアクセスと AWS Management Console によるアクセスを許可] オプションを選択します。

d. Next: Permissions (次のステップ: 許可) を選択します。

3. [Attach Permissions policies] (許可ポリシーのアタッチ) ページにあるポリシーの [Filter] (フィルター) に **`Athena`** を入力します。
4. [AmazonAthenaFullAccess] マネージドポリシーを選択してから、[Next: Tags] (次のステップ: タグ) を選択します。



5. [タグの追加] ページで、[次へ: レビュー] を選択します。
6. [Review] (確認) ページで、[Role name] (ロール名) にロールの名前 (*`Athena-LakeFormation-OktaRole`* など) を入力してから [Create role] (ロールの作成) を選択します。



次に、Lake Formation、AWS Glue API、および Simple Storage Service (Amazon S3) 内の Athena のクエリ結果へのアクセスを許可するインラインポリシーを追加します。

IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

Lake Formation および AWS Glue に対するロールにインラインポリシーを追加する

1. IAM コンソールのロールのリストから、新しく作成した `Athena-LakeFormation-OktaRole` を選択します。
2. ロールの [Summary] (概要) ページにある [Permissions] (許可) タブで、[Add inline policy] (インラインポリシーの追加) を選択します。
3. [Create policy] (ポリシーの作成) ページで [JSON] を選択します。
4. Lake Formation と AWS Glue API へのアクセス権を提供する以下のようなインラインポリシーを追加します。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess",
            "glue:GetTable",
            "glue:GetTables",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:CreateDatabase",
            "glue:GetUserDefinedFunction",
            "glue:GetUserDefinedFunctions"
        ],
        "Resource": "*"
    }
}
```

5. [Review policy] (ポリシーの確認) をクリックします。
6. [Name] (名前) にポリシーの名前を入力します (**LakeFormationGlueInlinePolicy** など)。
7. [Create policy] を選択します。

Athena のクエリ結果の場所に対するロールにインラインポリシーを追加する

1. `Athena-LakeFormation-OktaRole` ロールの [Summary] (概要) ページにある [Permissions] (許可) タブで、[Add inline policy] (インラインポリシーの追加) を選択します。
2. [Create policy] (ポリシーの作成) ページで [JSON] を選択します。
3. Athena のクエリ結果の場所へのロールアクセスを許可する以下のようなインラインポリシーを追加します。例にある *<athena-query-results-bucket>* プレースホルダーを Simple Storage Service (Amazon S3) バケットの名前に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AthenaQueryResultsPermissionsForS3",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:PutObject",
                "s3:GetObject"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::<athena-query-results-bucket>",
                "arn:aws:s3:::<athena-query-results-bucket>/*"
            ]
        }
    ]
}
```

4. [Review policy] (ポリシーの確認) をクリックします。
5. [Name] (名前) にポリシーの名前を入力します (**AthenaQueryResultsInlinePolicy** など)。
6. [Create policy] を選択します。

次に、Lake Formation アクセスロールの ARN と、作成した SAML プロバイダーの ARN をコピーします。これらは、チュートリアルの次のセクションで Okta SAML アプリケーションを設定するときに必要です。

ロール ARN と SAML ID プロバイダー ARN をコピーする

1. IAM コンソールで、`Athena-LakeFormation-OktaRole` ロールの [Summary] (概要) ページにある [Role ARN] (ロール ARN) の横の [Copy to clipboard] (クリップボードにコピー) アイコンを選択します。ARN の形式は以下のようになっています。

   ```
   arn:aws:iam::<account-id>:role/Athena-LakeFormation-OktaRole
   ```
2. 後で参照できるように、完全な ARN を安全に保存しておきます。
3. IAM コンソールのナビゲーションペインで、[Identity providers] (ID プロバイダー) を選択します。
4. AthenaLakeFormationOkta プロバイダーを選択します。
5. [Summary] (概要) ページで、[Provider ARN] (プロバイダー ARN) の横にある [Copy to clipboard] (クリップボードにコピー) アイコンを選択します。ARN は次のようになります。

   ```
   arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta
   ```
6. 後で参照できるように、完全な ARN を安全に保存しておきます。

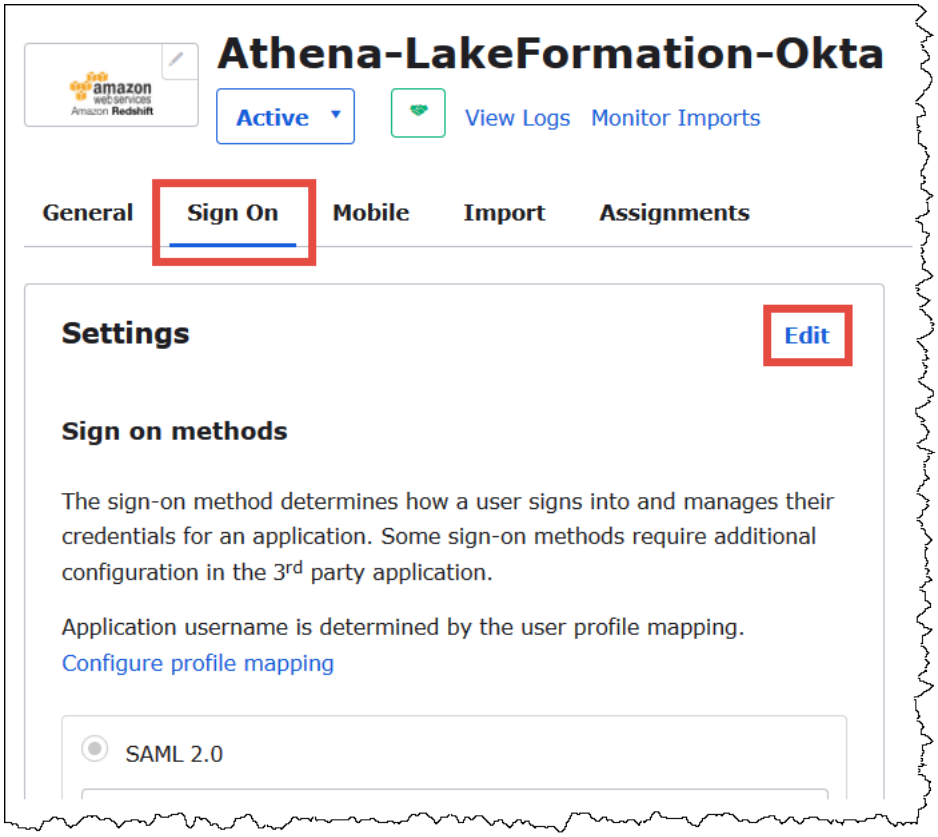
ステップ 5: IAM ロールと SAML ID プロバイダーを Okta アプリケーションに追加する

このステップでは、Okta Developer コンソールに戻り、以下のタスクを実行します。

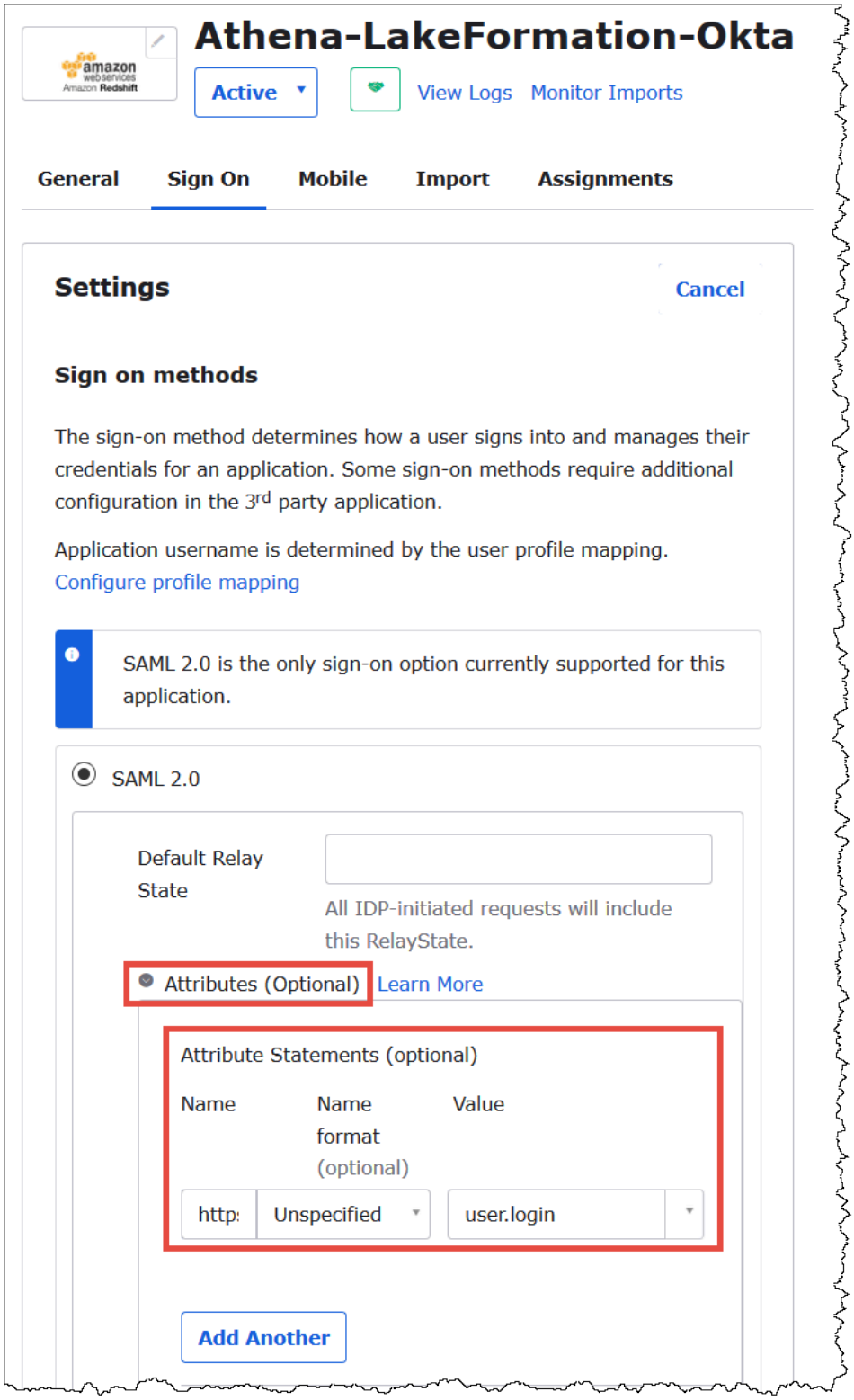
- ユーザーとグループの Lake Formation URL 属性を Okta アプリケーションに追加する。
- ID プロバイダーの ARN と IAM ロールの ARN を Okta アプリケーションに追加する。
- Okta アプリケーション ID をコピーする。Okta アプリケーション ID は、Athena に接続する JDBC プロファイルに必要です。

ユーザーとグループの Lake Formation URL 属性を Okta アプリケーションに追加する

1. Okta Developer コンソールにサインインします。
2. [Applications] (アプリケーション) タブを選択し、`Athena-LakeFormation-Okta` アプリケーションを選択します。
3. アプリケーションの [Sign On] (サインオン) タブを選択し、[Edit] (編集) を選択します。



4. [属性 (オプション)] を選択して展開します。



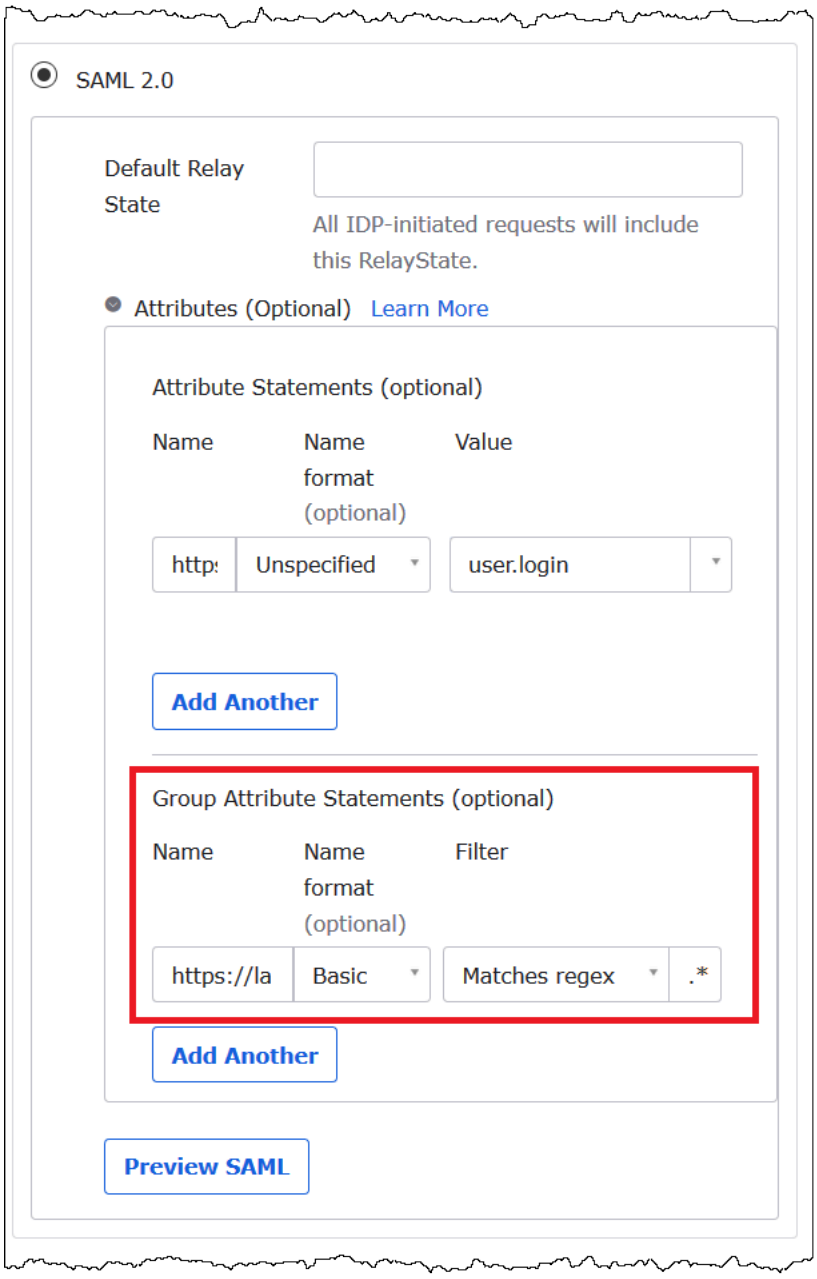
5. [属性ステートメント (オプション)] では、次の属性を追加します。

   - [Name] (名前) には **`https://lakeformation.amazon.com/SAML/Attributes/Username`** を入力します。

- [Value] (値) に「**user.login**」と入力します。

6. [Group Attribute Statements (optional)] (グループ属性ステートメント (オプション)) で、以下の属性を追加します。

   - [名前] に「**https://lakeformation.amazon.com/SAML/Attributes/Groups**」と入力します。
   - [Name format] (名前の形式) には **Basic** を入力します。
   - [Filter] (フィルター) には [Matches regex] (regex に一致) を選択し、フィルターボックスに **.*** を入力します。

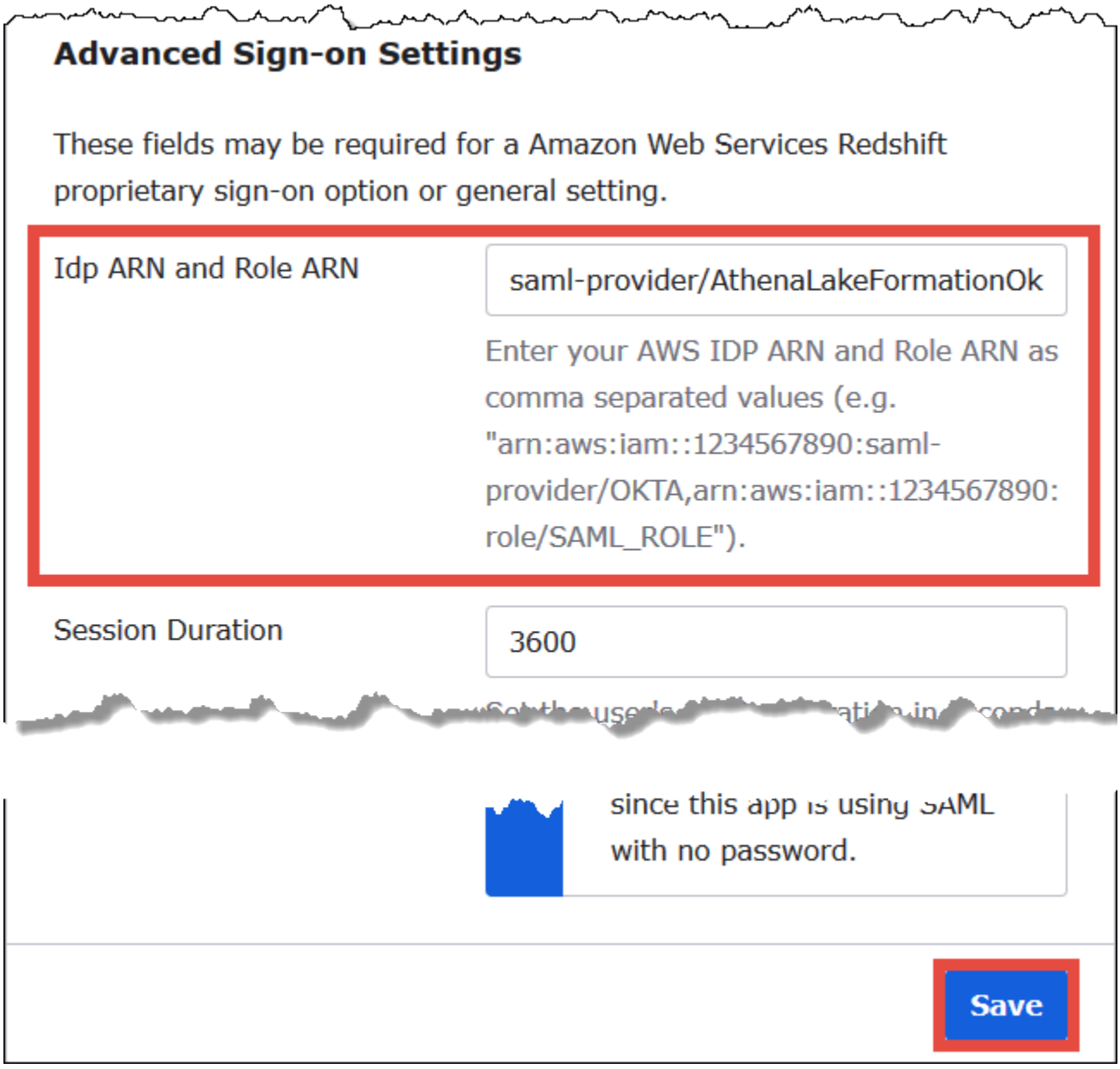


7. [Advanced Sign-On Settings] (高度なサインオン設定) までスクロールダウンします。ここで、Okta アプリケーションに ID プロバイダーと IAM ロールの ARN を追加します。

ID プロバイダーと IAM ロールの ARN を Okta アプリケーションに追加する

1. [Idp ARN とロール ARN] に、*<saml-arn>*、*<role-arn>* 形式の CSV として AWS ID プロバイダー ARN とロール ARN を入力します。組み合わされた文字列は以下のようになります。

```
arn:aws:iam::<account-id>:saml-provider/
AthenaLakeFormationOkta,arn:aws:iam::<account-id>:role/Athena-LakeFormation-
OktaRole
```

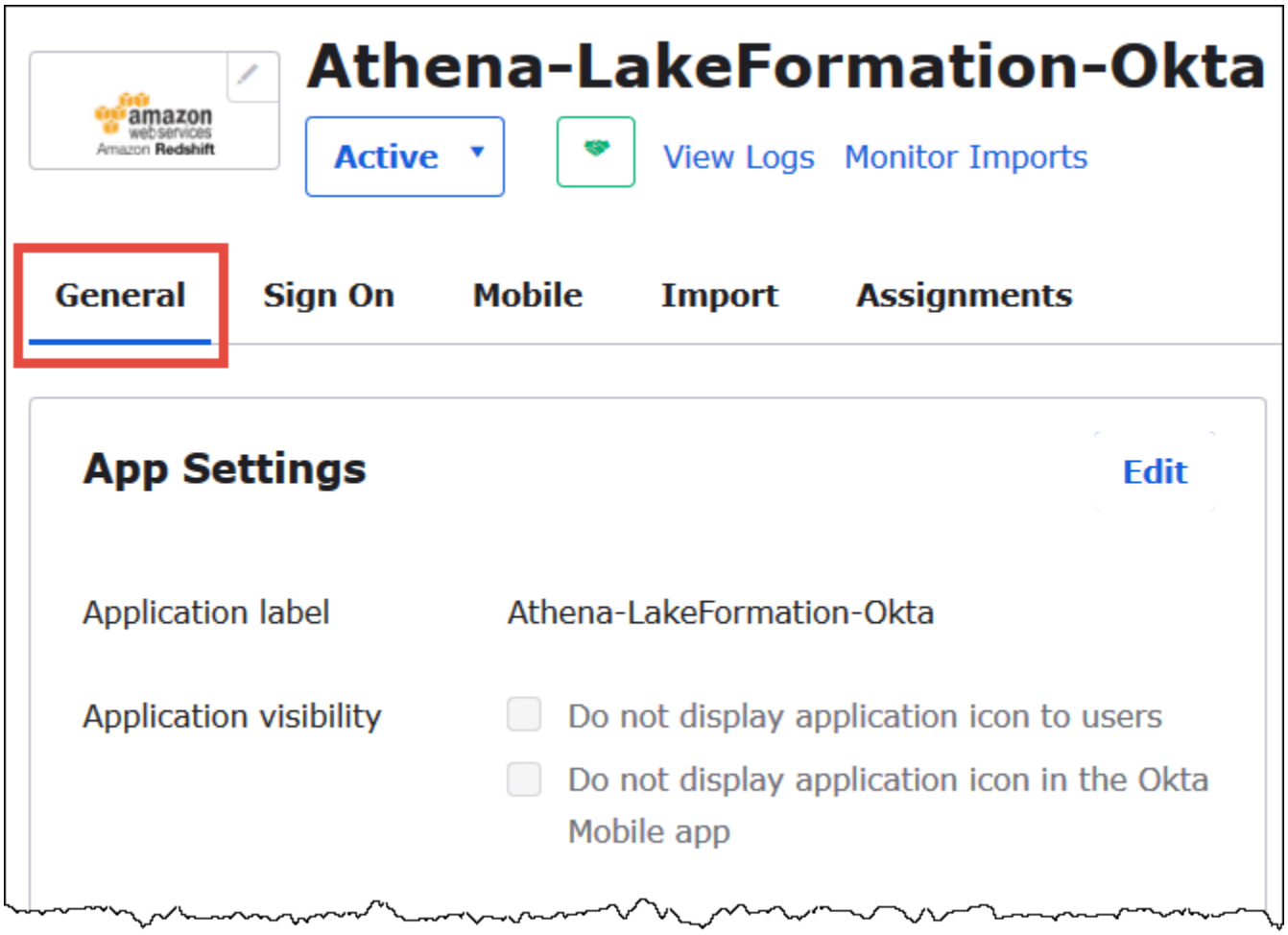


2. [Save] (保存) を選択します。

次に、Okta アプリケーション ID をコピーします。これは、後ほど Athena に接続する JDBC 文字列で必要になります。

Okta アプリケーション ID を見つけてコピーする

1. Okta アプリケーションの [General] (全般) タブを選択します。



2. [App Embed Link] (アプリ埋め込みリンク) までスクロールダウンします。
3. [Embed Link] (埋め込みリンク) から、URL の Okta アプリケーション ID の部分をコピーして、安全に保存しておきます。Okta アプリケーション ID は、URL の `amazon_aws_redshift/` と、次のスラッシュの間の部分です。例えば、URL に `amazon_aws_redshift/aaa/bbb` が含まれる場合、アプリケーション ID は `aaa` です。

> **Note**
>
> 埋め込みリンクを使用して Athena コンソールに直接ログインし、データベースを表示することはできません。SAML ユーザーおよびグループに対する Lake Formation のアクセス許可は、JDBC または ODBC ドライバーを使用して Athena にクエリを送信する場合のみ認識されます。データベースを表示するには、JDBC ドライバーを使用して Athena に接続する SQL Workbench/J ツールを使用できます。SQL Workbench/J ツールについては、手順 7: Athena JDBC クライアント経由でアクセスを検証する で説明します。

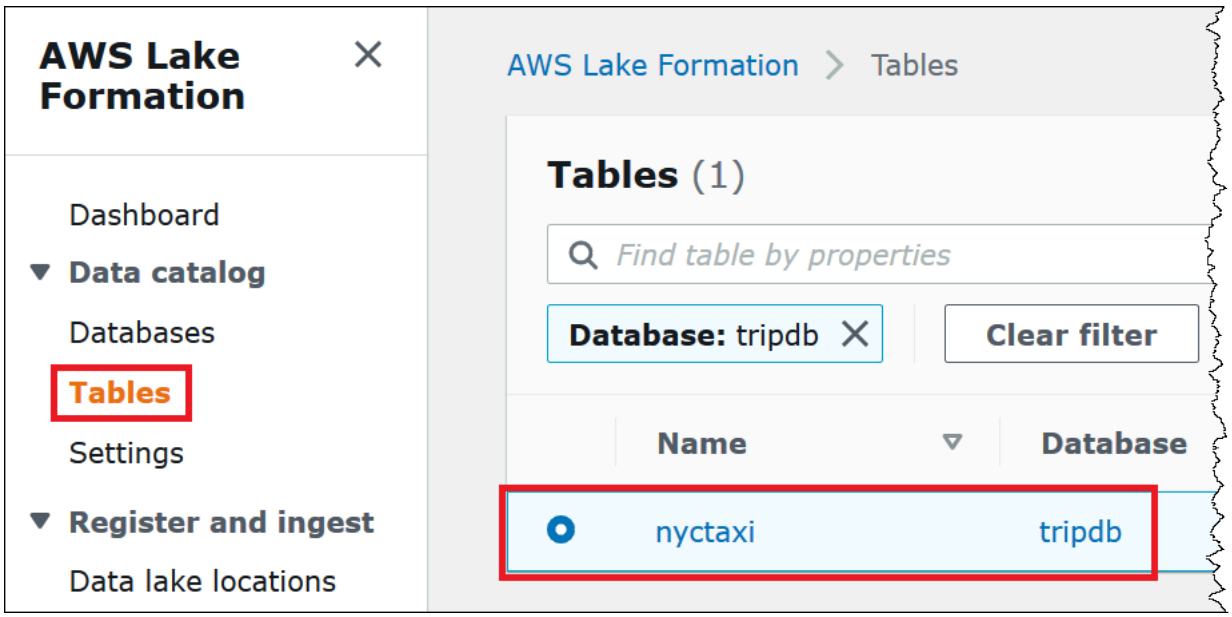
ステップ 6: AWS Lake Formation 経由でユーザーとグループの許可を付与する

このステップでは、Lake Formation コンソールを使用して、テーブルに対する許可を SAML ユーザーとグループに付与します。以下のタスクを実行します。

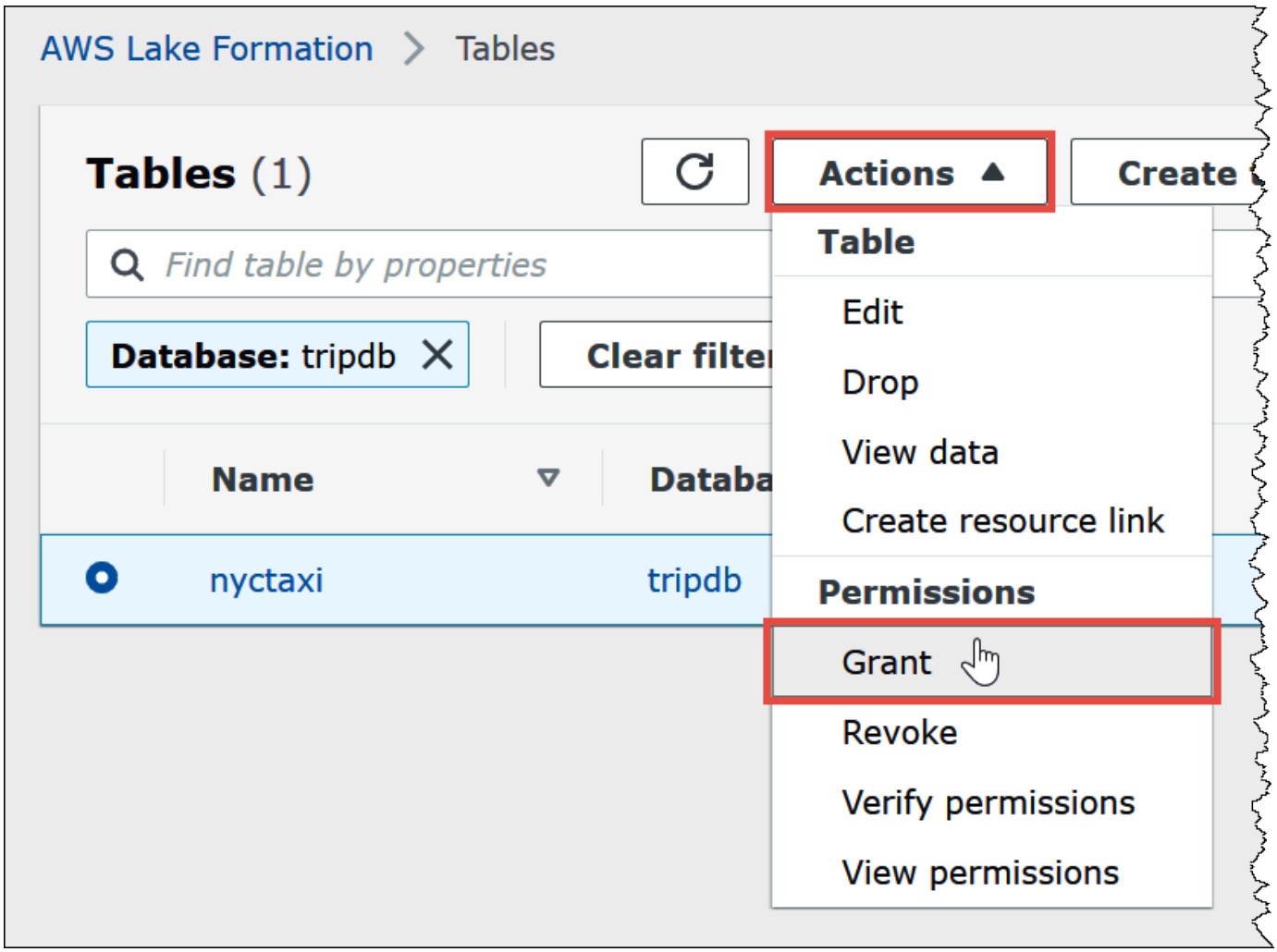
- Okta SAML ユーザーの ARN と、テーブルへの関連するユーザー許可を指定する。
- Okta SAML グループの ARN と、テーブルへの関連するユーザー許可を指定する。
- 付与した許可を検証する。

Lake Formation で Okta ユーザーに許可を付与する

1. データレイク管理者として AWS Management Console にログインします。
2. Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) を開きます。
3. ナビゲーションペインで [Tables] (テーブル) を選択してから、許可を付与するテーブルを選択します。このチュートリアルでは、`tripdb` データベースの `nyctaxi` テーブルを使用します。



4. [Actions] (アクション) から [Grant] (付与) を選択します。



5. [Grant Permissions] (許可の付与) ダイアログで、以下の情報を入力します。

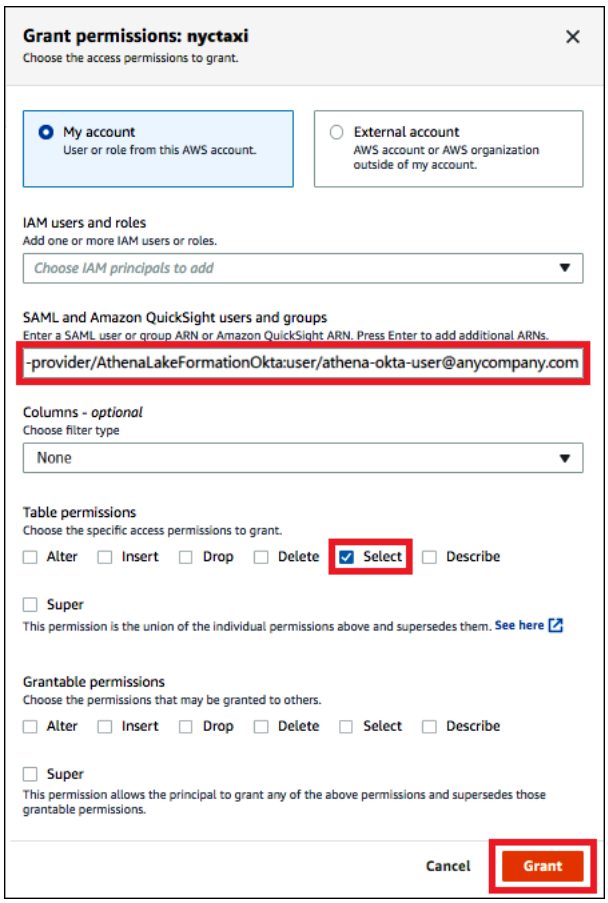
   a. [SAML and Amazon QuickSight users and groups] (SAML および Amazon QuickSight のユーザーとグループ) で、Okta SAML ユーザー ARN を以下の形式で入力します。

   ```
   arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta:user/<athena-okta-user>@<anycompany.com>
   ```

   b. [Columns] (列) の [Choose filter type] (フィルタータイプの選択) では、オプションで [Include columns] (列を含める) または [Exclude columns] (列を除外する) を選択します。

   c. フィルターの下にある [Choose one or more columns] (1 つ、または複数の列を選択する) ドロップダウンを使用して、ユーザーに含める列またはユーザーから除外する列を指定します。

   d. [Table permissions] (テーブル許可) には [SELECT] を選択します。このチュートリアルでは SELECT 許可のみを付与しますが、ユーザーによっては要件が異なる場合があります。



6. [Grant] (付与) を選択します。

ここで、Okta グループにも同様のステップを実行します。

Lake Formation で Okta グループに許可を付与する

1. Lake Formation コンソールの [Tables] (テーブル) ページで、[nyctaxi] テーブルが選択されていることを確認します。
2. [Actions] (アクション) から [Grant] (付与) を選択します。
3. [Grant Permissions] (許可の付与) ダイアログで、以下の情報を入力します。

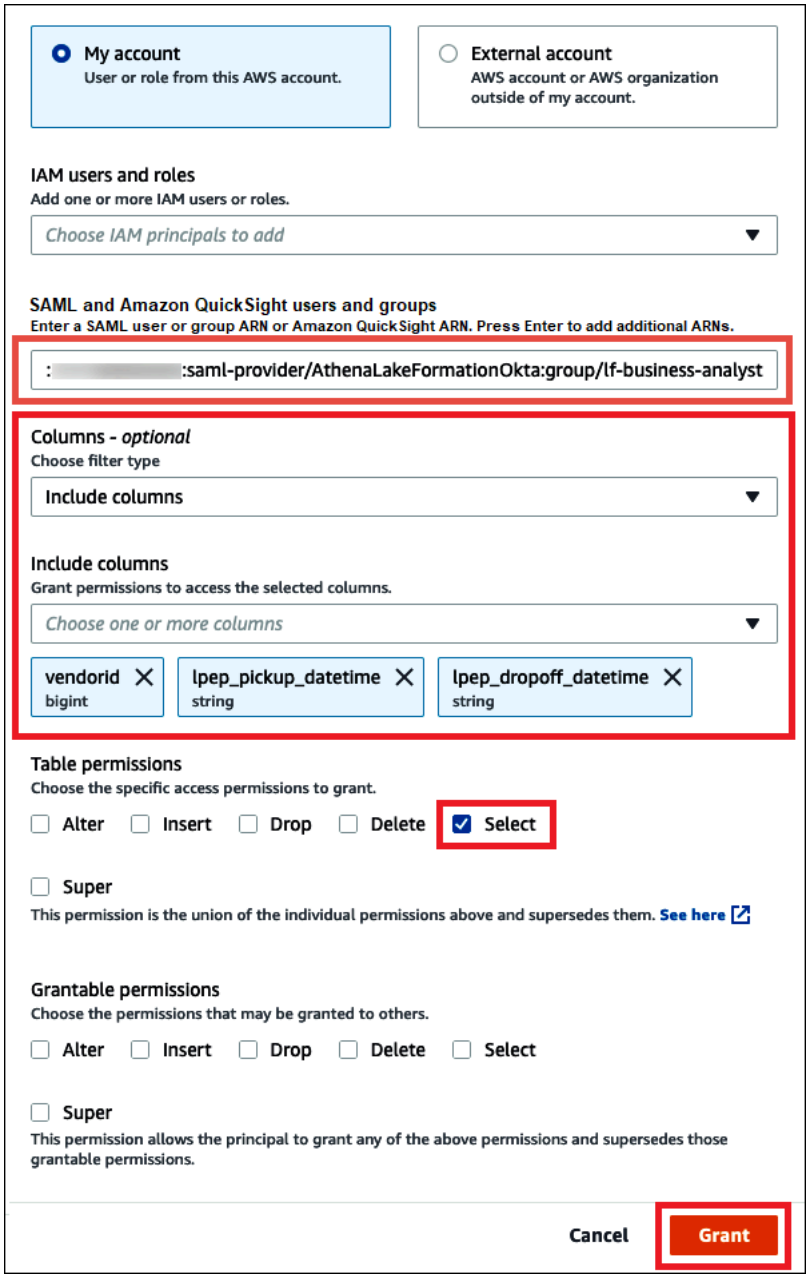
    a. [SAML and Amazon QuickSight users and groups] (SAML および Amazon QuickSight のユーザーとグループ) で、Okta SAML グループ ARN を以下の形式で入力します。

```
arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta:group/lf-business-analyst
```

b. [Columns] (列) の [Choose filter type] (フィルタータイプの選択) には、[Include columns] (フィルタータイプを含める) を選択します。

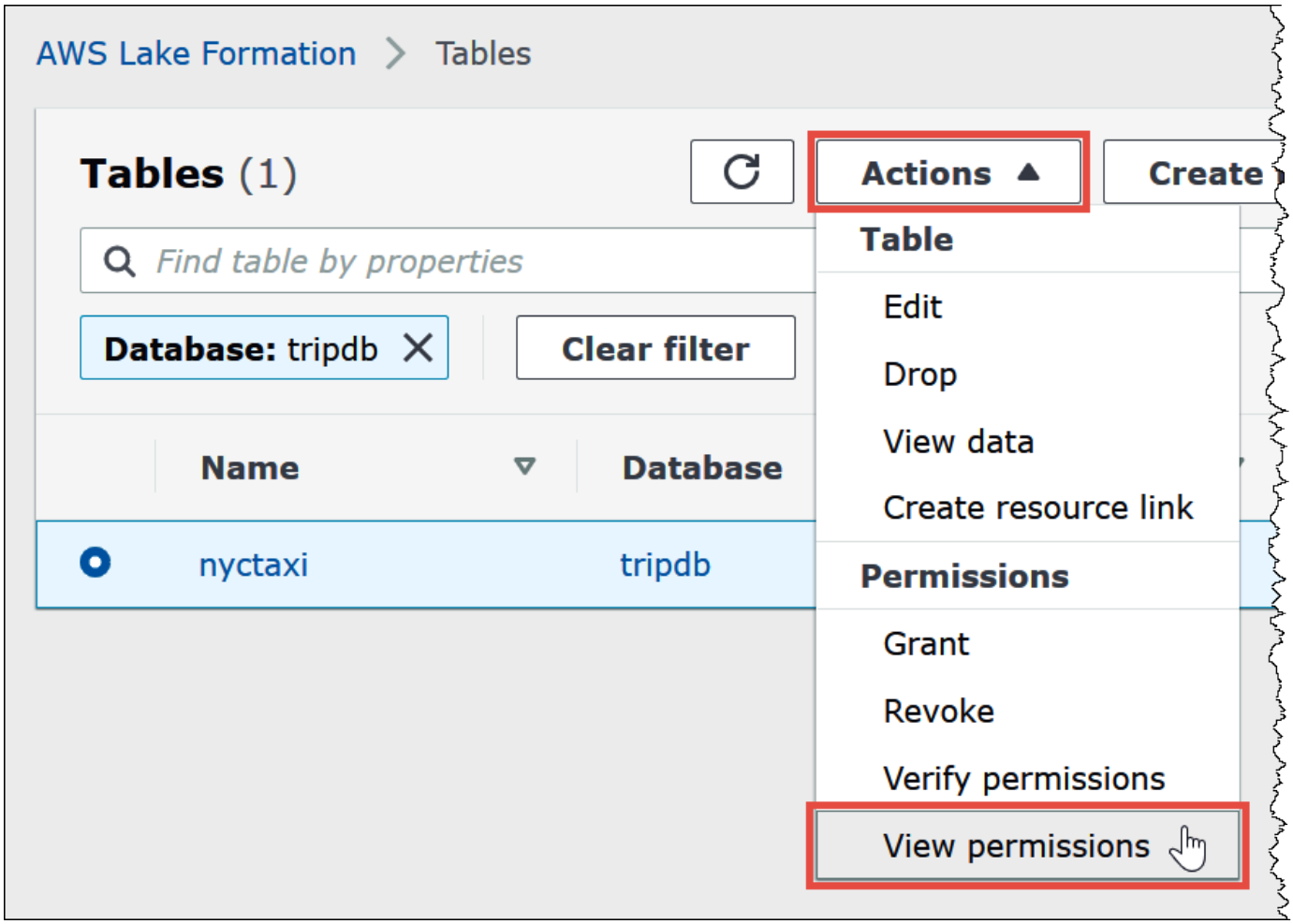
c. [Choose one or more columns] (1 つ、または複数の列を選択する) には、テーブルの最初の 3 列を選択します。

d. [Table permissions] (テーブル許可) には、付与する特定のアクセス許可を選択します。このチュートリアルでは SELECT 許可のみを付与しますが、ユーザーによっては要件が異なる場合があります。

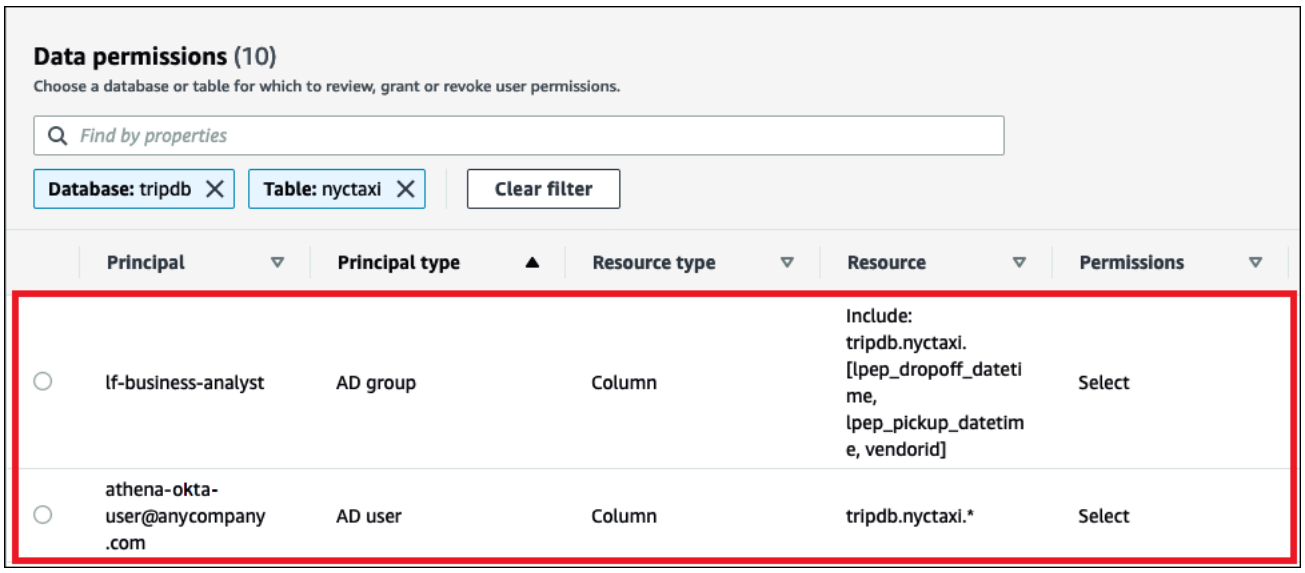


4. [Grant] (付与) を選択します。

5. 付与した許可を検証するには、[Actions] (アクション)、[View permissions] (許可の表示) の順に選択します。



`nyctaxi` テーブルの [Data permissions] (データのアクセス許可) ページに、athena-okta-user と lf-business-analyst グループのアクセス許可が表示されます。



手順 7: Athena JDBC クライアント経由でアクセスを検証する

これで、JDBC クライアントを使用して、Okta SAML ユーザーとして Athena へのテスト接続を実行する準備が整いました。

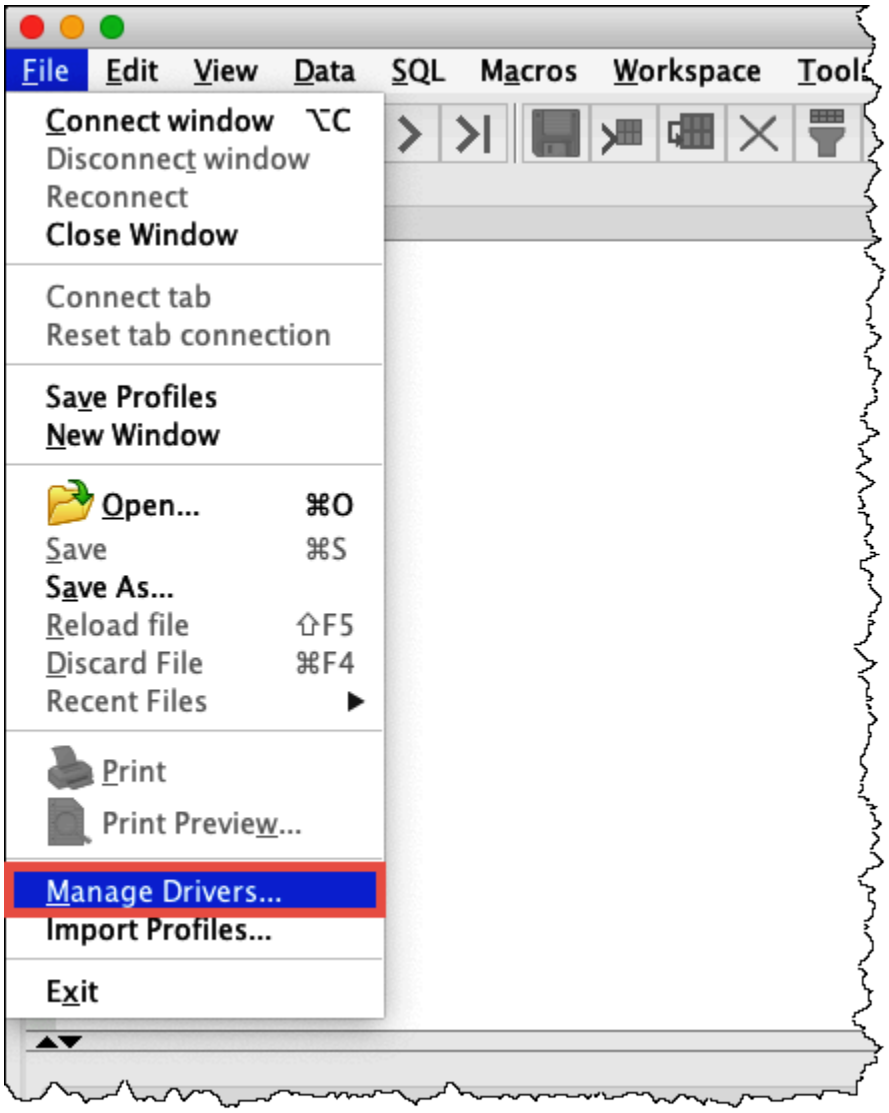
このセクションでは、以下のタスクを実行します。

- テストクライアントの準備 – Athena JDBC ドライバーをダウンロードし、SQL Workbench をインストールして、ドライバーを Workbench に追加する。このチュートリアルでは、Okta 認証経由で Athena にアクセスし、Lake Formation 許可を検証するために SQL Workbench を使用します。
- MySQL Workbench で以下を実行する。
  - Athena Okta ユーザーの接続の作成。
  - Athena Okta ユーザーとしてのテストクエリの実行。
  - ビジネスアナリストユーザーの接続の作成とテスト。
- Okta コンソールで、ビジネスアナリストユーザーをデベロッパーグループに追加する。
- Lake Formation コンソールで、デベロッパーグループのテーブル許可を設定する。
- SQL Workbench で、ビジネスアナリストユーザーとしてテストクエリを実行し、許可の変更が結果にどのように影響するかを検証する。

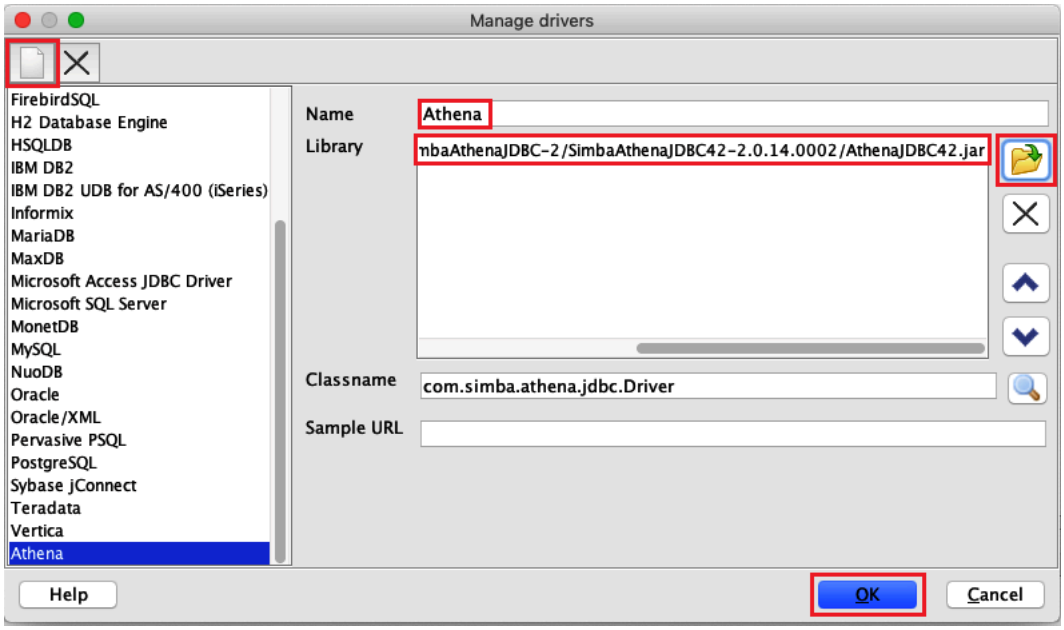
テストクライアントを準備する

1. JDBC を使用した Amazon Athena への接続 から、Lake Formation 対応の Athena JDBC ドライバー (2.0.14 以降のバージョン) をダウンロードして抽出します。

2. Modified Apache 2.0 License に基づいて利用できる無料の SQL Workbench/J SQL クエリツールをダウンロードしてインストールします。
3. SQL Workbench/J で [File] (ファイル)、[Manage Drivers] (ドライバーの管理) と選択します。



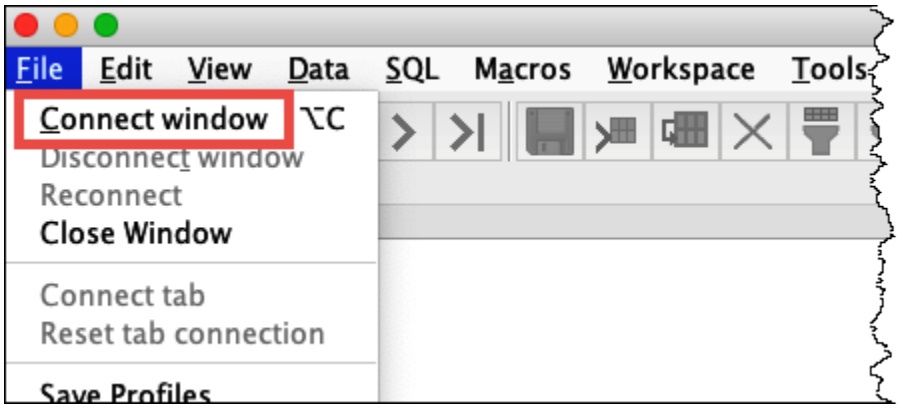
4. [Manage Drivers] (ドライバーの管理) ダイアログボックスで、以下のステップを実行します。
   a. 新しいドライバーのアイコンを選択します。
   b. [名前] に「**Athena**」と入力します。
   c. [Library] (ライブラリ) には、先ほどダウンロードした Simba Athena JDBC `.jar` ファイルを参照して選択します。
   d. [OK] をクリックします。



これで、Athena Okta ユーザーの接続を作成し、テストする準備ができました。

Okta ユーザーの接続を作成する

1. [File] (ファイル)、[Connect window] (ウィンドウの接続) と選択します。



2. [Connection profile] (接続プロファイル) ダイアログボックスで、以下の情報を入力して接続を作成します。

   - 名前のボックスに **`Athena_Okta_User_Connection`** と入力します。
   - [Driver] (ドライバー) には Simba Athena JDBC ドライバーを選択します。
   - [URL] で、以下のいずれかを実行します。
     - 接続 URL を使用するには、単一行の接続文字列を入力します。以下の例には、読みやすいように改行が追加されています。

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
AwsCredentialsProviderClass=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider;
user=athena-okta-user@anycompany.com;
password=password;
idp_host=okta-idp-domain;
App_ID=okta-app-id;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

- AWS プロファイルベースの URL を使用するには、以下のステップを実行します。

  1. 以下の例のような AWS 認証情報ファイルがある AWS プロファイルを設定します。

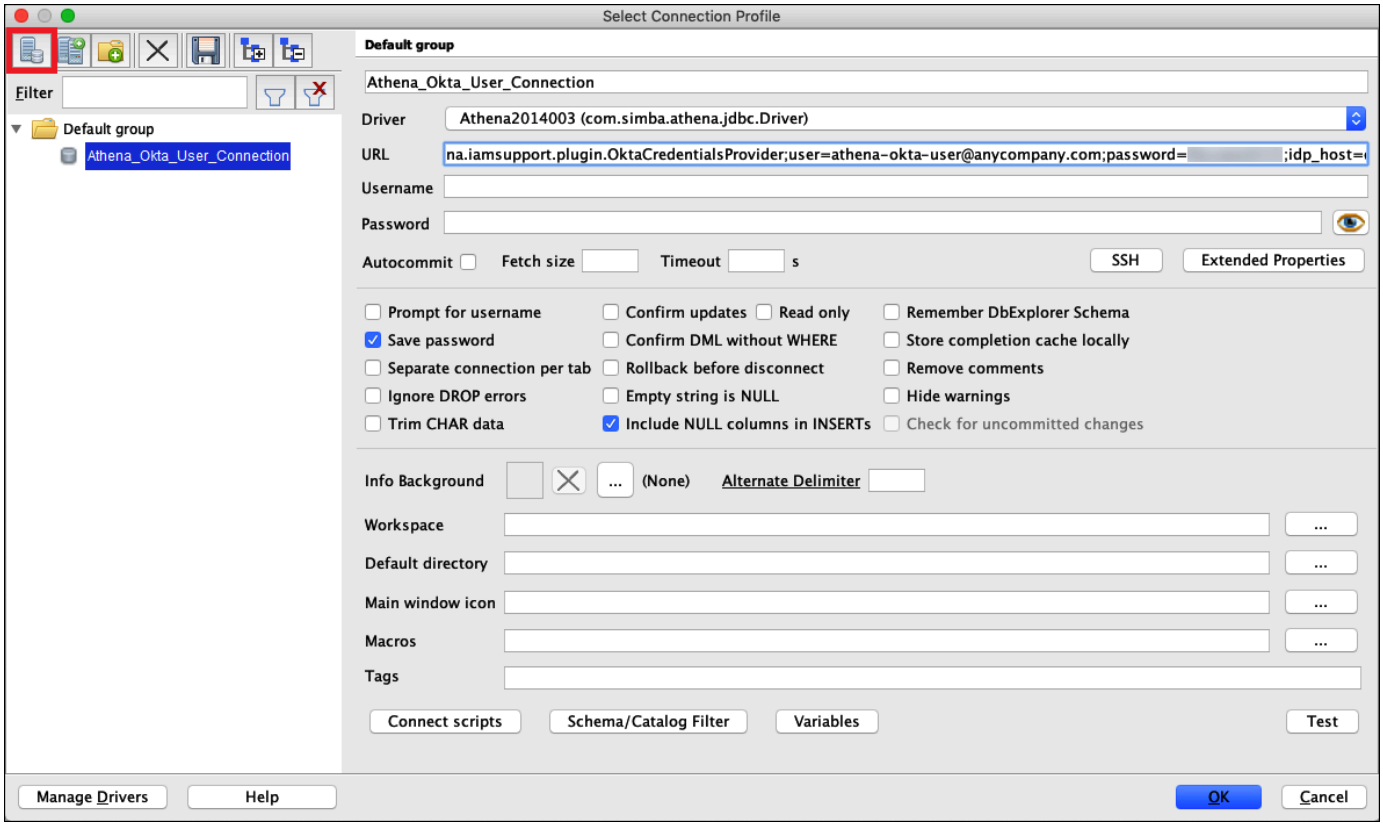
     ```
     [athena_lf_dev]
     plugin_name=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider
     idp_host=okta-idp-domain
     app_id=okta-app-id
     uid=athena-okta-user@anycompany.com
     pwd=password
     ```

  2. URL には、以下の例のような単一行の接続文字列を入力します。この例には、読みやすいように改行が追加されています。

     ```
     jdbc:awsathena://AwsRegion=region-id;
     S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
     profile=athena_lf_dev;
     SSL_Insecure=true;
     LakeFormationEnabled=true;
     ```

  これらの例は、Athena への接続に必要な URL の基本的な表現であることに注意してください。URL でサポートされているパラメータの完全なリストについては、「JDBC のドキュメント」を参照してください。

以下の画像は、接続 URL を使用する SQL Workbench 接続プロファイルを示しています。

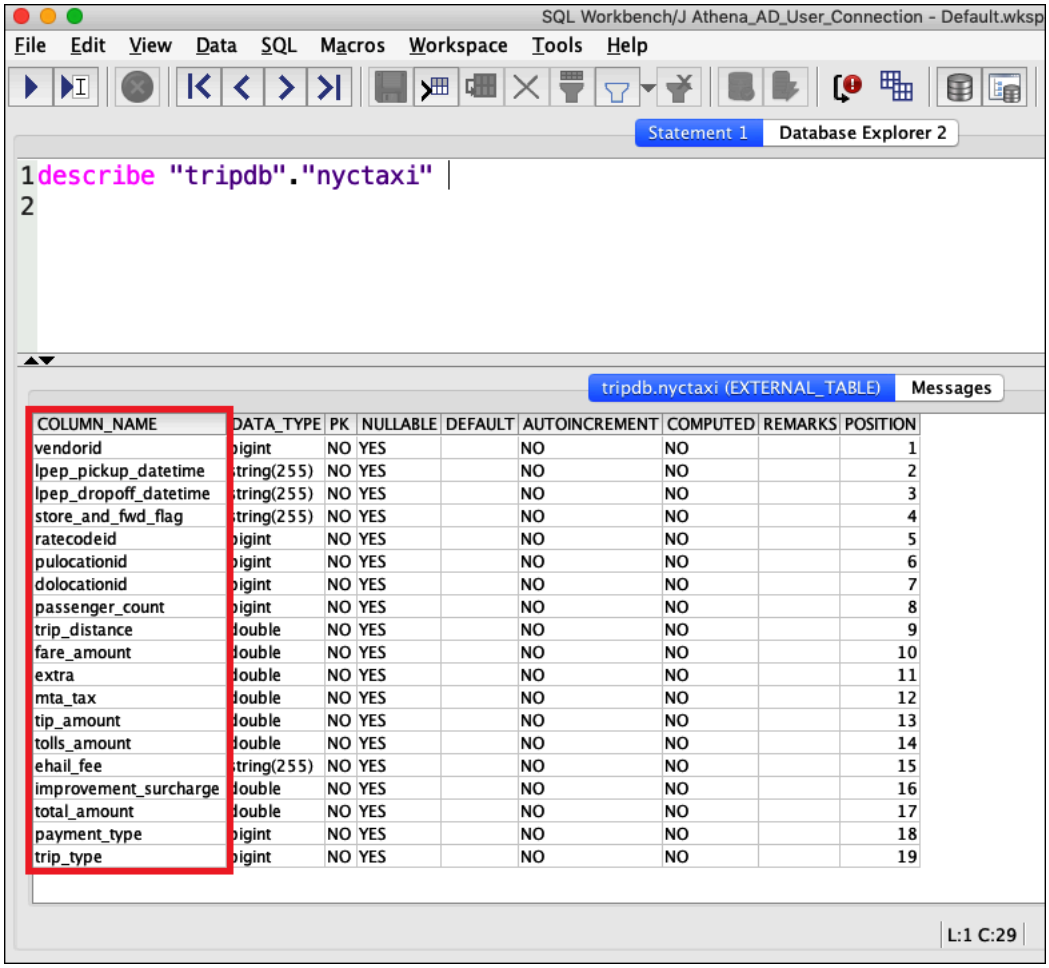


Okta ユーザーの接続を確立したところで、データを取得することによって接続をテストすることができます。

Okta ユーザーの接続をテストする

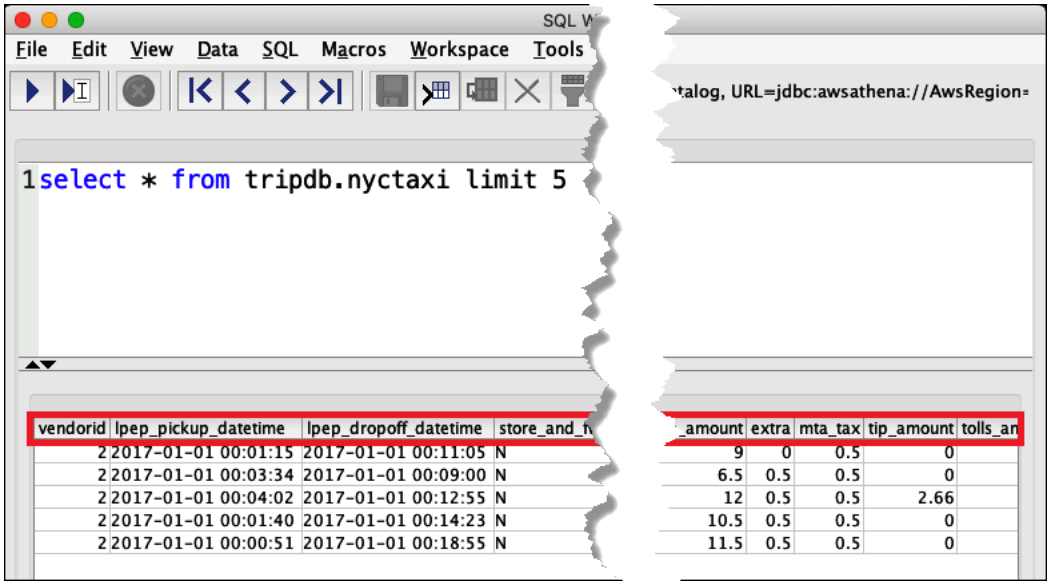
1. [Test] (テスト) をクリックして、接続が成功することを検証します。
2. SQL Workbench の [Statement] (ステートメント) ウィンドウから、以下の SQL DESCRIBE コマンドを実行します。すべての列が表示されていることを確認します。

```
DESCRIBE "tripdb"."nyctaxi"
```



3. SQL Workbench の [Statement] (ステートメント) ウィンドウから、以下の SQL SELECT コマンドを実行します。すべての列が表示されていることを確認します。

```
SELECT * FROM tripdb.nyctaxi LIMIT 5
```



次に、athena-ba-user が lf-business-analyst グループのメンバーとして、先ほど Lake Formation で指定したテーブル内の最初の 3 列にしかアクセスできないことを検証します。

athena-ba-user のアクセス権を検証する

1. SQL Workbench の [Connection profile] (接続プロファイル) ダイアログボックスで、別の接続プロファイルを作成します。

   - 接続プロファイル名には **`Athena_Okta_Group_Connection`** を入力します。
   - [Driver] (ドライバー) には Simba Athena JDBC ドライバーを選択します。
   - [URL] で、以下のいずれかを実行します。
     - 接続 URL を使用するには、単一行の接続文字列を入力します。以下の例には、読みやすいように改行が追加されています。

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
AwsCredentialsProviderClass=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider;
user=athena-ba-user@anycompany.com;
password=password;
idp_host=okta-idp-domain;
App_ID=okta-application-id;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

- AWS プロファイルベースの URL を使用するには、以下のステップを実行します。

  1. 以下の例のような認証情報ファイルがある AWS プロファイルを設定します。

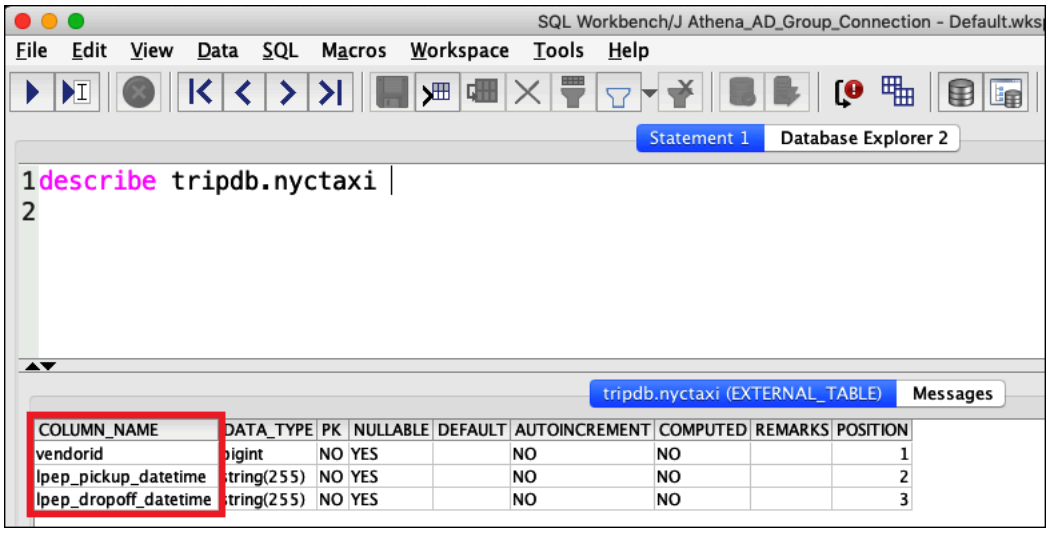
```
[athena_lf_ba]
plugin_name=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider
idp_host=okta-idp-domain
app_id=okta-application-id
uid=athena-ba-user@anycompany.com
pwd=password
```

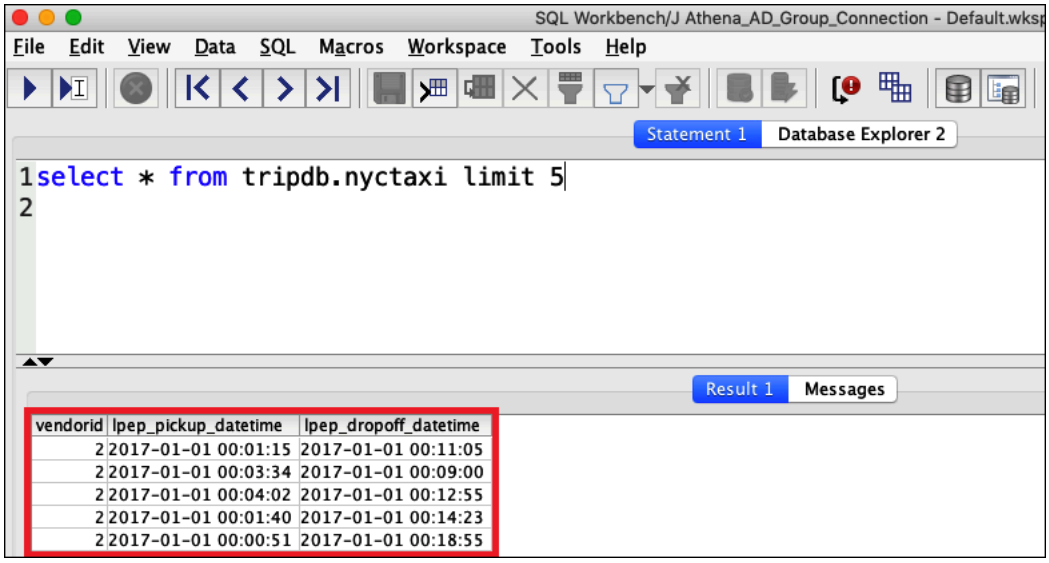
  2. URL には、以下のような単一行の接続文字列を入力します。この例には、読みやすいように改行が追加されています。

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
profile=athena_lf_ba;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

2. [Test] (テスト) をクリックして接続が成功することを確かめます。
3. [SQL Statement] (SQL ステートメント) ウィンドウから、以前に実行したものと同じ DESCRIBE と SELECT SQL コマンドを実行して、結果を調べます。

   athena-ba-user は lf-business-analyst グループのメンバーであることから、Lake Formation コンソールで指定した最初の 3 列のみが返されます。

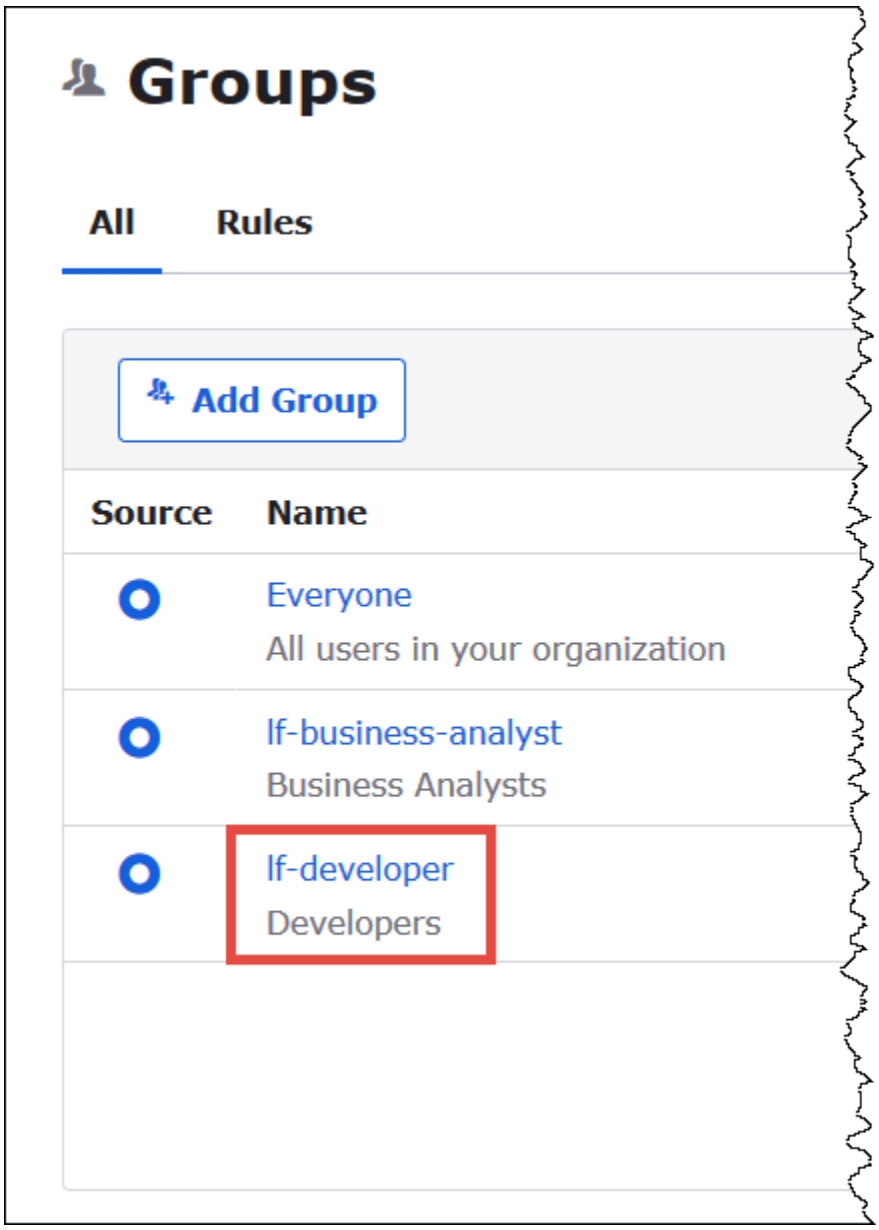




次に、Okta コンソールに戻って、`athena-ba-user` を `lf-developer` Okta グループに追加します。

athena-ba-user を lf-developer グループに追加する

1. 割り当てられた Okta ドメインの管理ユーザーとして Okta コンソールにサインインします。
2. [Directory] (ディレクトリ)、[Groups] (グループ) の順にクリックします。
3. グループページで lf-developer グループを選択します。



4. [Manage People] (ピープルの管理) をクリックします。
5. [Not Members] (非メンバー) リストから athena-ba-user を選択して lf-developer グループに追加します。
6. [Save] (保存) を選択します。

ここで Lake Formation コンソールに戻り、lf-developer グループのテーブル許可を設定します。

lf-developer グループのテーブル許可を設定する

1. Lake Formation コンソールにデータレイク管理者としてログインします。

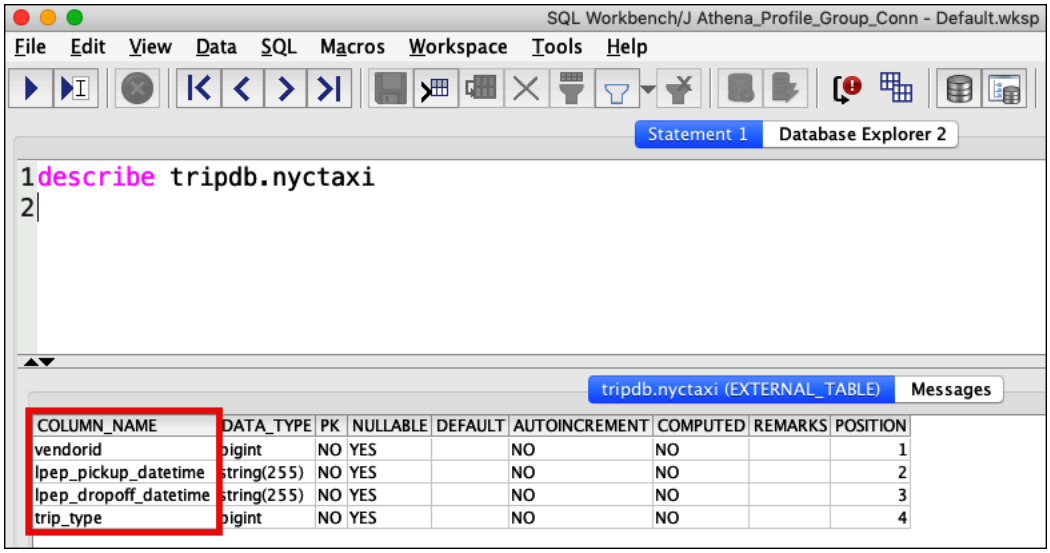
2. ナビゲーションペインで、[Table] (テーブル) を選択します。
3. nyctaxi テーブルを選択します。
4. [Actions] (アクション)、[Grant] (付与) と選択します。
5. [Grant Permissions] (許可の付与) ダイアログで、以下の情報を入力します。
   - [SAML and Amazon QuickSight users and groups] (SAML および Amazon QuickSight のユーザーとグループ) に、以下の形式で Okta SAML lf-developer グループ ARN を入力します。
   - [Columns] (列) の [Choose filter type] (フィルタータイプの選択) には、[Include columns] (フィルタータイプを含める) を選択します。
   - trip_type 列を選択します。
   - [Table permissions] (テーブル許可) には [SELECT] を選択します。
6. [Grant] (付与) を選択します。

これで、SQL Workbench を使用して lf-developer グループの許可への変更を検証できます。この変更は、lf-developer グループのメンバーになった athena-ba-user が利用できるデータに反映されています。

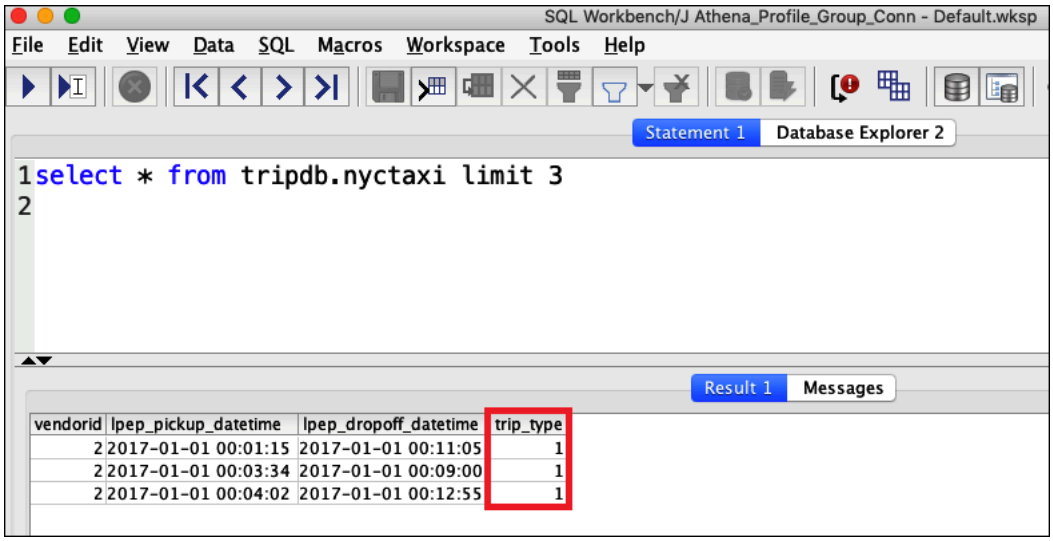
athena-ba-user の許可への変更を検証する

1. SQL Workbench プログラムを閉じてから、もう一度開きます。
2. athena-ba-user のプロファイルに接続します。
3. [Statement] (ステートメント) ウィンドウから、以前実行したものと同じ SQL ステートメントを発行します。

   今回は、trip_type 列が表示されています。



athena-ba-user は lf-developer グループと lf-business-analyst グループ両方のメンバーになったため、これらのグループの Lake Formation 許可の組み合わせによって、返される列が決定されます。



結論

このチュートリアルでは、Okta を SAML プロバイダーとして使用して、Athena の AWS Lake Formation との統合を設定しました。Lake Formation と IAM を使用して、SAML ユーザーが利用できるデータレイク AWS Glue データカタログ内のリソースを制御しました。

関連リソース

関連情報については、以下のリソースを参照してください。

- JDBC を使用した Amazon Athena への接続
- Athena API へのフェデレーションアクセスの有効化
- AWS Lake Formation デベロッパーガイド
- 「AWS Lake Formation デベロッパーガイド」の「データカタログへのアクセス許可の付与と取り消し」。
- 「IAM ユーザーガイド」の「ID プロバイダーとフェデレーション」。
- 「IAM ユーザーガイド」の「Creating IAM SAML identity providers」(IAM SAML ID プロバイダーの作成)。
- AWS セキュリティブログの「Windows アクティブディレクトリ、ADFS、および SAML 2.0 を使用した AWS へのフェデレーションの有効化」。

ワークロード管理

Athena のワークグループ、キャパシティ管理、パフォーマンスチューニング、圧縮サポート、タグ、サービスクォータ機能を使用してワークロードを管理できます。

トピック

- ワークグループを使用してクエリのアクセスとコストを制御する
- クエリ処理キャパシティの管理
- Athena でのパフォーマンスのチューニング
- Athena での圧縮のサポート
- Athena リソースへのタグ付け
- Service Quotas

ワークグループを使用してクエリのアクセスとコストを制御する

ワークグループを使用して、ユーザー、チーム、アプリケーション、またはワークロードを分離し、各クエリまたはワークグループ全体で処理できるデータ量に制限を設定して、コストを追跡できます。ワークグループは リソースとして機能するため、リソースレベルのアイデンティティベースのポリシーを使用して特定のワークグループへのアクセスを制御できます。また、Amazon CloudWatch でクエリ関連のメトリクスを表示する、スキャンされるデータの量に対する制限を設定してコストを制御する、しきい値を作成する、およびこれらのしきい値を超過したときに Amazon SNS などのアクションをトリガーすることもできます。

コストをさらに抑えるには、指定したデータ処理ユニットの数でキャパシティ予約を作成し、予約に 1 つ以上のワークグループを追加できます。詳細については、「クエリ処理キャパシティの管理」を参照してください。

ワークグループは、次のように IAM、CloudWatch、Amazon Simple Notification Service、および AWS コストと使用状況レポートと統合します。

- ワークグループでクエリを実行できるユーザーを管理するリソースレベルの許可を使用したアイデンティティベースの IAM ポリシー。
- クエリメトリクスを有効にすると、Athena がワークグループのクエリメトリクスを CloudWatch にパブリッシュします。
- Amazon SNS では、ワークグループでのクエリのデータ使用量が設定されたしきい値を超えたときに、指定されたワークグループユーザーにアラームを発行する Amazon SNS トピックを作成できます。
- Billing and Cost Management コンソールで、コスト配分タグとして設定されたタグとワークグループをタグ付けすると、そのワークグループのクエリの実行に関連するコストがそのコスト配分タグとともにコストと使用状況レポートに表示されます。

トピック

- クエリを実行するためのワークグループの使用
- CloudWatch のメトリクスとイベントを使用したコストの管理とクエリのモニタリング

AWS ビッグデータブログ記事「Amazon Athena ワークグループを使用したクエリの分離とコストの管理」も参照してください。この記事では、ワークグループを使用してワークロードを分離し、ユーザーアクセスを制御して、クエリの使用状況とコストを管理する方法が紹介されています。

クエリを実行するためのワークグループの使用

チーム、アプリケーション、またはさまざまなワークロードのクエリを分離するためにワークグループを使用することをお勧めします。たとえば、組織で 2 つの異なるチームに対して、別のワークグループを作成することができます。ワークロードを分割することもできます。たとえば、2 つの独立したワークグループを、1 つはレポート生成などの自動スケジュールされたアプリケーションのため、もう 1 つはアナリストによるアドホック使用のために作成できます。ワークグループを切り替えることができます。

トピック

- ワークグループを使用するメリット

- ワークグループの仕組み
- ワークグループのセットアップ
- ワークグループにアクセスするための IAM ポリシー
- ワークグループ設定
- ワークグループの管理
- IAM アイデンティティセンターが有効な Athena ワークグループの使用
- Athena ワークグループ API
- ワークグループのトラブルシューティング

ワークグループを使用するメリット

ワークグループにより、次のことが可能になります。

ユーザー、チーム、アプリケーション、またはワークロードをグループに分ける。	各ワークグループには、独自の固有のクエリ履歴と保存したクエリのリストがあります。詳細については、「ワークグループの仕組み」を参照してください。 ワークグループのすべてのクエリに対して、ワークグループ設定を設定することを選択できます。これには、クエリ結果を保存する Simple Storage Service (Amazon S3) の場所、予測されるバケット所有者、暗号化、クエリ結果バケットに書き込まれたオブジェクトのコントロールが含まれます。ワークグループ設定を強制することもできます。詳細については、「ワークグループ設定」を参照してください。
コストの制約を強制する。	ワークグループのクエリに 2 つのタイプのコスト制約を設定できます。 • [Per-query limit] (クエリごとの制限) は、クエリごとにスキャンされたデータ量のしきい値です。Athena は、指定されたしきい値を超えるとクエリをキャンセルします。この制限は、ワークグループ内で実行中の各クエリに適用されます。クエリごとの制限を 1 つだけ設定し、必要に応じてそれを更新できます。 • ワークグループごとの制限は、ワークグループでクエリによってスキャンされたデータ量に対して各ワークグループに設定できるしきい値です。しきい値を超過すると、特定のユーザーに E メー

	ルを送信するなど、選択したアクションをトリガーする Amazon SNS アラームがアクティブ化されます。各ワークグループに複数のワークグループごとの制限を設定できます。 詳細なステップについては、「データ使用量の制御制限の設定」を参照してください。
CloudWatch のすべてのワークグループクエリについてクエリ関連のメトリクスを追跡する。	メトリクスを発行するようにワークグループを設定した場合、ワークグループで実行されるクエリごとに、Athena が CloudWatch にメトリクスを発行します。Athena コンソール内の各ワークグループについて、クエリメトリクスを表示することができます。CloudWatch では、カスタムダッシュボードを作成して、これらのメトリクスにしきい値およびアラームを設定できます。

ワークグループの仕組み

Athena のワークグループには、以下の特徴があります。

- デフォルトでは、各アカウントにはプライマリワークグループがあり、デフォルトのアクセス許可により認証されたすべてのユーザーがこのワークグループにアクセスできます。プライマリワークグループは削除できません。
- 作成した各ワークグループには、そこで実行されたクエリについてのみ保存されたクエリおよびクエリ履歴が表示され、アカウントのすべてのクエリは表示されません。これにより、アカウント内の他のクエリからクエリが分離され、自分が保存したクエリおよび履歴内のクエリを見つけることがより効率的になります。
- ワークグループを無効にすると、有効にするまでクエリは実行されなくなります。無効なワークグループに送信されたクエリは、再び有効にするまで失敗します。
- アクセス権限がある場合は、空のワークグループと、保存したクエリを含むワークグループを削除できます。この場合、ワークグループを削除する前に、保存されたクエリが削除されることを Athena が警告します。他のユーザーがアクセスできるワークグループを削除する前に、そのユーザーが引き続きクエリを実行できる他のワークグループにアクセスできることを確認してください。
- ワークグループ全体の設定をセットアップし、ワークグループで実行されるすべてのクエリでその使用を強制することができます。設定には、Simple Storage Service (Amazon S3) にあるクエリ結果、予測されるバケット所有者、暗号化、クエリ結果バケットに書き込まれたオブジェクトのコントロールが含まれます。

> **Important**
>
> ワークグループ全体の設定を強制する場合、このワークグループで実行されるすべてのクエリはワークグループ設定を使用します。これは、クライアント側の設定がワークグループの設定と異なる場合でも発生します。詳細については、ワークグループ設定がクライアント側の設定を上書きする を参照してください。

ワークグループの制限事項

- アカウントのリージョンあたり最大 1000 個のワークグループを作成できます。
- プライマリワークグループは削除できません。
- 各ワークグループで最大 10 個のクエリタブを開くことができます。ワークグループを切り替えると、クエリタブは最大 3 つのワークグループに対して開いたままになります。

ワークグループのセットアップ

ワークグループをセットアップするには、ワークグループを作成し、それらの使用に対するアクセス権限を確立する必要があります。まず、組織が必要とするワークグループを決定し、作成します。次に、`workgroup` リソースでのユーザーアクセスおよびアクションを制御する IAM ワークグループポリシーをセットアップします。これで、これらのワークグループにアクセスできるユーザーは、その中でクエリを実行できるようになります。

> **Note**
>
> 初めて使用するときには、これらのタスクを使用してワークグループをセットアップします。Athena アカウントがすでにワークグループを使用している場合、各アカウントのユーザーには、そのアカウントの 1 つ、または複数のワークグループでクエリを実行するための許可が必要です。クエリを実行する前に、IAM ポリシーを確認してアクセスできるワークグループを確認し、必要に応じてポリシーを調整して、使用する予定のワークグループに切り替えます。

デフォルトでは、ワークグループを作成していない場合、アカウントのすべてのクエリはプライマリワークグループで実行されます。

Athena では、コンソールの右上にある [Workgroup] (ワークグループ) オプションに、現在のワークグループが表示されます。このオプションを使用して、ワークグループを切り替えることができます。クエリを実行すると、現在のワークグループでクエリが実行されます。コンソールのワークグループでクエリを実行するか、JDBC または ODBC ドライバーを介して API オペレーション、コマンドラインインターフェイス、またはクライアントアプリケーションを使用してクエリを実行できます。ワークグループにアクセスすると、そのワークグループの設定、メトリクス、およびデータ使用量の制御制限を表示できます。設定およびデータ使用量の制御制限を編集できる追加のアクセス許可を持つことができます。

ワークグループをセットアップするには

1. 作成するワークグループを決定します。たとえば、次のことを決定できます。

   - 各ワークグループでクエリを実行できるユーザー、およびワークグループ設定を所有しているユーザー。これにより、作成する IAM ポリシーが決まります。詳細については、「ワークグループにアクセスするための IAM ポリシー」を参照してください。
   - 各ワークグループで実行されるクエリのクエリ結果に使用する Amazon S3 内の場所。この場所は、ワークグループのクエリ結果にこれを指定する前に、Amazon S3 内に存在している必要があります。ワークグループを使用するすべてのユーザーは、この場所にアクセスできる必要があります。詳細については、「ワークグループ設定」を参照してください。
   - Simple Storage Service (Amazon S3) クエリ結果バケットの所有者が、バケットに書き込まれる新しいオブジェクトに対するフルコントロールを所有するかどうか。たとえば、クエリ結果の場所を別のアカウントが所有している場合に、クエリ結果の所有権とフルコントロールを他のアカウントに付与することができます。詳細については、「AclConfiguration」を参照してください。
   - 出力先のバケットの所有者になることが予想される AWS アカウント の ID を指定します。これは、オプションで追加できるセキュリティ対策です。バケット所有者のアカウント ID がここで指定した ID と一致しない場合、バケットに出力できません。詳細については、「Amazon S3 ユーザーガイド」の「バケット所有者条件によるバケット所有者の確認」を参照してください。この設定は、CTAS、INSERT INTO、UNLOAD ステートメントには適用されません。
   - どの暗号化設定が必要か、およびどのワークグループに暗号化する必要があるクエリがあるか。暗号化されたクエリと暗号化されていないクエリには別々のワークグループを作成することをお勧めします。そうすることで、その中で実行されるすべてのクエリに適用されるワークグループに対して暗号化を強制できます。詳細については、「Amazon S3 に保存された Athena のクエリ結果の暗号化」を参照してください。

2. 必要に応じてワークグループを作成し、それらにタグを追加します。この手順については、「ワークグループの作成」を参照してください。
3. ユーザー、グループ、またはロール用の IAM ポリシーを作成して、それらがワークグループにアクセスできるようにします。ポリシーは、ワークグループメンバーシップおよび `workgroup` リソースのアクションへのアクセスを確立します。詳細なステップについては、「ワークグループにアクセスするための IAM ポリシー」を参照してください。JSON ポリシーの例については、「ワークグループとタグへのアクセス」を参照してください。
4. ワークグループ設定を指定します。クエリ結果用に Simple Storage Service (Amazon S3) の場所を指定し、オプションで予想されるバケット所有者と暗号化設定、クエリ結果バケットに書き込まれるオブジェクトのコントロールを指定します。ワークグループ設定を強制できます。詳細については、「ワークグループ設定」を参照してください。

   > **Important**
   >
   > [override client-side settings] (クライアント側の設定を上書きする) 場合は、Athena がワークグループの設定を使用します。これは、ドライバー、コマンドラインインターフェイス、または API オペレーションを使用してコンソールで実行するクエリに影響します。
   > クエリの実行が継続されている間に、特定の Amazon S3 バケットでの結果の可用性に基づいて構築されたオートメーションが破損する可能性があります。上書きする前に、ユーザーに通知することをお勧めします。ワークグループ設定を上書きするように設定した後は、ドライバーまたは API でクライアント側の設定の指定を省略できます。

5. クエリの実行に使用するワークグループをユーザーに通知します。使用できるワークグループ名、必要な IAM ポリシー、およびワークグループ設定についてアカウントのユーザーに通知する E メールを送信します。
6. クエリおよびワークグループについて、データ使用量の制御制限とも呼ばれるコスト管理制限を設定します。しきい値を超過したときに通知を送るには、Amazon SNS トピックを作成してサブスクリプションを設定します。詳しい手順については、「データ使用量の制御制限の設定」と、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS の開始方法」を参照してください。
7. ワークグループに切り替え、クエリを実行できるようにします。クエリを実行するには、適切なワークグループに切り替えます。詳細なステップについては、「the section called “クエリを実行するワークグループの指定”」を参照してください。

ワークグループにアクセスするための IAM ポリシー

ワークグループへのアクセスを制御するには、リソースレベルの IAM 許可、またはアイデンティティベースの IAM ポリシーを使用します。IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

> **Note**
>
> 信頼できる ID 伝達が有効になっているワークグループにアクセスするには、Athena GetWorkGroup API アクションの応答によって返される `IdentityCenterApplicationArn` に IAM アイデンティティセンターのユーザーを割り当てる必要があります。

以下の手順は、Athena に固有の手順です。

IAM 固有の情報については、このセクションの最後に表示されているリンク先を参照してください。JSON キャパシティ予約ポリシーの例についての情報は、「ワークグループのポリシーの例」を参照してください。

IAM コンソールのビジュアルエディタを使用してワークグループポリシーを作成する

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[Policies] (ポリシー)、[Create policy] (ポリシーの作成) の順にクリックします。
3. [Visual editor] (ビジュアルエディタ) タブで、[Choose a service] (サービスの選択) をクリックします。次に、ポリシーに追加する Athena を選択します。
4. [Select actions] (アクションの選択) を選択し、ポリシーに追加するアクションを選択します。ビジュアルエディタが Athena で利用できるアクションを表示します。詳細については、「サービス承認リファレンス」の「Amazon Athena 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。
5. [Add actions] (アクションの追加) をクリックして特定のアクションを入力、またはワイルドカード (*) を使用して複数のアクションを指定します。

デフォルトでは、作成しているポリシーが選択するアクションを許可します。Athena 内の workgroup リソースに対するリソースレベルのアクセス許可をサポートするアクションを 1 つ、または複数選択すると、エディタが workgroup リソースをリストします。

6. [リソース] を選択して、ポリシーの特定のワークグループを指定します。JSON ワークグループポリシーの例については、「ワークグループのポリシーの例」を参照してください。
7. 以下のように workgroup リソースを指定します。

```
arn:aws:athena:<region>:<user-account>:workgroup/<workgroup-name>
```

8. [Review policy] (ポリシーの確認) をクリックして、作成するポリシーの [Name] (名前) と [Description (説明) (オプション) を入力します。ポリシー概要を確認して、意図したアクセス許可を付与したことを確認します。
9. [Create Policy] (ポリシーの作成) をクリックして、新しいポリシーを保存します。
10. このアイデンティティベースのポリシーをユーザー、グループ、またはロールにアタッチします。

詳細については、「サービス認証リファレンス」と「IAM ユーザーガイド」で以下のトピックを参照してください。

- Amazon Athena のアクション、リソース、および条件キー
- ビジュアルエディタでのポリシーの作成
- IAM ポリシーの追加と削除
- リソースへのアクセスの制御

JSON ワークグループポリシーの例については、「ワークグループのポリシーの例」を参照してください。

Amazon Athena アクションの完全なリストについては、「Amazon Athena API リファレンス」の API アクション名を参照してください。

ワークグループのポリシーの例

このセクションには、ワークグループに対するさまざまなアクションを有効にするために使用できるポリシーの例が含まれています。IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

ワークグループは、Athena によって管理されている IAM リソースです。そのため、ワークグループポリシーが入力として workgroup を実行するアクションを使用する場合、ワークグループの ARN を次のように指定する必要があります。

```
"Resource": [arn:aws:athena:<region>:<user-account>:workgroup/<workgroup-name>]
```

<workgroup-name> は、ワークグループの名前です。たとえば、test_workgroup という名前のワークグループの場合は、次のようにリソースとして指定します。

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"]
```

mazon Athena アクションの完全なリストについては、Amazon Athena API リファレンスの API アクション名を参照してください。IAM ポリシーの詳細については、「IAM ユーザーガイド」で「ビジュアルエディタでのポリシーの作成」を参照してください。ワークグループ用の IAM ポリシーの作成に関する詳細については、「 ワークグループにアクセスするための IAM ポリシー」を参照してください。

- Example policy for full access to all workgroups
- Example policy for full access to a specified workgroup
- Example policy for running queries in a specified workgroup
- Example policy for running queries in the primary workgroup
- Example policy for management operations on a specified workgroup
- Example policy for listing workgroups
- Example policy for running and stopping queries in a specific workgroup
- Example policy for working with named queries in a specific workgroup
- Example policy for working with Spark notebooks

Example すべてのワークグループへのフルアクセスのポリシーの例

次のポリシーでは、アカウントに存在している可能性があるすべてのワークグループリソースへのフルアクセスを許可します。アカウントの他のすべてのユーザーのワークグループを管理する必要があるユーザーにこのポリシーを使用することをお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "athena:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Example 指定したワークグループへのフルアクセスのポリシーの例

次のポリシーでは、workgroupA という名前の 1 つの特定のワークグループリソースへのフルアクセスを許可します。特定のワークグループを完全に制御しているユーザーにこのポリシーを使用できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions",
                "athena:ListWorkGroups",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
```

```
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:DeleteWorkGroup",
                "athena:UpdateWorkGroup",
                "athena:GetWorkGroup",
                "athena:CreateWorkGroup"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        }
    ]
}
```

Example 指定したワークグループでクエリを実行するためのポリシーの例

次のポリシーでは、ユーザは指定された workgroupA でクエリを実行し、それらを表示することを許可されています。ユーザーは、ワークグループの更新や削除など、ワークグループ自体の管理タスクを実行することはできません。

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions",
```

```
                "athena:ListWorkGroups",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        }
    ]
}
```

Example プライマリワークグループでクエリを実行するためのポリシーの例

上記の例を変更して、特定のユーザーがプライマリワークグループでクエリを実行できるようにすることができます。

> **Note**
>
> 指定したワークグループでクエリを実行するように設定されているすべてのユーザーに、このプライマリワークグループのリソースを追加することをお勧めします。このリソースをワークグループのユーザーポリシーに追加すると、指定したワークグループが削除されたり無効になったりした場合に役立ちます。この場合、プライマリワークグループでクエリを実行し続けることができます。

アカウントのユーザーがプライマリワークグループでクエリを実行できるようにするには、次の例のように、プライマリワークグループの ARN を含む行を Example policy for running queries in a specified workgroup のリソースセクションに追加します。

```
arn:aws:athena:us-east-1:123456789012:workgroup/primary"
```

Example 指定したワークグループに対する管理オペレーションのポリシーの例

次のポリシーでは、ユーザはワークグループ test_workgroup の作成、削除、詳細の取得、および更新を許可されています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateWorkGroup",
                "athena:GetWorkGroup",
                "athena:DeleteWorkGroup",
                "athena:UpdateWorkGroup"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
            ]
```

```
        }
    ]
}
```

Example ワークグループを一覧表示するためのポリシーの例

次のポリシーでは、すべてのワークグループをすべてのユーザーが一覧表示できるようにします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 特定のワークグループでクエリを実行および停止するためのポリシーの例

このポリシーでは、ユーザーは、ワークグループでクエリを実行できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "athena:StopQueryExecution"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
            ]
        }
    ]
}
```

Example 特定のワークグループで名前付きクエリを使用するポリシーの例

次のポリシーでは、ユーザーは指定されたワークグループの名前付きクエリに関する情報を作成、削除、および取得するアクセス権限を持っています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:DeleteNamedQuery"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
     ]
        }
    ]
}
```

Example Athena で Spark ノートブックを操作するためのポリシーの例

Athena で Spark ノートブックを操作するには、次のようなポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreatingWorkGroupWithDefaults",
            "Action": [
                "athena:CreateWorkGroup",
                "s3:CreateBucket",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:GetBucketLocation",
                "athena:ImportNotebook"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/Demo*",
```

```
                "arn:aws:s3:::123456789012-us-east-1-athena-results-bucket-*",
                "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-*",
                "arn:aws:iam::123456789012:policy/service-role/
AWSAthenaSparkRolePolicy-*"
            ]
        },
        {
            "Sid": "AllowRunningCalculations",
            "Action": [
                "athena:ListWorkGroups",
                "athena:GetWorkGroup",
                "athena:StartSession",
                "athena:CreateNotebook",
                "athena:ListNotebookMetadata",
                "athena:ListNotebookSessions",
                "athena:GetSessionStatus",
                "athena:GetSession",
                "athena:GetNotebookMetadata",
                "athena:CreatePresignedNotebookUrl"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/Demo*"
        },
        {
            "Sid": "AllowListWorkGroupAndEngineVersions",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

ワークグループ設定

各ワークグループには次の設定があります。

- 一意の名前。英数字、ダッシュ、アンダースコアを含む 1 ~ 128 文字を含めることができます。ワークグループを作成したら、その名前を変更することはできません。ただし、同じ設定と別の名前で新しいワークグループを作成することはできます。

- ワークグループで実行されているすべてのクエリに適用される設定。この 8 つの原則は以下のとおりです。
  - このワークグループで実行されるすべてのクエリのクエリ結果を保存するための Amazon S3 の場所。この場所は、作成時にワークグループに指定する前に存在している必要があります。Amazon S3 バケットを作成する方法については、「Creating a bucket」(バケットを作成する) を参照してください。
  - クエリ結果に対するバケット所有者のコントロール — Simple Storage Service (Amazon S3) クエリ結果バケットの所有者が、バケットに書き込まれる新しいオブジェクトに対するフルコントロールを所有するかどうか。たとえば、クエリ結果の場所を別のアカウントが所有している場合に、クエリ結果の所有権とフルコントロールを他のアカウントに付与することができます。
  - 予想されるバケット所有者 — クエリ結果のバケット所有者になることが予想される AWS アカウント の ID。これは追加のセキュリティ対策です。バケット所有者のアカウント ID がここで指定した ID と一致しない場合、バケットに出力できません。詳細については、「Amazon S3 ユーザーガイド」の「バケット所有者条件によるバケット所有者の確認」を参照してください。

    > **Note**
    >
    > 予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。これは、外部 Simple Storage Service (Amazon S3) バケット内のデータソースの場所、`CTAS` や `INSERT INTO` の書き込み先のテーブルの場所、`UNLOAD` ステートメントの出力場所、串刺検索のバケットのスピルオペレーション、別のアカウントのテーブルに対して実行される `SELECT` クエリなど、他の Amazon S3 ロケーションには適用されません。

  - すべてのワークグループクエリに暗号化を使用する場合は、暗号化設定。ワークグループのすべてのクエリを暗号化できます。一部のクエリだけを暗号化することはできません。暗号化されているか、または暗号化されていないクエリを含めるには、別々のワークグループを作成することをお勧めします。

さらに、ワークグループはクライアント側の設定を上書きできます。ワークグループのリリース前に、結果の場所と暗号化オプションを JDBC または ODBC ドライバーのパラメータとして、または Athena コンソールの [Properties] (プロパティ) タブで指定することができます。これらの設定は、API オペレーションを介して直接指定することもできました。これらの設定は、「クライアント側の設定」と呼ばれています。ワークグループでは、これらの設定をワークグループレベルで設定し、クライアントレベルで使用可能なオプションを制御できます。ワークグループレベルの設定を強制することで、ユーザーがクライアント側の設定を個別に構成する必要もなくなります。ワークグ

ループで [クライアント側の設定を上書き] オプションを選択すると、クエリはワークグループ設定を使用し、クライアント側の設定を無視します。

[Override Client-Side Settings (クライアント側設定の上書き)] が選択されていると、ユーザーには設定が変更されたことがコンソールで通知されます。ワークグループ設定がこのように強制されている場合、ユーザーは対応するクライアント側の設定を省略できます。その後、コンソールで実行されるクエリは、クライアント側の設定があっても、ワークグループの設定を使用します。また、ワークグループ内のクエリが、AWS CLI、API オペレーション、JDBC または ODBC ドライバーを介して実行される場合、クエリ結果の場所や暗号化などのクライアント側の設定は、ワークグループ設定によってオーバーライドされます。ワークグループの設定を確認するには、ワークグループの詳細を表示します。

ワークグループのクエリに対してクエリ制限を設定することもできます。

ワークグループ設定がクライアント側の設定を上書きする

[Create workgroup (ワークグループの作成)] および [ワークグループの編集] ダイアログには、[Override client-side settings (クライアント側設定の上書き)] というタイトルのフィールドがあります。このフィールドは、デフォルトでは選択されていません。選択したかどうかに応じて、Athena が以下を実行します。

- [クライアント側設定の上書き] が選択されていない場合、ワークグループ設定はクライアントレベルで強制されません。ワークグループでクライアント側の設定を上書きするオプションが選択されていない場合、Athena はワークグループで実行されるすべてのクエリに、クエリ結果の場所、予想されるバケット所有者、暗号化、およびクエリ結果のバケットに書き込むオブジェクトの制御に関する設定を含む、クライアント側の設定を使用します。各ユーザーは、コンソールの [設定] メニューで自分の設定を指定できます。クライアント側の設定がされていない場合は、ワークグループ全体の設定が適用されます。AWS CLI、API アクション、または JDBC および ODBC ドライバーを使用して、クライアント側の設定を上書きしないワークグループでクエリを実行する場合、クエリにはクエリで指定した設定が使用されます。
- [クライアント側の設定の上書き] を選択した場合、ワークグループ設定はワークグループのすべてのクライアントにワークグループレベルで実施されます。ワークグループでクライアント側の設定を上書きするオプションを選択すると、Athena はワークグループで実行されるすべてのクエリにワークグループの設定を使用します。これには、クエリ結果の場所、予想バケット所有者、暗号化、クエリ結果バケットに書き込むオブジェクトの制御に関する設定が含まれます。ワークグループ設定は、コンソール、API アクション、または JDBC または ODBC ドライバーを使用するときにクエリに指定したクライアント側の設定よりも優先されます。

クライアント側の設定を上書きした場合、次にお客様または他のワークグループユーザーが Athena コンソールを開くと、Athena はワークグループのクエリでワークグループの設定を使用することが通知され、この変更を確認するよう求められます。

> **Important**
>
> API アクション、AWS CLI、または JDBC ドライバーと ODBC ドライバーを使用して、クライアント側の設定を上書きするワークグループ内のクエリを実行する場合は、クエリのクライアント側の設定を省略するか、ワークグループの設定と一致するように更新することを確認します。クエリでクライアント側の設定を指定し、その設定を上書きするワークグループで実行した場合、クエリは実行されますが、ワークグループの設定が使用されます。ワークグループの設定の表示に関する詳細は、「ワークグループの詳細の表示」を参照してください。

ワークグループの管理

https://console.aws.amazon.com/athena/ で、以下のタスクを実行できます。

Statement	説明
ワークグループの作成	新しいワークグループを作成します。
ワークグループの編集	ワークグループを編集して設定を変更します。ワークグループの名前を変更することはできませんが、同じ設定と別の名前で新しいワークグループを作成することはできます。
ワークグループの詳細の表示	名前、説明、データ使用量の制限、クエリ結果の場所、予想されるクエリ結果のバケット所有者、暗号化、クエリ結果バケットに書き込まれるオブジェクトのコントロールなど、ワークグループの詳細が表示されます。[Override client-side settings (クライアント側設定の上書き)] がオンになっている場合、このワークグループがその設定を強制しているかどうかも確認できます。
ワークグループの削除	ワークグループを削除します。ワークグループ、クエリ履歴、保存されたクエリを削除すると、ワークグループの設定とクエリごとのデータ制限コントロールが削除されます。ワークグループ全体のデータ制限制御は CloudWatch で保持され、個別に削除することができます。

Statement	説明
	プライマリワークグループは削除できません。
ワークグループの切り替え	アクセス権があるワークグループを切り替えます。
ワークグループ間で保存されたクエリのコピー	ワークグループ間で保存されたクエリをコピーします。これは、例えばプレビューワークグループでクエリを作成し、それを非プレビューワークグループで使用できるようにしたい場合などに実行できます。
ワークグループの有効化および無効化	ワークグループを有効または無効にします。ワークグループを無効にすると、そのユーザーはクエリを実行したり、新しい名前付きクエリを作成したりできません。アクセスできる場合でも、メトリクス、データ使用制限コントロール、ワークグループの設定、クエリ履歴、保存されたクエリを表示できます。
クエリを実行するワークグループの指定	クエリを実行する前に、使用するワークグループを Athena に指定する必要があります。ワークグループに対するアクセス権限が必要です。
IAM アイデンティティセンター認証を使用する Athena ワークグループを作成する	Athena で IAM アイデンティティセンター ID を使用するには、IAM アイデンティティセンターが有効なワークグループを作成する必要があります。ワークグループを作成したら、IAM アイデンティティセンターのコンソールまたは API を使用して IAM アイデンティティセンターのユーザーまたはグループをワークグループに割り当てることができます。

ワークグループの作成

ワークグループを作成するには、`CreateWorkgroup` API アクションへのアクセス権限が必要です。「ワークグループとタグへのアクセス」および「 ワークグループにアクセスするための IAM ポリシー」を参照してください。タグを追加している場合は、`TagResource` にアクセス権限を追加する必要もあります。「ワークグループのタグポリシーの例」を参照してください。

コンソールでワークグループを作成するには

1. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

2. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
3. [Workgroups] (ワークグループ) ページで、[Create workgroup] (ワークグループを作成する) をクリックします。
4. [Create workgroup] (ワークグループを作成する) ページで、次のようにフィールドに入力します。

フィールド	説明
ワークグループ名	必須。ワークグループの一意の名前を入力します。1 ~ 128 文字で入力してください。(A ~ Z、a ~ z、0 ~ 9、_、-、.)。この名前は変更できません。
説明	オプション。ワークグループの説明を入力します。最大 1024 文字を含めることができます。
[Choose the type of engine] (エンジンのタイプを選択する)	Amazon S3 のデータに対してアドホック SQL クエリを実行する場合、または事前構築済みのデータソースコネクタを使用して、Amazon S3 の外部のさまざまなデータソースに対してフェデレーテッドクエリを実行する場合には、[Athena SQL] を選択します。Athena クエリエディタ、AWS CLI、または Athena API を使用してクエリを実行できます。

フィールド	説明
	Python と Apache Spark を使用して Jupyter Notebook アプリケーションを作成、編集、実行する場合には、[Apache Spark] を選択します。Jupyter Notebook には、コード、テキスト、マークダウン、数学、プロット、リッチメディアを含むセルのリストが含まれています。セルは Athena のインタラクティブなノートブックセッションで、計算として順番に実行されます。Spark 対応ワークグループの作成と設定については、「Athena での Spark 対応ワークグループの作成」を参照してください。 ワークグループを作成すると、その分析エンジンをアップグレードできますが (例えば、Athena エンジンバージョン 2 から Athena エンジンバージョン 3 へ)、そのエンジンタイプは変更できません。例えば、Athena エンジンのバージョン 3 のワークグループを PySpark エンジンのバージョン 3 のワークグループに変更することはできません。
クエリエンジンを更新する	新しい Athena エンジンバージョンがリリースされるときに、ワークグループを更新する方法を選択します。Athena にワークグループを更新するタイミングを決定させる、またはエンジンバージョンを手動で選択することができます。詳細については、「Athena エンジンのバージョニング」を参照してください。
[Authentication mode] (認証モード)	[AWS Identity and Access Management (IAM)] を選択してワークグループに IAM 認証またはフェデレーションを使用します。Microsoft Active Directory などの SAML 2.0 ID プロバイダーのユーザーやグループなどのワークフォースアイデンティティをサポートしたい場合は、[IAM アイデンティティセンター] を選択してください。詳細については、「AWS IAM Identity Center ユーザーガイド」の「Trusted identity propagation across applications」を参照してください。

フィールド	説明
IAM アイデンティティセンターにアクセスするためのサービスロール	Athena には、ユーザーに代わって IAM アイデンティティセンターにアクセスするための IAM アクセス許可が必要です。IAM サービスロールの詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
クエリ結果の場所	オプション。Amazon S3 バケットまたはプレフィックスへのパスを入力します。このバケットとプレフィックスは、指定する前に存在している必要があります。 **Note** コンソールでクエリを実行する場合、クエリ結果の場所の指定はオプションです。ワークグループまたは [Settings (設定)] で指定しない場合、Athena はデフォルトのクエリ結果の場所を使用します。API またはドライバを使用してクエリを実行する場合、個々のクエリには OutputLocation、またはワークグループには WorkGroupConfiguration を使って、2 つの場所の少なくとも 1 つにクエリ結果の場所を指定する必要があります。

フィールド	説明
予想されるバケット所有者	オプション。出力先バケットの所有者になると予想される AWS アカウント の ID を入力します。これは追加のセキュリティ対策です。バケット所有者のアカウント ID がここで指定した ID と一致しない場合、バケットに出力できません。詳細については、「Amazon S3 ユーザーガイド」の「バケット所有者条件によるバケット所有者の確認」を参照してください。 **Note** 予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。これは、外部 Simple Storage Service (Amazon S3) バケット内のデータソースの場所、`CTAS` や `INSERT INTO` の書き込み先のテーブルの場所、`UNLOAD` ステートメントの出力場所、串刺検索のバケットのスピルオペレーション、別のアカウントのテーブルに対して実行される `SELECT` クエリなど、他の Amazon S3 ロケーションには適用されません。

フィールド	説明
クエリ結果に対するフルコントロールをバケット所有者に割り当てる	このフィールドは、デフォルトでは選択されていません。これを選択し、クエリ結果の場所バケットで ACL が有効になっている場合、クエリ結果に対するフルコントロールアクセスがバケット所有者に付与されます。たとえば、クエリ結果の場所を別のアカウントが所有している場合に、このオプションを使用して、クエリ結果の所有権とフルコントロールを他のアカウントに付与することができます。 バケットの S3 オブジェクトの所有権の設定が [Bucket owner preferred] (バケット所有者推奨) となっている場合、バケット所有者は、このワークグループから書き込まれたすべてのクエリ結果オブジェクトも所有します。たとえば、外部アカウントのワークグループでこのオプションが有効になっており、そのクエリ結果の場所がアカウントの Simple Storage Service (Amazon S3) バケットに設定されている場合に、このバケットの S3 オブジェクトの所有権が [Bucket owner preferred] (バケット所有者推奨) と設定されていると、外部ワークグループのクエリ結果を所有し、フルコントロールアクセスを持つことになります。 クエリ結果バケットの S3 オブジェクトの所有権設定が [Bucket owner enforced] (バケット所有者の強制) となっているときにこのプションを選択しても、効果はありません。詳細については、「Simple Storage Service (Amazon S3) ユーザーガイド」の「オブジェクトの所有権の設定」を参照してください。
クエリ結果の暗号化	オプション。Amazon S3 に保存されている結果を暗号化します。選択すると、ワークグループのすべてのクエリは暗号化されます。 選択すると、[Encryption type (暗号化タイプ)]、[Encryption key (暗号化キー)] を選択して、[KMS Key ARN (KMS キー ARN)] を入力できます。 キーがない場合は、AWS KMS コンソールを開いて作成します。詳細については、「デベロッパーガイド」の「AWS Key Management Serviceキーの作成」を参照してください。

フィールド	説明
######を最小暗号化として設定	オプション。このオプションを選択すると、ワークグループのすべてのユーザーに対して、クエリ結果に暗号化の最小限のタイプを強制できます。このオプションを選択すると、暗号化タイプの階層を示す表が表示されます。この表には、特定の暗号化タイプを最小限として指定した場合に、ワークグループユーザが使用できる暗号化タイプも表示されます。このオプションを使用するには、[クライアント側設定の上書き] を選択しないでください。 詳細については、「ワークグループの最小暗号化の設定」を参照してください。
S3 Access Grants を有効にする	認証モードとして [IAM アイデンティティセンター] を選択すると、このフィールドがデフォルトで選択されます。このオプションを選択すると、IAM アイデンティティセンターのユーザーまたはグループベースのアクセス権限が Amazon S3 ロケーションに適用されます。
ユーザー ID ベースの S3 プレフィックスを作成する	このオプションを選択すると、Athena はクエリ結果を保存するときに Amazon S3 プレフィックスを作成します。プレフィックスはユーザーの IAM アイデンティティセンターのユーザー ID に基づいています。
[Publish query metrics to CloudWatch] (クエリメトリクスを CloudWatch に発行)	このフィールドは、デフォルトで選択されています。クエリメトリクスを CloudWatch に発行します。「CloudWatch メトリクスによる Athena クエリのモニタリング」を参照してください。
[Override client-side settings (クライアント側設定の上書き)]	このフィールドは、デフォルトでは選択されていません。選択すると、ワークグループ設定はワークグループのすべてのクエリに適用され、クライアント側の設定を上書きします。詳細については、「ワークグループ設定がクライアント側の設定を上書きする」を参照してください。

フィールド	説明
リクエスタ支払い S3 バケット	オプション。ワークグループユーザーが、リクエスタ支払いとして設定された Amazon S3 バケットに保存されているデータでクエリを実行する場合は、[Turn on queries on requester pays buckets in Amazon S3] (Amazon S3 のリクエスタ支払いバケットでのクエリをオンにする) を選択します。クエリを実行しているユーザーのアカウントには、クエリに関連付けられた該当するデータアクセスの料金とデータ転送料金が請求されます。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「リクエスタ支払いバケット」を参照してください。
[Manage per query data usage control] (クエリごとのデータ使用量の制御を管理)	オプション。クエリがスキャンできるデータの最大量の制限を設定します。ワークグループに対してクエリごとの制限を 1 つだけ設定できます。この制限は、ワークグループのすべてのクエリに適用され、クエリが制限を超えるとキャンセルされます。詳細については、「データ使用量の制御制限の設定」を参照してください。
ワークグループのデータ使用に関するアラート	オプション。このワークグループで実行しているクエリが特定の期間内に指定した量のデータをスキャンする際、複数のアラートしきい値を設定します。アラートは Amazon CloudWatch アラームを使用して実装され、ワークグループ内のすべてのクエリに適用されます。詳細については、『Amazon CloudWatch 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。
タグ	オプション。ワークグループに 1 つ以上のタグを追加します。タグは、Athena ワークグループリソースに割り当てるラベルです。キーと値から構成されます。AWS タグ付けのベストプラクティスを使用して、一貫性のあるタグのセットを作成し、目的、所有者、環境に基づいてワークグループを分類します。IAM ポリシーでタグを使用したり、課金コストを管理することもできます。同じワークグループに重複したタグキーを使用しないでください。詳細については、「the section called “リソースのタギング”」を参照してください。

5. [Create workgroup] (ワークグループの作成) を選択します。ワークグループが [Workgroups] (ワークグループ) ページのリストに表示されます。

また、CreateWorkGroup API オペレーションを使用してワークグループを作成することもできます。

> **Important**
>
> ワークグループを作成したら、ワークグループ関連のアクションを実行できるようにするワークグループにアクセスするための IAM ポリシー IAM を作成します。

ワークグループの編集

ワークグループを編集するには、UpdateWorkgroup API オペレーションへのアクセス権限が必要です。「ワークグループとタグへのアクセス」および「 ワークグループにアクセスするための IAM ポリシー」を参照してください。タグを追加または編集している場合、TagResource へのアクセス権限も必要です。「ワークグループのタグポリシーの例」を参照してください。

コンソールでワークグループを編集するには

1. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
2. [Workgroups] (ワークグループ) ページで、編集するワークグループのボタンを選択します。
3. [Actions]、[Edit] の順に選択します。
4. 必要に応じて、フィールドの値を変更します。フィールドのリストについては、「ワークグループの作成」を参照してください。ワークグループの名前を除くすべてのフィールドを変更できます。名前を変更する必要がある場合は、新しい名前と同じ設定で別のワークグループを作成してください。
5. [Save changes] (変更の保存) を選択します。更新されたワークグループが [Workgroups] (ワークグループ) ページのリストに表示されます。

ワークグループの詳細の表示

各ワークグループの詳細を表示できます。詳細には、ワークグループの名前、説明、有効になっているか無効になっているか、ワークグループで実行されるクエリに使用される設定 (クエリ結果の場所や予想されるバケット所有者、暗号化、クエリ結果バケットに書き込まれるオブジェクトのコントロールなど) が含まれます。ワークグループにデータ使用量の制限がある場合は、それらも表示されます。

ワークグループの詳細を表示するには

1. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
2. [Workgroups] (ワークグループ) ページで、表示するワークグループのリンクを選択します。ワークグループの [Overview Details] (概要の詳細) ページが表示されます。

ワークグループの削除

アクセス権限がある場合は、ワークグループを削除できます。プライマリワークグループは削除できません。

アクセス権限がある場合は、いつでも空のワークグループを削除できます。保存したクエリが含まれているワークグループを削除することもできます。この場合、ワークグループの削除に進む前に、保存されたクエリが削除されることを Athena が警告します。

ワークグループにいる間にワークグループを削除した場合、コンソールは焦点をプライマリワークグループに切り替えます。アクセスできる場合は、クエリを実行して設定を表示します。

ワークグループを削除すると、その設定とクエリごとのデータ制限コントロールは削除されます。ワークグループ全体のデータ制限制御は CloudWatch で保持され、そこで必要に応じて削除することができます。

> **Important**
>
> ワークグループを削除する前に、そのユーザーが引き続きクエリを実行できる他のワークグループに属していることを確認してください。ユーザーの IAM ポリシーが、ユーザーにこのワークグループのみでクエリの実行を許可していた場合にそのワークグループを削除すると、ユーザーはクエリを実行する許可を失います。詳細については、「Example policy for running queries in the primary workgroup」を参照してください。

コンソールでワークグループを削除するには

1. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
2. [Workgroups] (ワークグループ) ページで、削除するワークグループのボタンを選択します。
3. [アクション]、[削除] の順に選択します。

4. [Delete workgroup] (ワークグループを削除する) の確認プロンプトで、ワークグループの名前と入力し、[Delete] (削除) を選択します。

API オペレーションでワークグループを削除するには、`DeleteWorkGroup` アクションを使用します。

ワークグループの切り替え

両方のワークグループにアクセス権限がある場合は、あるワークグループから別のワークグループに切り替えることができます。

各ワークグループで最大 10 個のクエリタブを開くことができます。ワークグループを切り替えると、クエリタブは最大 3 つのワークグループに対して開いたままになります。

ワークグループを切り替えるには

1. Athena コンソールで、右上の [Workgroups] (ワークグループ) オプションを使用して、ワークグループを選択します。
2. [Workgroup ***workgroup-name*** setting] (ワークグループ (ワークグループ名) の設定) ダイアログボックスが表示されたら、[Acknowledge] (承認する) を選択します。

[Workgroup] (ワークグループ) オプションには、切り替えたワークグループの名前が表示されます。これで、このワークグループでクエリを実行できます。

ワークグループ間で保存されたクエリのコピー

現在、Athena コンソールには、保存されたクエリを 1 つのワークグループから別のワークグループに直接コピーするオプションはありませんが、以下の手順を使用することで、同じタスクを手動で実行することができます。

ワークグループ間で保存されたクエリをコピーする

1. Athena コンソールで、クエリをコピーするワークグループから、[Saved queries] (保存されたクエリ) タブをクリックします。
2. コピーする保存されたクエリのリンクを選択します。Athena により、クエリエディタでクエリが開かれます。
3. クエリエディタで、クエリテキストを選択してから **Ctrl+C** を押してそれをコピーします。
4. 宛先ワークグループに切り替えるか、ワークグループを作成してから、それに切り替えます。

5. クエリエディタで新しいタブを開き、**Ctrl+V** を押して新しいタブにテキストを貼り付けます。
6. クエリエディタで [Save as] (名前を付けて保存) をクリックして、クエリを宛先のワークグループに保存します。
7. [Choose name] (名前の選択) ダイアログボックスで、クエリの名前とオプションの説明を入力します。
8. [Save] を選択します。

ワークグループの有効化および無効化

アクセス権限がある場合は、API オペレーションを使用するか、JDBC および ODBC ドライバーを使用して、コンソールでワークグループを有効化または無効化できます。

ワークグループを有効化または無効化するには

1. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
2. [Workgroup] (ワークグループ) ページで、ワークグループのリンクを選択します。
3. 右上の [Enable workgroup] (ワークグループの有効化) または [Disable workgroup] (ワークグループの無効化) を選択します。
4. 確認プロンプトで、[Enable] (有効化) または [Disable] (無効化) を選択します。ワークグループを無効にすると、そのユーザーはその中でクエリを実行したり、新しい名前付きクエリを作成したりできません。ワークグループを有効にした場合、ユーザーはそれを使用してクエリを実行できます。

クエリを実行するワークグループの指定

使用するワークグループを指定するには、ワークグループに対するアクセス許可が必要です。

使用するワークグループを指定するには

1. 使用する予定のワークグループでクエリを実行できるアクセス権限であることを確認します。詳細については、「the section called “ ワークグループにアクセスするための IAM ポリシー”」を参照してください。
2. ワークグループを指定するには、次のいずれかのオプションを使用してください。

   - Athena コンソールを使用している場合は、ワークグループを切り替えてワークグループを設定します。

- Athena API オペレーションを使用している場合は、API アクションにワークグループ名を指定します。例えば、以下のように StartQueryExecution でワークグループ名を設定できます。

```
StartQueryExecutionRequest startQueryExecutionRequest = new
 StartQueryExecutionRequest()
              .withQueryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
              .withQueryExecutionContext(queryExecutionContext)
              .withWorkGroup(WorkgroupName)
```

- JDBC または ODBC ドライバーを使用している場合、Workgroup 設定パラメータを使用して接続文字列にワークグループ名を設定します。ドライバーがワークグループ名を Athena に渡します。次の例のように、接続文字列にワークグループパラメータを指定します。

```
jdbc:awsathena://AwsRegion=<AWSREGION>;UID=<ACCESSKEY>;
PWD=<SECRETKEY>;S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/<athena-
output>-<AWSREGION>/;
Workgroup=<WORKGROUPNAME>;
```

ワークグループの最小暗号化の設定

Athena SQL ワークグループの管理者として、ワークグループからのすべてのクエリ結果に Amazon S3 で最小限の暗号化を強制できます。この機能を使用すると、クエリ結果が暗号化されていない状態で Amazon S3 バケットに保存されないようにすることができます。

最小暗号化が有効になっているワークグループのユーザーがクエリを送信する場合、設定した最小レベル、または使用可能な場合はそれ以上のレベルの暗号化しか設定できません。Athena は、ユーザーがクエリを実行したときに指定されたレベル、またはワークグループに設定されたレベルのいずれかでクエリ結果を暗号化します。

以下のレベルが使用可能です。

- Basic – Amazon S3 マネージドキー(SSE-S3)を用いたサーバー側の暗号化。
- Intermediate – KMS マネージドキー(SSE-KMS)を用いたサーバー側の暗号化。
- Advanced – KMS マネージドキー(CSE-KMS)を用いたクライアント側の暗号化。

考慮事項と制約事項

- Apache Spark 対応のワークグループでは、暗号化の下限設定機能は使用できません。

- 最小限の暗号化機能は、ワークグループが「クライアント側の設定を上書き」オプションを有効にしていない場合にのみ機能します。
- ワークグループで [クライアント側の設定を上書き] オプションが有効になっている場合は、ワークグループの暗号化設定が優先され、最小暗号化設定は影響を受けません。
- この機能を有効にするためのコストはかかりません。

ワークグループの最小限の暗号化を有効にする

ワークグループを作成または更新するときに、Athena SQL ワークグループからのクエリ結果の最小暗号化レベルを有効にできます。これを行うには、Athena コンソール、Athena API、または AWS CLI を使用できます。

Athena コンソールを使用して最小限の暗号化を有効にします

Athena コンソールを使用してワークグループの作成または編集を開始するには、「ワークグループの作成」または「ワークグループの編集」を参照してください。ワークグループを設定する際は、以下の手順で最小限の暗号化を有効にしてください。

ワークグループのクエリ結果において最小暗号化レベルを設定するには

1. [追加設定] セクションで、[設定] を展開します。
2. [クライアント側の設定を上書き]オプションを解除するか、選択されていないことを確認します。
3. [追加設定] セクションで、[クエリ結果設定] を展開します。
4. [クエリ結果を暗号化] オプションを選択します。
5. [暗号化タイプ] では、ワークグループのクエリ結果に Athena が使用する暗号化方法 (SSE_S3、SSE_KMS、または CSE_KMS) を選択します。これらの暗号化タイプは、基本、中級、および上級のセキュリティレベルに対応しています。
6. 暗号化の最小レベルとして選択した暗号化方法をすべてのユーザーに強制するには、「**#####** を最小暗号化として設定」を選択します。

   このオプションを選択すると、選択した暗号化タイプが最小になったときにユーザーに許可される暗号化階層と暗号化レベルが表に表示されます。
7. ワークグループを作成するか、ワークグループ設定を更新したら、[ワークグループを作成] または [変更を保存] を選択します。

Athena API または AWS CLI を使用して、最小限の暗号化を有効にする

CreateWorkGroup API または UpdateWorkgroup API を使用して Athena SQL ワークグループを作成または更新する場合は、EnforceWorkgroupConfiguration を `false` に、EnableMinumEncryptionConfiguration を `true` に設定し、EncryptionOption を使用して暗号化の種類を指定します。

AWS CLI では、`create-work-group` または `update-work-group` コマンドを `--configuration` または `--configuration-updates` パラメータとともに使用して API のパラメータに対応するオプションを指定します。

IAM アイデンティティセンターが有効な Athena ワークグループの使用

AWS IAM Identity Center の信頼できる ID の伝達機能により、従業員の ID を AWS 分析サービス全体で使用できるようになります。信頼できる ID の伝達により、サービス固有の ID プロバイダー設定や IAM ロールのセットアップを行う必要がなくなります。

IAM アイデンティティセンターを使用すると、ワークフォースユーザーとも呼ばれるワークフォース ID のサインインセキュリティを管理できます。IAM アイデンティティセンターでは、ワークフォースユーザーを作成または接続し、すべての AWS アカウントとアプリケーションにまたがるアクセスを一元管理できます。マルチアカウントのアクセス許可を使用して、これらのユーザーに AWS アカウント へのアクセス権を割り当てることができます。アプリケーションの割り当てを使用して、IAM アイデンティティセンターが有効なアプリケーション、クラウドアプリケーション、カスタマーセキュリティアサーションマークアップ言語 (SAML 2.0) アプリケーションへのアクセス権をユーザーに割り当てることができます。詳細については、「AWS IAM Identity Center ユーザーガイド」の「Trusted identity propagation across applications」を参照してください。

現在、Athena SQL は信頼できる ID の伝達をサポートしているため、Amazon EMR Studio と EMR Studio の Athena SQL インターフェイスで同じ ID を使用できます。EMR Studio の Athena SQL で IAM アイデンティティセンター ID を使用するには、Athena で IAM アイデンティティセンターが有効なワークグループを作成する必要があります。その後、IAM アイデンティティセンターのコンソールまたは API を使用して IAM アイデンティティセンターのユーザーまたはグループを IAM アイデンティティセンターが有効なワークグループに割り当てることができます。信頼できる ID 伝達を使用する Athena ワークグループからのクエリは、IAM アイデンティティセンターが有効な EMR Studio の Athena SQL インターフェイスから実行する必要があります。

考慮事項と制約事項

Amazon Athena で信頼できる ID 伝達を使用する際は、以下の点を考慮してください。

- ワークグループの認証方法は、ワークグループの作成後に変更することはできません。

  - 既存の Athena SQL ワークグループは、IAM アイデンティティセンターが有効なワークグループをサポートするように変更することはできません。
  - IAM アイデンティティセンターが有効なワークグループは、リソースレベルの IAM アクセス許可、またはアイデンティティベースの IAM ポリシーをサポートするように変更することはできません。
- 信頼できる ID 伝達が有効になっているワークグループにアクセスするには、Athena GetWorkGroup API アクションの応答によって返される `IdentityCenterApplicationArn` に IAM アイデンティティセンターのユーザーを割り当てる必要があります。
- Amazon S3 Access Grants は、信頼できる ID 伝達アイデンティティを使用するように設定する必要があります。詳細については、「Amazon S3 ユーザーガイド」の「S3 Access Grants and corporate directory identities」を参照してください。
- IAM アイデンティティセンターが有効な Athena ワークグループには、IAM アイデンティティセンターの ID を使用するように Lake Formation を設定する必要があります。設定情報については、「AWS Lake Formation デベロッパーガイドガイド」の「Integrating IAM Identity Center」を参照してください。
- 信頼できる ID 伝達を使用するワークグループでは、デフォルトでクエリは 30 分後にタイムアウトします。クエリのタイムアウトの延長をリクエストすることはできますが、信頼できる ID 伝達ワークグループでクエリを実行できる最大時間は 1 時間です。
- 信頼できる ID 伝達ワークグループにおけるユーザーまたはグループの使用権限の変更は、反映されるまでに最大 1 時間かかる場合があります。
- 信頼できる ID 伝達を使用する Athena ワークグループのクエリは、Athena コンソールから直接実行することはできません。これらは、IAM アイデンティティセンターが有効な EMR Studio の Athena インターフェイスから実行する必要があります。EMR Studio での Athena の使用についての詳細は、「Amazon EMR 管理ガイド」の「Use the Amazon Athena SQL editor in EMR Studio」を参照してください。
- 信頼できる ID 伝達は、以下の Athena 機能との互換性がありません。
  - `aws:CalledVia` コンテキストキー
  - Athena for Spark ワークグループ
  - Athena API へのフェデレーションアクセス
  - Lake Formation と Athena JDBC および ODBC ドライバーを使用した Athena へのフェデレーションアクセス
- Athena で信頼できる ID 伝達を使用できるのは、以下の AWS リージョン のみです。
  - `us-east-2` — 米国東部 (オハイオ)

- us-east-1 — 米国東部 (バージニア北部)
- us-west-1 — 米国西部 (北カリフォルニア)
- us-west-2 — 米国西部 (オレゴン)
- af-south-1 – アフリカ (ケープタウン)
- ap-east-1 – アジアパシフィック (香港)
- ap-southeast-3 – アジアパシフィック (ジャカルタ)
- ap-south-1 – アジアパシフィック (ムンバイ)
- ap-northeast-3 – アジアパシフィック (大阪)
- ap-northeast-2 – アジアパシフィック (ソウル)
- ap-southeast-1 – アジアパシフィック (シンガポール)
- ap-southeast-2 – アジアパシフィック (シドニー)
- ap-northeast-1 – アジアパシフィック (東京)
- ca-central-1 — カナダ (中部)
- eu-central-1 – 欧州 (フランクフルト)
- eu-west-1 – 欧州 (アイルランド)
- eu-west-2 – 欧州 (ロンドン)
- eu-south-1 – 欧州 (ミラノ)
- eu-west-3 – 欧州 (パリ)
- eu-north-1 – 欧州 (ストックホルム)
- me-south-1 – 中東 (バーレーン)
- sa-east-1 – 南米 (サンパウロ)

必要なアクセス許可

Athena コンソールで IAM アイデンティティセンターが有効なワークグループを作成する管理者の IAM ユーザーには、次のポリシーがアタッチされている必要があります。

- AmazonAthenaFullAccess マネージドポリシー。詳細については、「AWS 管理ポリシー: AmazonAthenaFullAccess」を参照してください。
- IAM と IAM アイデンティティセンターのアクションを許可する以下のインラインポリシー

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Action": [
                "iam:createRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:ListRoles",
                "iam:PassRole",
                "identitystore:ListUsers",
                "identitystore:ListGroups",
                "identitystore:CreateUser",
                "identitystore:CreateGroup",
                "sso:ListInstances",
                "sso:CreateInstance",
                "sso:DeleteInstance",
                "sso:DescribeUser",
                "sso:DescribeGroup",
                "sso:ListTrustedTokenIssuers",
                "sso:DescribeTrustedTokenIssuer",
                "sso:ListApplicationAssignments",
                "sso:DescribeRegisteredRegions",
                "sso:GetManagedApplicationInstance",
                "sso:GetSharedSsoConfiguration",
                "sso:PutApplicationAssignmentConfiguration",
                "sso:CreateApplication",
                "sso:DeleteApplication",
                "sso:PutApplicationGrant",
                "sso:PutApplicationAuthenticationMethod",
                "sso:PutApplicationAccessScope",
                "sso:ListDirectoryAssociations",
                "sso:CreateApplicationAssignment",
                "sso:DeleteApplicationAssignment",
                "organizations:ListDelegatedAdministrators",
                "organizations:DescribeAccount",
                "organizations:DescribeOrganization",
                "organizations:CreateOrganization",
                "sso-directory:SearchUsers",
                "sso-directory:SearchGroups",
                "sso-directory:CreateUser"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
```

```
        }
    ]
}
```

IAM アイデンティティセンターが有効な Athena ワークグループの作成

以下の手順は、IAM アイデンティティセンターが有効な Athena ワークグループの作成に関連するステップとオプションを示しています。Athena ワークグループで使用できるその他の設定オプションの説明については、「ワークグループの作成」を参照してください。

Athena コンソールで SSO が有効なワークグループを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
3. [Workgroups] (ワークグループ) ページで、[Create workgroup] (ワークグループを作成する) をクリックします。
4. [ワークグループを作成] ページの [ワークグループ名] に、ワークグループの名前を入力します。
5. [分析エンジン] には、[Athena SQL] のデフォルトを使用してください。
6. [認証] には、[IAM アイデンティティセンター] を選択します。
7. [IAM アイデンティティセンターのアクセス用のサービスロール] で、既存のサービスロールを選択するか、新しいサービスロールを作成します。

   Athena には、ユーザーに代わって IAM アイデンティティセンターにアクセスするための権限が必要です。Athena がこれを行うにはサービスロールが必要です。サービスロールはお客様が管理する IAM ロールであり、AWS のサービスがお客様に代わって AWS の他のサービスにアクセスすることを許可します。詳細については、『IAM ユーザーガイド』の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
8. [クエリ結果の設定] を展開し、[クエリ結果の場所] に Amazon S3 パスを入力または選択します。
9. (オプション) [クエリ結果の暗号化] を選択します。
10. (オプション) [ユーザー ID ベースの S3 プレフィックスを作成する] を選択します。

    IAM アイデンティティセンターが有効なワークグループを作成すると、[S3 Access Grants を有効にする] オプションがデフォルトで選択されます。Amazon S3 Access Grants を使用して、Amazon S3 の Athena クエリ結果の場所 (プレフィックス) へのアクセスを制御することが

できます。Amazon S3 Access Grants の詳細については、「Managing access with Amazon S3 Access Grants」を参照してください。

IAM アイデンティティセンター認証を使用する Athena ワークグループでは、Amazon S3 Access Grants によって管理される ID ベースのクエリ結果の場所の作成を有効にすることができます。これらのユーザー ID ベースの Amazon S3 プレフィックスにより、Athena ワークグループのユーザーは、クエリ結果を同じワークグループ内の他のユーザーから分離しておくことができます。

ユーザープレフィックスオプションを有効にすると、Athena はユーザー ID を Amazon S3 パスプレフィックスとしてワークグループのクエリ結果の出力場所 (例: `s3://DOC-EXAMPLE-BUCKET/${user_id}`) に追加します。この機能を使用するには、`user_id` プレフィックスが付いた場所へのユーザー権限のみを許可するように Access Grants を設定する必要があります。Athena クエリ結果へのアクセスを制限する Amazon S3 Access Grants ロケーションのサンプルロールポリシーについては、「サンプルロールポリシー」を参照してください。

> **Note**
>
> ユーザ ID S3 プレフィックスオプションを選択すると、次のステップで説明されているように、ワークグループのクライアント側設定の上書きオプションが自動的に有効になります。ユーザ ID プレフィックス機能には、クライアント側設定の上書きオプションが必要です。

11. [設定] を展開し、[クライアント側の設定を上書き] が選択されていることを確認します。

    [クライアント側の設定の上書き] を選択した場合、ワークグループ内のすべてのクライアントに対して、ワークグループ設定がワークグループレベルで適用されます。詳細については、「ワークグループ設定がクライアント側の設定を上書きする」を参照してください。

12. (オプション) ワークグループの作成 の説明に従って、必要なその他の構成設定を行います。

13. [Create workgroup] (ワークグループの作成) を選択します。

14. Athena コンソールの [ワークグループ] セクションを使用して、IAM Identity Center ディレクトリからのユーザーまたはグループを IAM Identity Center が有効化された Athena ワークグループに割り当てます。

サンプルロールポリシー

次のサンプルは、Amazon S3 Access Grant ロケーションにアタッチされるロールのポリシーを示すもので、Athena クエリ結果へのアクセスを制限します。

```
{
    "Statement": [{
        "Action": ["s3:*"],
        "Condition": {
            "ArnNotEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringNotEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Deny",
        "Resource": "*",
        "Sid": "ExplicitDenyS3"
    }, {
        "Action": ["kms:*"],
        "Effect": "Deny",
        "NotResource": "arn:aws:kms:${region}:${account}:key/${keyid}",
        "Sid": "ExplictDenyKMS"
    }, {
        "Action": ["s3:ListMultipartUploadParts", "s3:GetObject"],
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION/${identitystore:UserId}/
*",
        "Sid": "ObjectLevelReadPermissions"
    }, {
        "Action": ["s3:PutObject", "s3:AbortMultipartUpload"],
        "Condition": {
            "ArnEquals": {
```

```
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION/${identitystore:UserId}/
*",
        "Sid": "ObjectLevelWritePermissions"
    }, {
        "Action": "s3:ListBucket",
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            },
            "StringLikeIfExists": {
                "s3:prefix": ["${identitystore:UserId}", "${identitystore:UserId}/*"]
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION",
        "Sid": "BucketLevelReadPermissions"
    }, {
        "Action": ["kms:GenerateDataKey", "kms:Decrypt"],
        "Effect": "Allow",
        "Resource": "arn:aws:kms:${region}:${account}:key/${keyid}",
        "Sid": "KMSPermissions"
    }],
    "Version": "2012-10-17"
}
```

Athena ワークグループ API

以下は、Athena ワークグループに使用される REST API オペレーションの一部です。`ListWorkGroups` を除く次のすべてのオペレーションで、ワークグループを指定する必要があります。`StartQueryExecution` のような他のオペレーションでは、ワークグループパラメータは

オプションであり、オペレーションはここには一覧表示されません。オペレーションの完全なリストについては、「Amazon Athena API リファレンス」を参照してください。

- CreateWorkGroup
- DeleteWorkGroup
- GetWorkGroup
- ListWorkGroups
- UpdateWorkGroup

ワークグループのトラブルシューティング

ワークグループのトラブルシューティングを行うには、以下のヒントを参考にしてください。

- アカウントの個々のユーザーのアクセス権限を確認します。それらは、クエリ結果の場所、およびクエリを実行するワークグループへのアクセス権を持っている必要があります。ワークグループを切り替える場合は、両方のワークグループに対するアクセス権限も必要です。詳細については、ワークグループにアクセスするための IAM ポリシー を参照してください。
- Athena コンソールの内容に注目して、どのワークグループでクエリを実行するのかを確認します。ドライバーを使用する場合は、ワークグループを必要なものに設定する必要があります。詳細については、the section called “クエリを実行するワークグループの指定” を参照してください。
- API またはドライバーを使用してクエリを実行する場合は、次の方法のいずれかを使用してクエリ結果の場所を指定する必要があります。個々のクエリについては OutputLocation (クライアント側) を使用します。ワークグループでは WorkGroupConfiguration を使用します。場所がどちらの方法でも指定されていない場合、Athena はクエリランタイムでエラーを発行します。
- クライアント側の設定をワークグループの設定で上書きすると、クエリ結果の場所でエラーが発生する可能性があります。例えば、ワークグループのユーザーに、クエリ結果を保存するための Amazon S3 内のワークグループの場所に対する許可がない場合があります。この場合は、必要な許可を追加します。
- ワークグループは、API オペレーションの動作に変更を加えます。以下の既存の API オペレーションの呼び出しには、アカウントのユーザーが、呼び出しを行うワークグループに対して IAM にリソースベースの許可を持っている必要があります。ワークグループ、およびワークグループのアクションに対するアクセス許可がない場合は、CreateNamedQuery、DeleteNamedQuery、GetNamedQuery、ListNamedQueries、StartQueryExecu (この API アクションはドライバーでの使用のみに利用でき、一般的な使用には利用できません) の API アクションが `AccessDeniedException` をスローします。詳細については、「サービス

承認リファレンス」の「Amazon Athena 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。

BatchGetQueryExecution および BatchGetNamedQuery API オペレーションの呼び出しは、ユーザーがアクセス権を持つワークグループで実行されるクエリに関する情報しか返しません。ユーザがワークグループにアクセスできない場合、これらの API オペレーションは未処理の ID リストの一部として未承認クエリ ID を返します。詳細については、「the section called “ Athena ワークグループ API”」を参照してください。

- クエリが実行されるワークグループが強制されたクエリ結果の場所で設定されている場合は、CTAS クエリに `external_location` を指定しないでください。この場合、Athena がエラーを発行して、`external_location` を指定するクエリを失敗させます。たとえば、クエリ結果の場所に関するクライアント側の設定を上書きし、ワークグループに独自の場所を使用するように強制すると、次のクエリは失敗します。`CREATE TABLE <DB>.<TABLE1> WITH (format='Parquet', external_location='s3://DOC-EXAMPLE-BUCKET/test/') AS SELECT * FROM <DB>.<TABLE2> LIMIT 10;`

次のエラーが表示されることがあります。この表は、ワークグループに関連したエラーのリストを提供し、ソリューションを提案します。

ワークグループのエラー

エラー	表示される場合
「query state CANCELED.」(クエリの状態がキャンセルされました)「Bytes scanned limit was exceeded.」(スキャンされたバイト数が制限を超えました)	クエリがクエリごとのデータ制限に達し、キャンセルされました。読み込むデータを少なくするためにクエリを書き換えることを検討するか、アカウント管理者に連絡してください。
「User: *arn:aws:iam::123456789012:user/abc* is not authorized to perform: athena:StartQueryExecution on resource: *arn:aws:athena:us-east-1:123456789012:workgroup/workgroupname* 」(ユーザー: arn:aws:iam::123456789012:user/abc には実行する権限がありません: リソースでの athena:StartQueryE	ユーザーがワークグループでクエリを実行しましたが、アクセスできません。ワークグループにアクセスできるようにポリシーを更新してください。

エラー	表示される場合
xecution: arn:aws:athena:us-east-1:12 3456789012:workgroup/workgroupname)	
INVALID_INPUT. 「WorkGroup <name> is disabled.」(WorkGroup <name> は無効です)	ユーザーがワークグループでクエリを実行しましたが、そのワークグループが無効になっています。ワークグループは管理者によって無効にされている可能性があります。また、アクセス権限がない可能性もあります。いずれの場合も、ワークグループを変更するアクセス権限を持つ管理者に連絡してください。
INVALID_INPUT. 「WorkGroup <name> is not found.」(WorkGroup <name> が見つかりません)	ユーザーがワークグループでクエリを実行しましたが、そのワークグループが存在しません。ワークグループが削除された場合、このエラーが発生する可能性があります。別のワークグループに切り替えてクエリを実行してください。
「InvalidRequestException: when calling the StartQueryExecution operation: No output location provided.」(InvalidRequestException: StartQueryExecution オペレーションを呼び出す場合: 出力場所が指定されていません) 「An output location is required either through the Workgroup result configuration setting or as an API input.」(出力場所は、ワークグループ結果構成設定を使用するか、API 入力として指定する必要があります)	ユーザーがクエリ結果の場所を指定せずに API を使用してクエリを実行しました。2 つの方法のいずれかを使用して、クエリ結果の出力場所を設定する必要があります。個別のクエリの場合は「OutputLocation」(クライアント側) を使用するか、ワークグループの場合は「WorkGroupConfiguration」を使用するかのいずれかです。

エラー	表示される場合
「The Create Table As Select query failed because it was submitted with an 'external_location' property to an Athena Workgroup that enforces a centralized output location for all queries.」(Create Table As Select クエリは、すべてのクエリに対して集中管理された出力場所を強制する Athena ワークグループに 'external_location' プロパティを指定して送信されたため、失敗しました) 「Please remove the 'external_location' property and resubmit the query.」('external_location' プロパティを削除して、クエリを再送信してください)	クエリが実行されるワークグループに、強制されたクエリ結果の場所が設定されていて、CTAS クエリに `external_location` を指定しました。この場合、`external_location` を削除し、クエリを再実行します。
「Cannot create prepared statement *`prepared_statement_name`* .」(準備済みステートメントの prepared_statement_name を作成できません) このワークグループ内の準備済みステートメントの数が上限の 1,000 を超えています。	ワークグループに含まれる準備済みステートメントの数が制限の 1,000 を超えています。この問題を回避するには、DEALLOCATE PREPARE を使用してワークグループから 1 つ以上の準備済みステートメントを削除します。または、新しいワークグループを作成します。

CloudWatch のメトリクスとイベントを使用したコストの管理とクエリのモニタリング

ワークグループは、クエリまたはワークグループごとにデータ使用状況の管理制限を設定し、これらの制限を超えたときのアラームをセットアップして、クエリメトリクスを CloudWatch に発行することを可能にします。

各ワークグループに、以下を行えます。

- クエリごとおよびワークグループごとに [Data usage controls (データ使用量の制御)] を設定し、クエリがしきい値を超えた場合に実行されるアクションを設定する。
- クエリメトリクスを表示して分析し、それらを CloudWatch に発行する。コンソールでワークグループを作成すると、メトリクスを CloudWatch に発行するための設定が選択されます。API オペレーションを使用する場合は、メトリクスの発行を有効にする必要があります。メトリクスが発行

されると、それらが [Workgroups] (ワークグループ) パネルの [Metrics] (メトリクス) タブに表示されます。プライマリワークグループでは、メトリクスはデフォルトで無効になっています。

動画

以下のビデオは、CloudWatch でカスタムダッシュボードを作成し、メトリクスにアラームとトリガーを設定する方法を説明しています。Athena コンソールから、事前入力されているダッシュボードを直接使用して、これらのクエリメトリクスを取り込むことができます。

Monitoring Amazon Athena queries using Amazon CloudWatch

トピック

- CloudWatch クエリメトリクスの有効化
- CloudWatch メトリクスによる Athena クエリのモニタリング
- Amazon EventBridge イベントで Athena クエリを監視する
- Athena の使用状況に関するメトリクスのモニタリング
- データ使用量の制御制限の設定

CloudWatch クエリメトリクスの有効化

クエリメトリクスを CloudWatch に発行するための設定は、コンソールでワークグループを作成するときにデフォルトで選択されます。

Athena コンソールでワークグループのクエリメトリクスを有効または無効にする

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. ナビゲーションペインで、[Global networks (グローバルネットワーク)] を選択します。
4. 変更するワークグループのリンクを選択します。
5. ワークグループの詳細ページで、[Edit] (編集) を選択します。
6. [設定] セクションで、[AWS CloudWatch にクエリメトリクスを発行] を選択または解除します。

API オペレーション、コマンドラインインターフェイス、または JDBC ドライバーでクライアントアプリケーションを使用してワークグループを作成する場合にクエリメトリクスの発行を有効にするには、WorkGroupConfiguration で `PublishCloudWatchMetricsEnabled` を `true` に設定します。次の例では、メトリクス設定のみを示し、その他の設定は省略しています。

```
"WorkGroupConfiguration": {
      "PublishCloudWatchMetricsEnabled": "true"
     ....
     }
```

CloudWatch メトリクスによる Athena クエリのモニタリング

Athena は、[publish query metrics to CloudWatch] (CloudWatch にクエリメトリクスを投稿する) オプションが選択されている場合に、クエリ関連のメトリクスを Amazon CloudWatch に発行します。CloudWatch でカスタムダッシュボードを作成し、メトリクスのアラームとトリガーを設定する、または Athena コンソールから事前入力されているダッシュボードを直接使用することができます。

ワークグループでのクエリにクエリメトリクスを有効にすると、そのメトリクスが各ワークグループごとに、Athena コンソールにある [Workgroups] (ワークグループ) パネルの [Metrics] (メトリクス) タブに表示されます。

Athena は、以下のメトリクスを CloudWatch コンソールに発行します。

- `DPUAllocated` — クエリを実行するためにキャパシティ予約にプロビジョニングされた DPU (データ処理ユニット) の総数。
- `DPUConsumed` — 予約の特定の時間に、`RUNNING` 状態のクエリによってアクティブに消費される DPU 数。ワークグループがキャパシティ予約に関連付けられている場合にのみ発行される指標であり、予約に関連付けられているすべてのワークグループが含まれます。
- `DPUCount` — クエリによって消費される DPU の最大数は、クエリの完了時に 1 回だけ発行されます。
- `EngineExecutionTime` — クエリの実行にかかったミリ秒数。
- `ProcessedBytes` — Athena が DML クエリごとにスキャンしたバイト数。
- `QueryPlanningTime` — Athena がクエリ処理フローの計画にかかったミリ秒数。
- `QueryQueueTime` — クエリがリソースを待ちながらクエリキューにいたミリ秒数。
- `ServicePreProcessingTime` — クエリエンジンにクエリを送信する前に Athena がクエリを前処理するためにかかったミリ秒数。
- `ServiceProcessingTime` — クエリエンジンがクエリの実行を終了した後に Athena がクエリ結果を処理するためにかかったミリ秒数。
- `TotalExecutionTime` — Athena が DDL または DML クエリの実行にかかったミリ秒数。

詳細については、このドキュメントで後述される「Athena 向けの CloudWatch メトリクスとディメンションのリスト」を参照してください。

これらのメトリクスには、以下のディメンションがあります。

- `CapacityReservation` — クエリの実行に使用されたキャパシティ予約の名前 (該当する場合)。
- `QueryState` - `SUCCEEDED`、`FAILED`、`CANCELED`
- `QueryType` - `DML`、`DDL`、`UTILITY`
- `WorkGroup` – ワークグループの名前

Athena は、次のメトリクスを `AmazonAthenaForApacheSpark` 名前空間で CloudWatch コンソールに発行します。

- DPUCount - 計算を実行するために、セッション中に消費された DPU の数。

このメトリクスには、次のディメンションがあります。

- SessionId - 計算が送信されるセッションの ID。
- WorkGroup - ワークグループの名前。

詳細については、このトピックで後述する「Athena 向けの CloudWatch メトリクスとディメンションのリスト」を参照してください。Athena の使用状況のメトリクスについては、「Athena の使用状況に関するメトリクスのモニタリング」を参照してください。

ワークグループのクエリメトリクスをコンソールに表示するには

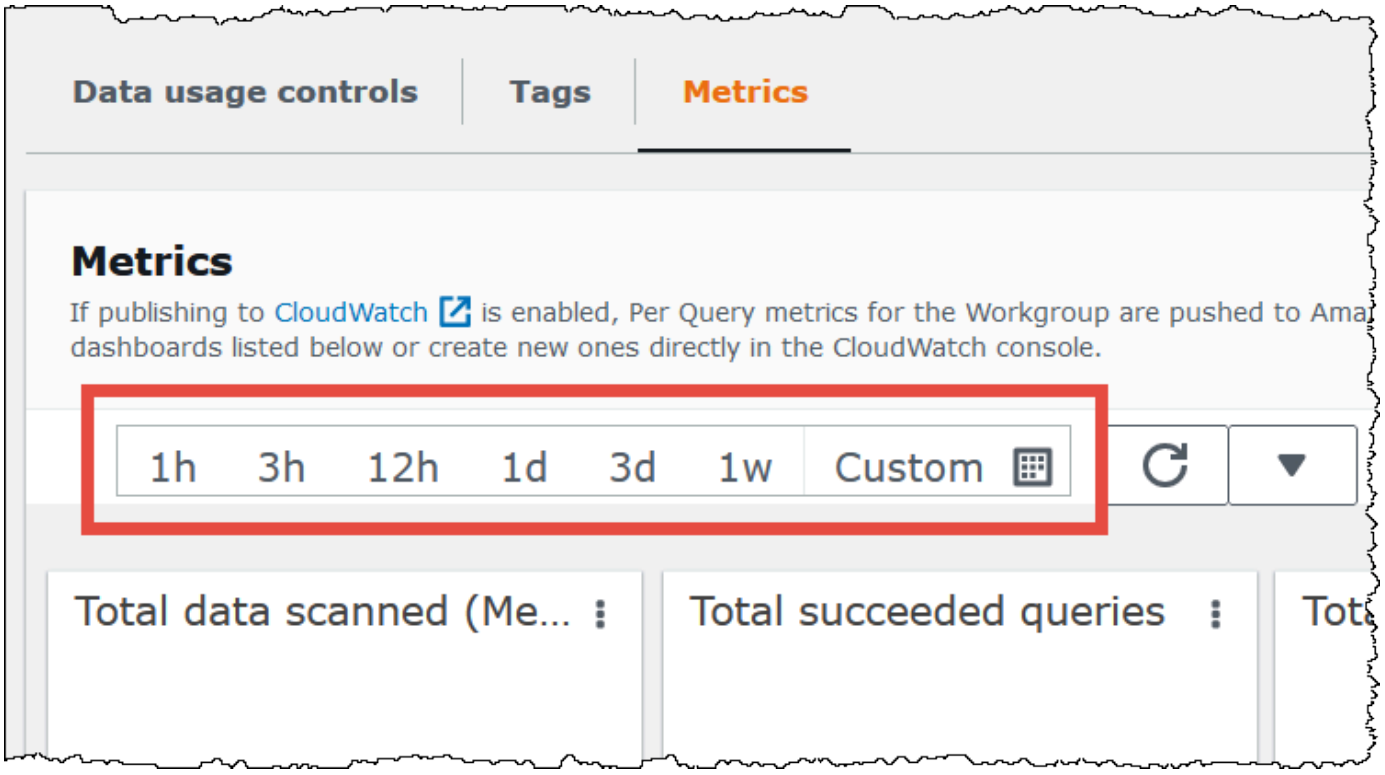
1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. ナビゲーションペインで、[Global networks (グローバルネットワーク)] を選択します。
4. リストから目的のワークグループを選択し、[Metrics] (メトリクス) タブを選択します。

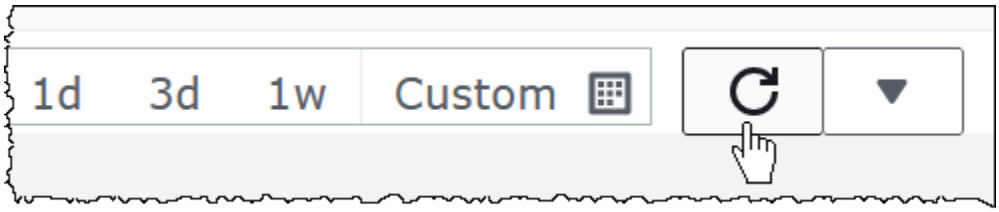
   メトリクスダッシュボードが表示されます。

> **Note**
>
> ワークグループのメトリクスを最近有効にした場合や、最近のクエリアクティビティがない場合は、ダッシュボードのグラフが空になる可能性があります。クエリアクティビティは、次のステップで指定する間隔に応じて CloudWatch から取得されます。

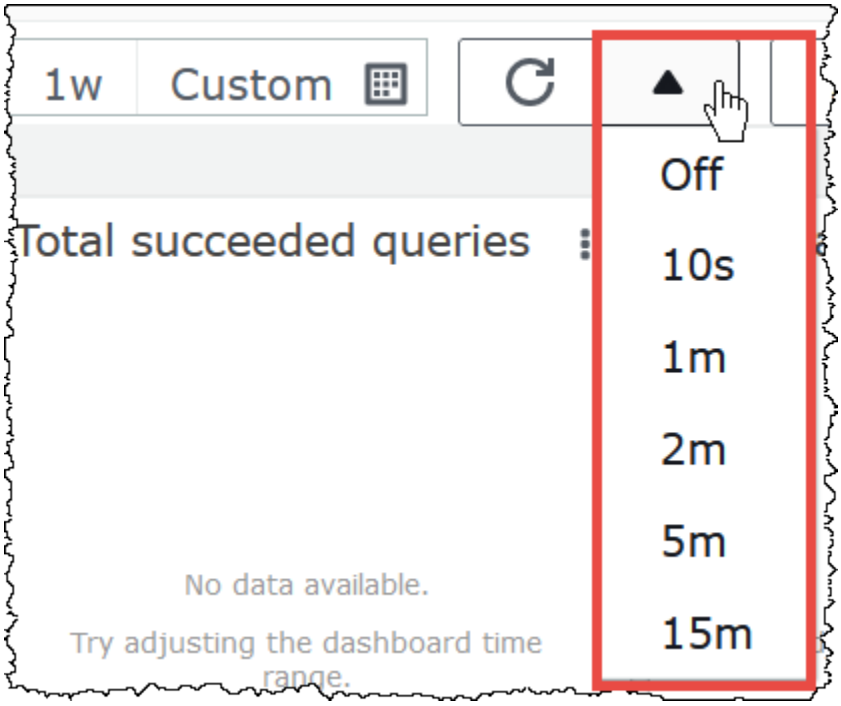
5. [Metrics] (メトリクス) セクションで、CloudWatch からクエリメトリクスをフェッチするために Athena が使用するメトリクス間隔を選択するか、カスタム間隔を指定します。



6. 表示されたメトリクスを更新するには、更新アイコンをクリックします。



7. 更新アイコンの横にある矢印をクリックして、メトリクス表示の更新頻度を選択します。



Amazon CloudWatch コンソールでメトリクスを表示する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. AWS/Athena 名前空間を選択します。

CLI を使用してメトリクスを表示するには

- 次のいずれかを行います。
  - Athena のメトリクスを一覧表示するには、コマンドプロンプトを開き、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace "AWS/Athena"
```

  - すべての使用可能なメトリクスのリストを表示するには、次のコマンドを使用します。

```
aws cloudwatch list-metrics"
```

Athena 向けの CloudWatch メトリクスとディメンションのリスト

Athena で CloudWatch メトリクスを有効にしている場合、Athena がワークグループごとに以下のメトリクスを CloudWatch に送信します。次のメトリクスは `AWS/Athena` 名前空間を使用します。

メトリクス名	説明
DPUAllocated	クエリを実行するためにキャパシティ予約にプロビジョニングされた DPU (データ処理ユニット) の総数。
DPUConsumed	予約の特定の時間に、`RUNNING` 状態のクエリによってアクティブに消費される DPU の数。このメトリックはワークグループがキャパシティ予約に関連付けられている場合にのみ発行され、予約に関連付けられているすべてのワークグループが含まれます。ワークグループをある予約から別の予約に移動した場合、指標にはそのワークグループが最初の予約に属していた時点のデータが含まれます。キャパシティ予約の詳細については、「クエリ処理キャパシティの管理」を参照してください。
DPUCount	クエリによって消費される DPU の最大数は、クエリの完了時に 1 回だけ発行されます。この指標は、キャパシティ予約に関連付けられているワークグループにのみ発行されます。
EngineExecutionTime	クエリの実行にかかった時間のミリ秒数。
ProcessedBytes	Athena が DML クエリごとにスキャンしたバイト数。(ユーザーによって、または制限に達した場合は自動的に) キャンセルされたクエリの場合、キャンセル前にスキャンされたデータの量も含まれます。このメトリクスは、DDL クエリではレポートされません。
QueryPlanningTime	Athena でクエリ処理フローの計画にかかったミリ秒数。これには、データソースからテーブルパーティションを取得するためにかかった時間も含まれます。クエリエンジンがクエリの計画を実行するため、クエリの計画時間は EngineExecutionTime のサブセットであることに注意してください。

メトリクス名	説明
QueryQueueTime	クエリがリソースを待ってキューに並んでいたミリ秒数。一時的なエラーが発生した場合は、クエリを自動的にキューに追加し直すことができます。
ServicePreProcessingTime	クエリエンジンにクエリを送信する前に Athena がクエリを前処理するためにかかったミリ秒数。
ServiceProcessingTime	クエリエンジンがクエリの実行を終了した後に Athena がクエリ結果を処理するためにかかったミリ秒数。
TotalExecutionTime	Athena が DDL クエリまたは DML クエリの実行にかかったミリ秒数。TotalExecutionTime には、QueryQueueTime、QueryPlanningTime、EngineExecutionTime、ServiceProcessingTime が含まれます。

これらの Athena 用メトリクスには、以下のディメンションがあります。

ディメンション	説明
CapacityReservation	クエリの実行に使用されたキャパシティ予約の名前 (該当する場合)。キャパシティ予約が使用されていないとき、このディメンションはデータを返しません。
QueryState	クエリの状態。 有効な統計: SUCCEEDED、FAILED、CANCELED。
QueryType	クエリタイプ。 有効な統計: DDL、DML または UTILITY。実行されたクエリステートメントのタイプ。DDL は DDL (データ定義言語) クエリステートメントを示します。DML は、CREATE TABLE AS SELECT などの DML (データ操作言語) クエリステートメントを示します。UTILITYは、SHOW CREATE TABLE、または DESCRIBE TABLE などの DDL と DML 以外のクエリステートメントを示します。

ディメンション	説明
WorkGroup	ワークグループの名前。

Amazon EventBridge イベントで Athena クエリを監視する

Amazon Athena を Amazon EventBridge と使用し、クエリの状態に関するリアルタイムの通知を受け取ることができます。送信したクエリの状態が推移すると、Athena がそのクエリ状態の推移に関する情報が含まれたイベントを EventBridge に発行します。関心のあるイベントのシンプルなルールを作成し、イベントがルールに一致したときに自動化されたアクションが実行されるようにできます。たとえば、クエリが最終状態に達したときに AWS Lambda 関数を呼び出すルールを作成できます。イベントは、ベストエフォートベースで出力されます。

Athena のイベントルールを作成する前に、以下を実行する必要があります。

- EventBridge のイベント、ルール、ターゲットに精通しておいてください。詳細については、「Amazon EventBridge とは?」を参照してください。ルールの設定方法に関する詳細については、「Amazon EventBridge の開始方法」を参照してください。
- イベントのルールで使用するターゲットを作成する。

> **Note**
>
> Athena では、現在 Athena Query State Change という 1 つのイベントタイプを提供していますが、他のタイプのイベントと詳細が追加する場合があります。イベントの JSON データをプログラムで逆シリアル化する場合は、不明なプロパティが追加されているときにアプリケーションで対応する準備ができていることを確認してください。

Athena イベントの形式

以下は、Amazon Athena イベントの基本的なパターンです。

```
{
    "source":[
        "aws.athena"
    ],
    "detail-type":[
        "Athena Query State Change"
```

```
    ],
    "detail":{
        "currentState":[
            "SUCCEEDED"
        ]
    }
}
```

Athena Query State Change イベント

次の例は、currentState の値が SUCCEEDED の Athena Query State Change イベントを示しています。

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Athena Query State Change",
    "source":"aws.athena",
    "account":"123456789012",
    "time":"2019-10-06T09:30:10Z",
    "region":"us-east-1",
    "resources":[

    ],
    "detail":{
        "versionId":"0",
        "currentState":"SUCCEEDED",
        "previousState":"RUNNING",
        "statementType":"DDL",
        "queryExecutionId":"01234567-0123-0123-0123-012345678901",
        "workgroupName":"primary",
        "sequenceNumber":"3"
    }
}
```

次の例は、currentState の値が FAILED の Athena Query State Change イベントを示しています。athenaError ブロックは、currentState が FAILED の場合のみ表示されます。errorCategory、および errorType の値についての詳細は、「Athena エラーカタログ」を参照してください。

```
{
    "version":"0",
```

```
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Athena Query State Change",
    "source":"aws.athena",
    "account":"123456789012",
    "time":"2019-10-06T09:30:10Z",
    "region":"us-east-1",
    "resources":[
    ],
    "detail":{
        "athenaError": {
            "errorCategory": 2.0, //Value depends on nature of exception
            "errorType": 1306.0, //Type depends on nature of exception
            "errorMessage": "Amazon S3 bucket not found", //Message depends on nature
 of exception
            "retryable":false //Retryable value depends on nature of exception
        },
        "versionId":"0",
        "currentState": "FAILED",
        "previousState": "RUNNING",
        "statementType":"DML",
        "queryExecutionId":"01234567-0123-0123-0123-012345678901",
        "workgroupName":"primary",
        "sequenceNumber":"3"
    }
}
```

出力プロパティ

JSON 出力には、以下のプロパティが含まれます。

プロパティ	説明
athenaError	currentState が FAILED の場合にのみ表示されます。エラーカテゴリ、エラータイプ、エラーメッセージ、およびエラーの原因となったアクションを再試行できるかどうかなど、発生したエラーに関する情報が含まれます。これらのフィールドそれぞれの値は、エラーの性質によって異なります。errorCategory 、および errorType の値についての詳細は、「Athena エラーカタログ」を参照してください。
versionId	詳細オブジェクトのスキーマのバージョン番号。

プロパティ	説明
currentState	イベント発生時のクエリの移行後の状態。
previousState	イベント発生時のクエリの移行前の状態。
statementType	実行されたクエリステートメントのタイプ。
queryExecutionId	実行されたクエリの一意の識別子。
workgroupName	クエリが実行されたワークグループの名前。
sequenceNumber	実行された単一のクエリが関与する受信イベントの重複排除と順序付けを可能にする単調増加数。同じ状態移行に対して重複するイベントが発行された場合、sequenceNumber 値は同じです。再キューイングがまれに発生するクエリなど、状態移行が複数回発生するクエリの場合は、sequenceNumber を使用して、currentState および previousState 値が同じイベントを順序付けできます。

例

以下の例は、サブスクライブした Amazon SNS トピックにイベントを発行します。Athena がクエリされると、E メールを受信します。この例は、Amazon SNS トピックが存在し、そのトピックにサブスクライブしていることを前提としています。

Athena イベントを Amazon SNS トピックに発行する

1. Amazon SNS トピックのターゲットを作成します。次の例のように、EventBridge イベントが Amazon SNS トピックに発行するため、サービスプリンシパルの events.amazonaws.com 許可を付与します。

```
{
    "Effect":"Allow",
    "Principal":{
        "Service":"events.amazonaws.com"
    },
    "Action":"sns:Publish",
    "Resource":"arn:aws:sns:us-east-1:111111111111:your-sns-topic"
```

```
}
```

2. 以下の例にあるように、AWS CLI `events put-rule` コマンドを使用して Athena イベントのルールを作成します。

```
aws events put-rule --name {ruleName} --event-pattern '{"source": ["aws.athena"]}'
```

3. 以下の例にあるように、AWS CLI `events put-targets` コマンドを使用して Amazon SNS トピックのターゲットをルールにアタッチします。

```
aws events put-targets --rule {ruleName} --targets Id=1,Arn=arn:aws:sns:us-
east-1:111111111111:your-sns-topic
```

4. Athena をクエリして、呼び出されるターゲットを確認します。Amazon SNS トピックから対応する E メールを受信します。

Amazon Athena での AWS User Notifications の使用

AWS User Notifications を使用して配信チャネルを設定して、Amazon Athena イベントに関する通知を受け取ることができます。指定したルールにイベントが一致すると、通知を受け取ります。イベントの通知は、E メール、AWS Chatbot チャット通知、AWS Console Mobile Application プッシュ通知など、複数のチャネルを通じて受け取ることができます。また、コンソール通知センターの通知を確認することもできます。User Notifications は集約をサポートしているため、特定のイベント中に受け取る通知の数を減らすことができます。

詳細については、AWS User Notifications ユーザーガイドを参照してください。

Athena の使用状況に関するメトリクスのモニタリング

CloudWatch の使用状況メトリクスにより、CloudWatch グラフやダッシュボード上に現在のサービスの使用状況を表示することで、アカウントがどのようにリソースを使用しているかを可視化できます。

Athena での使用可能性に関するメトリクスは、Athena の AWS のサービス クォータと対応しています。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。Athena でのサービスクォータの詳細については、「Service Quotas」を参照してください。AWS の使用状況メトリクスの詳細については、「Amazon CloudWatch ユーザーガイド」の「AWS 使用状況メトリクス」を参照してください。

Athena は、以下のメトリクスを `AWS/Usage` 名前空間に公開します。

メトリクス名	説明
ResourceCount	クエリタイプ (DML または DDL) で分けられた、各アカウントの AWS リージョンごとのキュー登録されたクエリと実行中クエリすべての合計数。このメトリクスでは、最大数のみが有用な統計情報です。 このメトリクスは、1 分ごとに定期的に発行されます。実行するクエリがない場合、このメトリクスではレポートを何も (0 すらも) 返しません。このメトリクスは、それ自体が取得された時点でアクティブなクエリが実行されている場合にのみ公開されます。

以下のディメンションは、Athena によって発行された使用状況メトリクスを絞り込むために使用されます。

ディメンション	説明
Service	リソースを含む AWS のサービス の名前。Athena では、このディメンションの値は Athena になります。
Resource	実行中のリソースタイプ。Athena クエリの使用状況に関するリソース値は ActiveQueryCount です。
Type	報告されるエンティティのタイプ。現在、Athena 使用状況メトリクスで有効な値は Resource のみです。
Class	追跡されているリソースのクラス。Athena の場合、Class は DML または DDL となります。

CloudWatch コンソールでの Athena リソースの使用状況メトリクスの表示

CloudWatch コンソールでは、Athena の使用状況メトリクスのグラフを表示したり、使用量がサービスクォータに近づいた場合に警告するためのアラームを設定したりできます。

Athena リソースの使用状況メトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [使用]を選択し、[AWS リソース別]を選択します。

   サービスクォータ使用状況メトリクスのリストが表示されます。
4. [Athena] と [ActiveQueryCount] の横にあるチェックボックスをオンにします。
5. [グラフ化したメトリクス] タブを選択します。

   その上のグラフに、その AWS リソースの現在の使用状況が表示されます。

サービスクォータをグラフに追加し、サービスクォータに近づいた場合に通知を行うアラームを設定する方法については、「Amazon CloudWatch ユーザーガイド」の「Visualizing your service quotas and setting alarms」(サービスクォータの可視化とアラームの設定) を参照してください。ワークグループごとの使用制限の設定については、「データ使用量の制御制限の設定」を参照してください。

データ使用量の制御制限の設定

Athena では、クエリごとの制限とワークグループごとの制限の 2 つのタイプのコスト制御を設定できます。ワークグループごとに、クエリごとの制限を 1 つとワークグループごとの制限を複数設定できます。

- クエリごとの制御制限は、クエリごとにスキャンされるデータの合計量を指定します。ワークグループで実行されるクエリが制限を超えると、キャンセルされます。ワークグループに作成できるクエリごとの制御制限は 1 つだけであり、その中で実行される各クエリに適用されます。変更する必要がある場合は、制限を編集します。詳細なステップについては、「クエリごとにデータ使用量の制御を作成するには」を参照してください。
- ワークグループ全体のデータ使用量の制御制限は、指定された期間中にこのワークグループで実行されるすべてのクエリに対してスキャンされるデータの合計量を指定します。ワークグループごとに複数の制限を作成できます。ワークグループ全体のクエリ制限を使用すると、ワークグループで実行されているクエリによってスキャンされたデータの時間単位または日単位の集計に複数のしきい値を設定できます。

  スキャンされたデータの合計量がしきい値を超えた場合に、Amazon SNS トピックに通知をプッシュできます。これを実行するには、制限を超えたときに管理者が通知を受けるように、Amazon SNS アラームと Athena コンソールでのアクションを設定します。詳細なステップについては、「ワークグループごとにデータ使用量の制御を作成するには」を参照してください。Athena が

CloudWatch コンソールから発行する任意のメトリクスに対してアラームとアクションを作成することもできます。たとえば、失敗した多くのクエリに対してアラートを設定できます。このアラートは、数が特定のしきい値を超えた場合に管理者への E メールをトリガーできます。制限を超えた場合、アクションが指定されたユーザーに Amazon SNS アラーム通知を送信します。

実行できるその他のアクションは、以下のとおりです。

- Lambda 関数を呼び出す。詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS 通知を使用した Lambda 関数の呼び出し」を参照してください。
- ワークグループを無効にして、さらにクエリが実行されないようにする。この手順については、「ワークグループの有効化および無効化」を参照してください。

クエリごと、ワークグループごとの制限は相互に独立しています。指定したどちらかの制限を超えた場合にアクションが実行されます。複数のユーザーが同じワークグループで同時にクエリを実行した場合、各クエリは指定された制限を超えないものの、スキャンされたデータの合計がワークグループごとのデータ使用制限を超える可能性があります。この場合、Amazon SNS アラームがユーザーに送信されます。

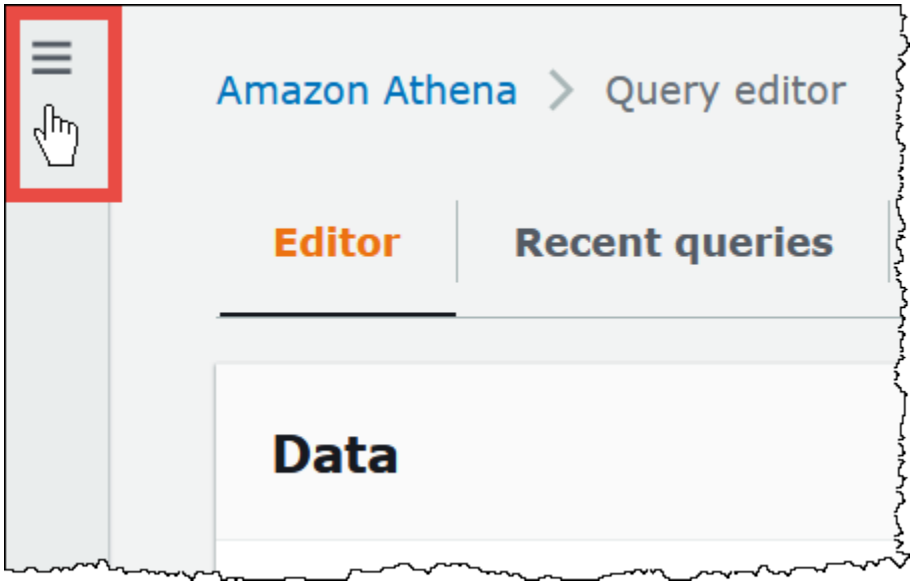
クエリごとのデータ使用量の制御を作成するには

クエリごとの制御制限は、クエリごとにスキャンされるデータの合計量を指定します。ワークグループで実行されるクエリが制限を超えると、キャンセルされます。キャンセルされたクエリの料金は、Amazon Athena の料金表に従って請求されます。

> **Note**
>
> キャンセルされた、または失敗したクエリの場合、Athena が既に部分的な結果を Amazon S3 に書き込んでいる可能性があります。このような場合、Athena は結果が保存されている Amazon S3 プレフィックスから部分的な結果を削除しません。部分的な結果がある Amazon S3 プレフィックスを削除する必要があります。Athena は、Amazon S3 マルチパートアップロードを使用して Amazon S3 データを書き込みます。クエリが失敗した場合にはマルチパートアップロードを中止するように、バケットライフサイクルポリシーを設定することが推奨されます。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「マルチパートアップロードを使用したオブジェクトのアップロードとコピー」を参照してください。

ワークグループに作成できるクエリごとの制御制限は 1 つだけであり、その中で実行される各クエリに適用されます。変更する必要がある場合は、制限を編集します。

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。



3. ナビゲーションペインで、[Global networks] (グローバルネットワーク) を選択します。
4. リストからワークグループ名を選択します。
5. [Data usage controls] (データ使用量の制御) タブの、[Per query data usage control] (クエリごとのデータ使用量の制御) セクションで、[Manage] (管理) を選択します。
6. [Manage per query data usage control] (クエリごとのデータ使用量の制御を管理) ページで、次の値を指定します。

   - [Data limit] (データの制限) で、10 MB (最小) から 7 EB (最大) の値を指定します。

     > **Note**
     >
     > これらはワークグループのデータ使用量の制御のためにコンソールによって設定される制限です。これは、Athena でのクエリ制限を表すものではありません。

   - 単位については、ドロップダウンリストから単位の値 (Kilobytes KB または Exabytes EB など) を選択します。

デフォルトのアクションでは、制限を超えた場合はクエリをキャンセルします。この設定は変更できません。

7. [Save] を選択します。

ワークグループごとのデータ使用量アラートを作成または編集するには

ワークグループで実行されているクエリが特定の期間内に指定された量のデータをスキャンする際、複数のアラートしきい値を設定できます。アラートは Amazon CloudWatch アラームを使用して実装され、ワークグループ内のすべてのクエリに適用されます。しきい値に達すると、Amazon SNS から指定したユーザーに E メールを送信できます。しきい値に達した場合でも、クエリは自動的にキャンセルされません。

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. ナビゲーションペインで、[Global networks] (グローバルネットワーク) を選択します。
4. リストからワークグループ名を選択します。
5. [Edit] (編集) を選択して、ワークグループの設定を編集します。
6. 下にある [Workgroup data usage alerts - optional] (ワークグループのデータ使用量アラート - オプション) までスクロールし、展開します。
7. [Add alert] (アラートの追加) を選択します。
8. [Data usage threshold configuration] (データ使用量のしきい値の設定) については、次のように値を指定します。

   - [Data threshold] (データのしきい値) については、数値を指定し、ドロップダウンリストから単位の値を選択します。
   - [Time period] (期間) については、ドロップダウンリストから期間を選択します。
   - [SNS topic selection] (SNS トピックの選択) については、ドロップダウンリストから Amazon SNS トピックを選択します。または、[Create SNS topic] (SNS トピックの作成) を選択して Amazon SNS コンソールに直接移動し、Amazon SNS トピックを作成して、Athena アカウントのユーザーの 1 人にサブスクリプションをセットアップします。詳細については、[Amazon Simple 通知サービスデベロッパーガイド] の [Amazon SNS の使用開始] を参照してください。

9. 新しいアラートを作成する場合は [Add alert] (アラートを追加) を選択し、既存のアラートを保存する場合は [Save] (保存) を選択します。

クエリ処理キャパシティの管理

キャパシティ予約を使用して、Athena で実行するクエリ専用の処理キャパシティを確保することができます。キャパシティ予約を使用することで、ワークロード管理機能で重要な相互作用ワークロードの優先順位付け、制御、スケーリングを便利に行えます。たとえば、キャパシティを追加して同時に実行できるクエリの数を増やしたり、そのキャパシティを使用できるワークロードを制御したり、ワークロード間でキャパシティを共有したりすることがいつでも可能です。キャパシティは Athena によって完全に管理され、必要な限り保持されます。セットアップは簡単で、SQL ステートメントを変更する必要はありません。

クエリの処理キャパシティを確保するには、キャパシティ予約を作成し、必要なデータ処理ユニット (DPU) の数を指定し、予約に 1 つ以上のワークグループを割り当てます。

キャパシティ予約を使用する際に、ワークグループは重要な役割を果たします。ワークグループによって、クエリを論理的グルーピングに整理できます。キャパシティ予約によって、キャパシティをワークグループに選択的に割り当て、各ワークグループのクエリの動作と請求方法を制御できます。ワークグループの詳細については、「ワークグループを使用してクエリのアクセスとコストを制御する」を参照してください。

ワークグループを予約に割り当てると、割り当てられたワークグループに送信するクエリに優先順位を付けることができます。たとえば、時間的制約のある財務報告クエリに使用されるワークグループにキャパシティを割り当てて、それらのクエリを別のワークグループ内の重要度の低いクエリから隔離することができます。これにより、重要なワークロードに対して安定したクエリ実行が可能になり、他のワークロードは独立して実行できます。

キャパシティ予約とワークグループを組み合わせて使用すると、さまざまな要件に対応することができます。例としていくつかのシナリオを以下に挙げます。

- 隔離 — 重要なワークロードを隔離するには、1 つの予約に 1 つのワークグループを割り当てます。割り当てられたワークグループのクエリのみが、選択した予約の処理能力を使用します。
- 共有 — 複数のワークロードは 1 つの予約からキャパシティを共有できます。たとえば、特定の複数ワークロードの月次コストを予測できるようにするには、1 つの予約に複数のワークグループを割り当てます。割り当てられたワークグループがその予約のキャパシティを共有します。
- 混合モデル — キャパシティ予約およびクエリ単位の請求を同じアカウントで同時に使用できます。たとえば、運用アプリケーションをサポートするクエリを確実に実行するには、それらのクエ

リのワークグループをキャパシティ予約に割り当てます。クエリを運用ワークグループに移動する前に開発するとき、予約に関連付けされていない別のワークグループを使用するため、クエリ単位の課金を使用します。

DPU について

キャパシティはデータ処理ユニット (DPU) で測定されます。DPU は、Athena がユーザーに代わってデータにアクセスして処理するために使用するコンピューティングリソースとメモリリソースです。1 つの DPU では、4 個の vCPU と 16 GB のメモリが使用できます。指定する DPU の数によって、同時に実行できるクエリ数が決まります。例えば、DPU 256 個の予約では、DPU 128 個の予約の約 2 倍の同時クエリ数に対応できます。

アカウントとリージョンごとに、最大合計 1,000 個の DPU で最大 100 個のキャパシティ予約を作成できます。リクエストできる DPU の最小数は 24 個です。ユースケースに 1,000 個を超える DPU が必要な場合は、athena-feedback@amazon.com までご連絡ください。

キャパシティ要件の見積もりについては、「キャパシティ要件の特定」を参照してください。料金については、「Amazon Athena の料金」を参照してください。

考慮事項と制約事項

- この機能には、Athena エンジンバージョン 3 が必要です。
- 1 つのワークグループは一度に 1 つの予約に割り当てることができ、1 つの予約には最大 20 のワークグループを追加できます。
- Spark 対応のワークグループをキャパシティ予約に追加することはできません。
- 予約に割り当てられているワークグループを削除するには、まず予約からワークグループを削除します。
- プロビジョニングする DPU の最小数は 24 個です。
- アカウントとリージョンごとに、最大合計 1,000 個の DPU で最大 100 個のキャパシティ予約を作成できます。
- キャパシティのリクエストは保証されないため、完了するまでに最大 30 分かかる場合があります。
- 1 件の予約あたりの最低請求期間は 1 時間です。1 時間が経過すると、キャパシティは 1 分単位で請求されます。料金については、「Amazon Athena の料金」を参照してください。
- リザーブドキャパシティを別のキャパシティ予約、AWS アカウント、AWS リージョン に転送することはできません。

- キャパシティ予約の DDL クエリは DPU を消費します。
- プロビジョニングされたキャパシティで実行されるクエリは、DDL と DML のアクティブなクエリ制限にはカウントされません。
- すべての DPU が使用中の場合、送信されたクエリはキューに入れられます。このようなクエリは拒否されず、オンデマンドキャパシティにも送られません。
- DPUConsumed CloudWatch メトリクスは予約ごとではなく、ワークグループごとに対応します。そのため、ワークグループをある予約から別の予約に移動すると、そのワークグループが最初の予約に属していたときのデータが DPUConsumed メトリクスに含まれます。Athena の CloudWatch メトリクスの使用に関する詳細については、「CloudWatch メトリクスによる Athena クエリのモニタリング」を参照してください。
- 現在、この機能は以下の AWS リージョン で利用できます。
  - 米国東部 (オハイオ)
  - 米国東部 (バージニア北部)
  - 米国西部 (オレゴン)
  - 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)
  - アジアパシフィック (東京)
  - 欧州 (アイルランド)
  - 欧州 (ストックホルム)

トピック

キャパシティ要件の特定

キャパシティ予約を作成する前に、必要なキャパシティを見積もることで、正しい数の DPU を割り当てることができます。また、予約の使用後は、予約のキャパシティが不足していないか、超過していないかを確認するとよいでしょう。このトピックでは、これらの見積もりに使用できる手法について説明し、使用状況とコストを評価するためのいくつかの AWS ツールについても説明します。

トピック

必要なキャパシティの見積もり

キャパシティ要件を見積もるときは、特定のクエリに必要なキャパシティと、一般的に必要なキャパシティという 2 つの観点を考慮すると効果的です。

クエリごとのキャパシティ要件の見積もり

クエリで必要な DPU の数を判断するには、次のガイドラインを使用します。

- DDL クエリは 4 個の DPU を消費します。
- DML クエリは 4〜124 個の DPU を消費します。

Athena は、クエリが送信されるときに DML クエリに必要な DPU の数を決定します。この数は、データサイズ、ストレージ形式、クエリ構造、およびその他の要因によって変わります。通常、Athena は最も少なく、最も効率的な DPU 数を選択しようとします。Athena は、クエリを正常に完了するためにより多くの計算能力が必要であると判断した場合、クエリに割り当てられる DPU の数を増やします。

ワークロード固有のキャパシティ要件の見積もり

複数のクエリを同時に実行するために必要な容量を判断するには、次の表にある一般ガイドラインを考慮してください。

同時クエリ	必要な DPU
10	40 またはそれ以上
20	96 以上
30 またはそれ以上	240 以上

実際に必要な DPU の数は、目標と分析パターンによって異なることに注意してください。たとえば、クエリをキューに入れずにすぐに開始したい場合は、ピーク時の同時クエリ需要を判断し、それに応じて DPU の数をプロビジョニングします。

プロビジョニングする DPU の数は、ピーク需要よりも少なくても構いませんが、ピーク需要が発生するとキューイングが発生する可能性があります。キューイングが発生すると、Athena はクエリをキューに保持し、キャパシティが利用可能になるとクエリを実行します。

固定予算内でクエリを実行することが目標であれば、「AWS 料金計算ツール」を使って予算に合う DPU 数を特定できます。

最後に、データサイズ、ストレージ形式、およびクエリの記述方法が、クエリに必要な DPU に影響することを覚えておいてください。データを圧縮または分割したり、列形式に変換したりすることで、クエリのパフォーマンスを向上させることができます。詳細については、「Athena でのパフォーマンスのチューニング」を参照してください。

より多くのキャパシティが必要であることを示す兆候

割り当てられたキャパシティが不十分であることの 2 つの兆候として、キャパシティ不足のエラーメッセージとクエリキューイングがあります。

キャパシティ不足のエラーメッセージが表示されてクエリが失敗した場合、クエリに対してキャパシティ予約の DPU 数が少なすぎる可能性があります。たとえば、DPU 24 個の予約があり、24 個よりも多い DPU を必要とするクエリを実行すると、クエリは失敗します。Athena の EventBridge イベントを使用してこのクエリエラーを監視することができます。DPU をさらに追加してクエリを再実行してみてください。

多くのクエリがキューに入っている場合は、キャパシティがすべて他のクエリに使用されていることを意味します。キューを減らすには、次のいずれかの操作を行います。

- 予約に DPU を追加して、クエリの同時実行性を向上させます。
- 予約からワークグループを削除して、他のクエリに使えるキャパシティを増やします。

キャパシティ予約内のワークグループ内で Athena クエリキュー時間の CloudWatch メトリックを使用して、クエリキューイングが過剰になっていないかどうかを確認します。値が希望のしきい値を超える場合は、キャパシティ予約に DPU を追加します。

アイドル状態のキャパシティの確認

アイドル状態のキャパシティを確認するには、予約内の DPU の数を減らすか、ワークロードを増やしてから結果を確認します。

アイドル状態のキャパシティを確認するには

1. 次のいずれかを行います。
   - 予約内の DPU の数を減らします (利用可能なリソースを減らす)。
   - 予約にワークグループを追加します (ワークロードを増やす)。
2. CloudWatch を使用してクエリキュー時間を測定します。
3. キュー時間が望ましいレベルを超えて増加した場合は、次のいずれかの操作を行います。
   - ワークグループを削除します。
   - キャパシティ予約に DPU を追加します。
4. 変更のたびに、パフォーマンスとクエリキュー時間を確認します。
5. 引き続きワークロードや DPU 数を調整して、望ましいバランスをとってください。

望ましい時間を超えるキャパシティを維持したくない場合は、予約をキャンセルして後で別の予約を作成します。ただし、最近別の予約からキャパシティをキャンセルしていたとしても、新しいキャパシティのリクエストが保証されるわけではなく、新しい予約の作成には時間がかかります。

キャパシティ要件とコストを評価するためのツール

AWS の以下のサービスと機能を使用して、Athena の使用状況とコストを測定できます。

CloudWatch メトリクス

クエリ関連のメトリックスをワークグループレベルで Amazon CloudWatch に公開するように Athena を設定できます。ワークグループのメトリックスを有効にすると、ワークグループのクエリに関するメトリックスがワークグループの詳細ページの Athena コンソールに表示されます。

CloudWatch に公開される Athena メトリックスとそのディメンションについては、「CloudWatch メトリクスによる Athena クエリのモニタリング」を参照してください。

CloudWatch の使用状況メトリクス

CloudWatch の使用状況メトリクスにより、CloudWatch グラフやダッシュボード上に現在のサービスの使用状況を表示することで、アカウントがどのようにリソースを使用しているかを可視化できま

す。Athena での使用可能性に関するメトリクスは、Athena の AWSService Quotas と対応しています。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。

詳細については、「Athena の使用状況に関するメトリクスのモニタリング」を参照してください。

Amazon EventBridge イベント

Amazon Athena を Amazon EventBridge と使用し、クエリの状態に関するリアルタイムの通知を受け取ることができます。送信したクエリの状態が変更されると、Athena がそのクエリの状態推移に関する情報が含まれるイベントを EventBridge に発行します。関心のあるイベントのシンプルなルールを作成し、イベントがルールに一致したときに自動化されたアクションが実行されるようにできます。

詳細については、以下のリソースを参照してください。

- Amazon EventBridge イベントで Athena クエリを監視する
- Amazon EventBridge とは
- Amazon EventBridge イベント

タグ

Athena では、キャパシティ予約がタグをサポートしています。タグは、キーと値から構成されています。Athena でコストを追跡するには、AWS 生成のコスト配分タグを使用できます。AWS はコスト配分タグを使用し、[コストと使用状況のレポート] でリソースのコストを整理します。これにより、AWS コストを簡単に分類して追跡できるようになります。Athena のコスト配分タグを有効にするには、AWS Billing and Cost Management コンソールを使用します。

詳細については、以下のリソースを参照してください。

- Athena リソースへのタグ付け
- AWS 生成のコスト配分タグをアクティブ化する
- AWS コスト配分タグを使用する

キャパシティ予約の作成

まず、必要な数の DPU を含むキャパシティ予約を作成し、そのキャパシティをクエリに使用する 1 つ以上のワークグループを割り当てます。安定したパフォーマンスの提供や、コスト管理改善のために、必要に応じて後からキャパシティを調整できます。キャパシティ要件の見積もりについては、「キャパシティ要件の特定」を参照してください。

> **Important**
>
> キャパシティのリクエストは保証されないため、完了するまでに最大 30 分かかる場合があります。

キャパシティ予約を作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. [管理]、[キャパシティ予約] を選択します。
4. [キャパシティ予約を作成] を選択します。
5. [キャパシティ予約を作成] ページの [キャパシティ予約名] に名称を入力します。名称は一意でなくてはならず、1～128 文字で、a～z、A～Z、0～9、_ (アンダースコア)、. (ピリオド)、- (ハイフン)が使用できます。予約作成後に名称を変更することはできません。
6. DPU では、必要なデータ処理ユニット (DPU) の数を 4 個単位で選択または入力します。詳細については、「DPU について」を参照してください。
7. (オプション) [タグ] オプションを展開し、[新しいタグを追加] を選択し、キャパシティ予約リソースに関連付ける 1 つ以上のカスタムキー/値ペアを追加します。詳細については、「Athena リソースへのタグ付け」を参照してください。
8. [Review] (レビュー) を選択します。
9. [キャパシティ予約の作成を確認] プロンプトで、DPU の数、AWS リージョン、その他の情報を確認します。 q 問題がなければ、[送信] を選択します。

   詳細ページでは、キャパシティ予約の [ステータス] は [保留中] と表示されます。予約キャパシティでクエリを実行できる場合、そのステータスは [アクティブ] と表示されます。

この時点で、1 つ以上のワークグループを予約に追加する準備が整っています。この手順については、「予約へのワークグループの追加」を参照してください。

予約の管理

[キャパシティ予約] ページでは、キャパシティ予約を表示および管理できます。DPU の追加や削減、ワークグループ割り当ての変更、予約のタグ付けやキャンセルなどの管理タスクを実行できます。

キャパシティ予約の表示および管理

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. [管理]、[キャパシティ予約] を選択します。
4. [キャパシティ予約] ページでは、次のタスクを実行できます。
   - キャパシティ予約を作成するには、[キャパシティ予約を作成] を選択します。
   - 検索ボックスを使用して、予約を名前または DPU の数でフィルタリングします。
   - ステータスドロップダウンメニューを選択して、キャパシティ予約ステータス ([アクティブ]、[キャンセル済み] など) でフィルタリングします。予約ステータスについては、「予約ステータスについて」を参照してください。
   - キャパシティ予約の詳細を表示するには、予約のリンクを選択します。予約の詳細ページには、[キャパシティの編集]、[ワークグループの追加]、[ワークグループの削除]、[予約のキャンセル] のオプションがあります。
   - 予約を編集 (DPU を追加または削除するなど) するには、その予約のボタンを選択し、[編集] を選択します。
   - 予約をキャンセルするには、その予約のボタンを選択し、[キャンセル] を選択します。

予約ステータスについて

キャパシティ予約の取りうるステータス値は、次の表のとおりです。

ステータス	説明
[保留中]	Athena はキャパシティリクエストを処理中です。キャパシティはクエリを実行する準備ができていません。
[アクティブ]	クエリを実行するためのキャパシティがあります。
[失敗]	キャパシティのリクエストは正常に完了しませんでした。キャパシティの充足は保証されないことに注意してください。予約に失敗した場合、アカウントの DPU 制限にカウントされます。利用状況を解放するには、予約をキャンセルする必要があります。

ステータス	説明
[更新保留中]	Athena は予約の変更を処理中です。たとえば、予約を編集して DPU を追加または削除すると、このステータスが表示されます。
[キャンセル中]	Athena は予約のキャンセルリクエストを処理中です。予約を使用していたワークグループでまだ実行中のクエリは完了できますが、ワークグループ内の他のクエリはオンデマンド (プロビジョニングされていない) キャパシティを使用します。
キャンセル済み	キャパシティ予約のキャンセルが完了しました。キャンセルされた予約は、45 日間コンソールに残ります。45 日後、Athena は予約を削除します。この 45 日間は、その予約の編集や再使用はできませんが、過去の参照用に予約のタグを参照したり、詳細を確認したりすることはできます。 キャンセルされたキャパシティが後日再予約可能になる保証はありません。キャパシティを別の予約、AWS アカウント、AWS リージョン に移すことはできません。

アクティブ DPU とターゲット DPU について

Athena コンソールのキャパシティ予約のリストで、予約には 2 つの DPU 値 ([アクティブ DPU] と [ターゲット DPU]) が表示されます。

- [アクティブ DPU] — 予約内でクエリを実行できる DPU の数。たとえば、DPU 100 個をリクエストし、そのリクエストが受理された場合、[アクティブ DPU] には 100 と表示されます。
- [ターゲット DPU] — 予約の移行先の DPU の数。予約の作成中、または DPU 数の増減が保留中の場合、[ターゲット DPU] には [アクティブ DPU] とは異なる値が表示されます。

たとえば、DPU 24 個の予約を作成するリクエストを送信すると、[ステータス] は [保留中] になり、[アクティブ DPU] は 0、[ターゲット DPU] は 24 になります。

DPU 100 個の予約があり、予約を編集して DPU 20 個の増加をリクエストした場合、[ステータス] は[更新保留中] になり、[アクティブ DPU] は 100、[ターゲット DPU] は 120 になります。

DPU 100 個の予約があり、予約を編集して DPU 20 個の削減をリクエストした場合、[ステータス] は[更新保留中] になり、[アクティブ DPU] は 100、[ターゲット DPU] は 80 になります。

これらの移行中、Athena はリクエストに応じて DPU の取得または削減を能動的に行っています。[アクティブ DPU] が [ターゲット DPU] と等しい時は、目標数に達しており、保留中の変更はありません。

これらの値をプログラムで取得するには、GetCapacityReservation API アクションを呼び出します。API では、[アクティブ DPU] および [ターゲット DPU] をそれぞれ、`AllocatedDpus` および `TargetDpus` と呼びます。

トピック

- キャパシティ予約を編集する
- 予約へのワークグループの追加
- 予約からワークグループを削除する
- キャパシティ予約をキャンセルする
- キャパシティ予約の削除

キャパシティ予約を編集する

キャパシティ予約を作成したら、DPU の数を調整したり、カスタムタグを追加または削除したりできます。

キャパシティ予約を編集するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. [管理]、[キャパシティ予約] を選択します。
4. キャパシティ予約のリストで、次のいずれかを実行します。
   - 予約の横にあるボタンを選択し、[編集] を選択します。
   - 予約リンクを選択し、[編集] を選択します。
5. DPU の場合は、必要なデータ処理ユニットの数を 4 個単位で選択または入力します。設定できる DPU の最小数は 24 個です。詳細については、「DPU について」を参照してください。

> **Note**
>
> DPU は既存のキャパシティ予約にいつでも追加できます。ただし、予約を作成または予約に DPU を追加してから 1 時間後までは DPU 数を減らすことはできません。

6. (オプション) [タグ] で、[削除] を選択してタグを削除するか、[タグの追加] を選択して新しいタグを追加します。
7. 送信 を選択します。予約の詳細ページには、更新された構成が表示されます。

予約へのワークグループの追加

キャパシティ予約を作成すると、予約に最大 20 のワークグループを追加できます。予約にワークグループを追加すると、リザーブドキャパシティでどのクエリを実行すべきかを Athena に伝えます。予約に関連付けられていないワークグループからのクエリは、引き続きテラバイト (TB) ごとのスキャン済みデフォルト価格モデルを使用して実行されます。

予約に 2 つ以上のワークグループがある場合、それらのワークグループからのクエリはリザーブドキャパシティを使用できます。ワークグループの追加と削除は随時行うことができます。ワークグループを追加または削除しても、実行中のクエリは中断されません。

予約が保留中ステータスの場合、追加したワークグループからのクエリは、予約がアクティブになるまで、引き続きテラバイト (TB) ごとのスキャン済みデフォルト価格モデルを使用して実行されます。

キャパシティ予約に 1 つ以上のワークグループを追加するには

1. [キャパシティ予約] の詳細ページで、[ワークグループの追加] を選択します。
2. [ワークグループの追加] ページで、追加するワークグループを選択し、[ワークグループの追加] を選択します。1 つのワークグループを複数の予約に割り当てることはできません。

   [キャパシティ予約] の詳細ページに、追加したワークグループが表示されます。これらのワークグループで実行されるクエリは、予約がアクティブなときに、予約したキャパシティを使用します。

予約からワークグループを削除する

ワークグループ専用のキャパシティが不要になった場合や、ワークグループを専用の予約に移動したい場合は、いつでも削除できます。予約からワークグループを削除するプロセスは簡単です。予約か

らワークグループを削除すると、削除されたワークグループからのクエリはデフォルトでオンデマンド (プロビジョニングされていない) キャパシティを使用するようになり、スキャンされたテラバイト (TB) に基づいて請求されます。

予約から 1 つ以上のワークグループを削除するには

1. キャパシティ予約の詳細ページで、削除するワークグループを選択します。
2. [ワークグループを削除] を選択します。[ワークグループを削除しますか？] のプロンプトが表示され、ワークグループを予約から削除する前に、現在アクティブなクエリがすべて終了することを通知します。
3. [削除] を選択します。キャパシティ予約の詳細ページには、削除されたワークグループがもう存在しないことが表示されます。

キャパシティ予約をキャンセルする

キャパシティ予約が不要になった場合は、キャンセルすることができます。予約を使用していたワークグループでまだ実行中のクエリは完了できますが、ワークグループ内の他のクエリは予約を使用しなくなります。

> **Note**
>
> キャンセルされたキャパシティが後日再予約可能になる保証はありません。キャパシティを別の予約、AWS アカウント、AWS リージョン に移すことはできません。

キャパシティ予約をキャンセルするには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. [管理]、[キャパシティ予約] を選択します。
4. キャパシティ予約のリストで、次のいずれかを実行します。
   - 予約の横にあるボタンを選択し、[キャンセル] を選択します。
   - [予約リンク] を選択してから、[キャパシティ予約のキャンセル] を選択します。
5. [キャパシティ予約をキャンセルしますか？] プロンプトで [キャンセル] を入力し、[キャパシティ予約のキャンセル] を選択します。

予約のステータスが [キャンセル中] に変わり、キャンセルが進行中であることを知らせる進行状況バナーが表示されます。

キャンセルが完了すると、キャパシティ予約はそのまま残りますが、ステータスは [キャンセル済み] と表示されます。予約はキャンセル後 45 日で削除されます。この 45 日間は、キャンセルされた予約を編集したり再利用したりすることはできませんが、タグを参照、表示して、過去の履歴を参照することはできます。

キャパシティ予約の削除

キャンセルされたキャパシティ予約への参照をすべて削除したい場合は、予約を削除できます。予約は削除する前にキャンセルする必要があります。削除された予約はアカウントから直ちに削除され、参照できなくなります (ARN を使用する場合を含む)。

キャパシティ予約を削除するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. [管理]、[キャパシティ予約] を選択します。
4. キャパシティ予約のリストで、次のいずれかを実行します。
   - キャンセルされた予約の横にあるボタンを選択し、[アクション]、[削除] の順に選択します。
   - 予約リンクを選択し、[削除] を選択します。
5. [キャパシティ予約を削除しますか?] プロンプトが表示されたら、[削除] を選択します。

   バナーは、キャパシティ予約が正常に削除されたことを通知します。削除された予約は、キャパシティ予約のリストに表示されなくなります。

キャパシティ予約の IAM ポリシー

キャパシティ予約へのアクセスを制御するには、リソースレベルの IAM 許可、またはアイデンティティベースの IAM ポリシーを使用します。IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、IAM ユーザーガイドの「IAM でのセキュリティベストプラクティス」を参照してください。

以下の手順は、Athena に固有の手順です。

IAM 固有の情報については、このセクションの最後に表示されているリンク先を参照してください。JSON キャパシティ予約ポリシーの例についての情報は、「キャパシティ予約ポリシーの例」を参照してください。

IAM コンソールのビジュアルエディタを使用してキャパシティ予約ポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[Policies] (ポリシー)、[Create policy] (ポリシーの作成) の順にクリックします。
3. [Visual editor] (ビジュアルエディタ) タブで、[Choose a service] (サービスの選択) をクリックします。次に、ポリシーに追加する Athena を選択します。
4. [Select actions] (アクションの選択) を選択し、ポリシーに追加するアクションを選択します。ビジュアルエディタが Athena で利用できるアクションを表示します。詳細については、「サービス承認リファレンス」の「Amazon Athena 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。
5. [アクションの追加] を選択して特定のアクションを入力、またはワイルドカード文字 (*) を使用して複数のアクションを指定します。

   デフォルトでは、作成しているポリシーが選択するアクションを許可します。Athena 内の `capacity-reservation` リソースに対するリソースレベルのアクセス許可をサポートするアクションを 1 つ、または複数選択すると、エディタが `capacity-reservation` リソースをリストします。
6. [リソース] を選択し、ポリシーに関する特定のキャパシティ予約を指定します。JSON キャパシティ予約ポリシーの例については、「キャパシティ予約ポリシーの例」を参照してください。
7. 以下のように `capacity-reservation` リソースを指定します。

   ```
   arn:aws:athena:<region>:<user-account>:capacity-reservation/<capacity-reservation-name>
   ```
8. [Review policy] (ポリシーの確認) をクリックして、作成するポリシーの [Name] (名前) と [Description (説明) (オプション) を入力します。ポリシー概要を確認して、意図したアクセス許可を付与したことを確認します。
9. [Create Policy] (ポリシーの作成) をクリックして、新しいポリシーを保存します。
10. このアイデンティティベースのポリシーをユーザー、グループ、またはロールにアタッチします。

詳細については、「サービス認証リファレンス」と「IAM ユーザーガイド」で以下のトピックを参照してください。

- Amazon Athena のアクション、リソース、および条件キー
- ビジュアルエディタでのポリシーの作成
- IAM ポリシーの追加と削除
- リソースへのアクセスの制御

JSON キャパシティ予約ポリシーの例については、「キャパシティ予約ポリシーの例」を参照してください。

Amazon Athena アクションの完全なリストについては、「Amazon Athena API リファレンス」の API アクション名を参照してください。

キャパシティ予約ポリシーの例

このセクションには、キャパシティ予約に対するさまざまなアクションを有効にするために使用できるポリシーの例が含まれています。IAM ポリシーを使用するときは、常に IAM のベストプラクティスに従うようにしてください。詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティベストプラクティス」を参照してください。

キャパシティ予約は、Athena で管理されている IAM リソースです。そのため、キャパシティ予約ポリシーで `capacity-reservation` を入力として取るアクションを使用する場合、キャパシティ予約の ARN を次のように指定する必要があります:

```
"Resource": [arn:aws:athena:<region>:<user-account>:capacity-reservation/<capacity-reservation-name>]
```

`<capacity-reservation-name>` の場所はキャパシティ予約の名前です。たとえば、`test_capacity_reservation` という名前のキャパシティ予約の場合は、次のようにリソースとして指定します:

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:capacity-reservation/
test_capacity_reservation"]
```

mazon Athena アクションの完全なリストについては、Amazon Athena API リファレンスの API アクション名を参照してください。IAM ポリシーの詳細については、「IAM ユーザーガイド」で「ビジュアルエディタでのポリシーの作成」を参照してください。

- Example policy to list capacity reservations
- Example policy for management operations

Example キャパシティ予約を一覧表示するポリシーの例

次のポリシーによって、すべてのユーザーがすべてのキャパシティ予約を一覧表示できるようになります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListCapacityReservations"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 管理オペレーションに関するポリシーの例

次のポリシーによって、ユーザーはキャパシティ予約 `test_capacity_reservation` を作成、キャンセル、更新、その詳細を取得できます。また、このポリシーによって、ユーザーは `workgroupA` と`workgroupB` を`test_capacity_reservation` に割り当てられます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect": "Allow",
         "Action": [
             "athena:CreateCapacityReservation",
             "athena:GetCapacityReservation",
             "athena:CancelCapacityReservation",
             "athena:UpdateCapacityReservation",
             "athena:GetCapacityAssignmentConfiguration",
             "athena:PutCapacityAssignmentConfiguration"
         ],
         "Resource": [
```

```
            "arn:aws:athena:us-east-1:123456789012:capacity-
reservation/test_capacity_reservation",
            "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA",
            "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupB"
         ]
      }
   ]
}
```

Athena キャパシティ予約 API

次のリストには、Athena キャパシティ予約 API アクションへの参照リンクが含まれています。データ構造とその他の Athena API アクションについては、「Amazon Athena API リファレンス」を参照してください。

- CancelCapacityReservation
- CreateCapacityReservation
- GetCapacityAssignmentConfiguration
- GetCapacityReservation
- ListCapacityReservations
- PutCapacityAssignmentConfiguration
- UpdateCapacityReservation

Athena でのパフォーマンスのチューニング

このトピックでは、Athena クエリのパフォーマンスを向上させるための一般的な情報と具体的な提案、および制限やリソース使用量に関連するエラーの回避方法について説明します。

Service Quotas

Athena では、クエリ実行時間、アカウント内の同時クエリ数、API リクエスト率などの指標に対してクォータを適用しています。これらのクォータの詳細については、「Service Quotas」を参照してください。これらのクォータを超えると、送信時またはクエリ実行中にクエリが失敗します。

このページのパフォーマンス最適化の多くのヒントは、クエリの実行時間を短縮するのに役立ちます。最適化によって容量の制限がなくなり、同時実行クォータ内でより多くのクエリを実行できるようになり、実行時間が長すぎたためにクエリがキャンセルされるのを防ぐことができます。

同時実行クエリと API リクエストの数のクォータは、AWS アカウント および AWS リージョン ごとです。ワークロードが同じクォータに対して競合しないように、AWS アカウント につき 1 つのワークロードを実行する (またはプロビジョニング済みのキャパシティ予約を別々に使用する) ことをお勧めします。

同じアカウントで 2 つのワークロードを実行すると、そのうちの 1 つのワークロードで大量のクエリを実行できます。これにより、残りのワークロードがスロットリングされたり、クエリの実行がブロックされたりする可能性があります。これを回避するには、ワークロードを個別のアカウントに移動して、各ワークロードに独自の同時実行クォータを割り当てることができます。片方または両方のワークロードのプロビジョニング済みキャパシティ予約を作成しても、同じ目標が達成されます。

その他のサービスのクォータ

Athena がクエリを実行すると、クォータを適用する他のサービスを呼び出せます。クエリの実行中に、Athena は AWS Glue Data Catalog、Amazon S3、および IAM や AWS KMS などその他の AWS サービスなどに API 呼び出しを行えます。フェデレーションクエリを使用する場合、Athena は AWS Lambda に呼び出しも行います。これらのサービスにはすべて、超過できる独自の制限と割り当てがあります。クエリ実行でこれらのサービスからエラーが発生すると、ソースサービスからのエラーも含めて失敗します。復元可能なエラーは再試行されますが、問題が時間内に解決されない場合、クエリが失敗する可能性があります。エラーメッセージをよく読んで、Athena からのものか別のサービスからのものかを判断してください。いくつかの関連するエラーは、このドキュメントで説明されています。

Amazon S3 サービスクォータに起因するエラーの回避方法の詳細については、このドキュメントの後半の「ファイルが多すぎにならないようにする」を参照してください。Amazon S3 のパフォーマンスの最適化の詳細については、「Amazon S3 ユーザーガイド」の「ベストプラクティスの設計パターン: Amazon S3 パフォーマンスの最適化」を参照してください。

リソースの制限

Athena は分散クエリエンジンでクエリを実行します。クエリを送信すると、Athena エンジンのクエリプランナーはクエリを実行するのに必要な計算能力を見積もり、それに応じて計算ノードのクラスターを準備します。DDL クエリなどの一部のクエリは、1 つのノードでのみ実行されます。大規模なデータセットに対する複雑なクエリでは、はるかに大きなクラスターで実行されます。ノードは均一であり、メモリ、CPU、およびディスク構成は同じです。Athena は、より要求の厳しいクエリを処理するためにスケールアップするのではなく、スケールアウトします。

クエリの要求は、クエリを実行するクラスターで使用可能なリソースを超えることがあります。この場合にはクエリは失敗し、「このスケールファクターでリソースを使い果たしました」というエラーが表示されます。

最も一般的に消費されるリソースはメモリですが、まれにディスク容量になる場合もあります。メモリエラーは通常、エンジンが結合またはウィンドウ関数を実行するときに発生しますが、カウントや集計が異なる場合もあります。

クエリが「リソース不足」エラーで一度失敗した場合でも、もう一度実行すると成功する可能性があります。クエリの実行は確定的ではありません。データの読み込みにかかる時間や中間データセットがノードにどのように分散されるかなどの要因により、リソースの使用量が異なる場合があります。たとえば、2 つのテーブルを結合するクエリで、結合条件の値の分布に大きな偏りがあるとします。このようなクエリはほぼ成功しますが、最も一般的な値が同じノードで処理されてしまうと失敗することがあります。

クエリが利用可能なリソースを超えないようにするには、このドキュメントに記載されているパフォーマンスチューニングのヒントを参考にしてください。特に、使用可能なリソースを使い果たすクエリを最適化する方法のヒントについては、「結合の最適化」、「ウィンドウ機能の最適化」、および「近似を使用したクエリの最適化」を参照してください

クエリを最適化するテクニック

このセクションで説明するクエリ最適化手法を使用して、クエリの実行を高速化したり、Athena のリソース制限を超えるクエリの回避策として使用します。

結合の最適化

分散クエリエンジンで結合を実行するには、さまざまな方法があります。最も一般的なのは、分散ハッシュ結合と複雑な結合条件のクエリの 2 つです。

分散型ハッシュ結合

最も一般的なタイプの結合では、結合条件として等価比較を使用します。Athena はこのタイプの結合を分散ハッシュ結合として実行します。

分散型ハッシュ結合では、エンジンは結合の片側からルックアップテーブル (ハッシュテーブル) を作成します。この側はビルド側と呼ばれます。ビルド側のレコードはノード全体に分散されます。各ノードは、そのサブセットのルックアップテーブルを作成します。結合のもう一方の側 (プローブ側と呼ばれる) がノードを介してストリーミングされます。プローブ側のレコードは、ビルド側と同じ方法でノードに分散されます。これにより、各ノードは独自の検索テーブルで一致するレコードを検索して結合を実行できます。

結合のビルド側から作成されたルックアップテーブルがメモリに収まらない場合、クエリが失敗する可能性があります。ビルド側の合計サイズが利用可能なメモリよりも小さい場合でも、レコードの分

布に大きな偏りがあると、クエリが失敗する可能性があります。極端なケースでは、すべてのレコードの結合条件の値が同じで、1 つのノードのメモリに収まる必要がある場合があります。スキューの少ないクエリでも、値のセットが同じノードに送信され、その値の合計が使用可能なメモリを超えると、失敗する可能性があります。ノードにはレコードをディスクにスピルする機能がありますが、スピルするとクエリの実行が遅くなり、クエリの失敗を防ぐには不十分である場合があります。

Athena は、大きい方のリレーションをプローブ側に、小さい方のリレーションをビルド側として使用するように結合の順序を変更しようとします。ただし、Athena はテーブル内のデータを管理しないため、情報が限られており、多くの場合、最初のテーブルが大きく、2 番目のテーブルが小さいと想定する必要があります。

等価ベースの結合条件で結合を記述する場合、JOIN キーワードの左側の表がプローブ側、右側の表がビルド側であると仮定します。正しいテーブル (ビルド側) がテーブルのうち小さい方であることを確認してください。結合のビルド側をメモリに収まるほど小さくできない場合は、ビルドテーブルのサブセットを結合する複数のクエリを実行することを検討してください。

その他の結合型

結合条件が複雑なクエリ (LIKE、>、または他の演算子を使用するクエリなど) では、多くの場合、計算負荷が高くなります。最悪の場合には、結合の片側のすべてのレコードを、結合の反対側のすべてのレコードと比較する必要があります。実行時間はレコード数の 2 乗に比例して長くなるため、このようなクエリには最大実行時間を超えるリスクがあります。

Athena がクエリをどのように実行するかを事前に確認するには、EXPLAIN ステートメントを使用できます。詳細については、Athena での EXPLAIN および EXPLAIN ANALYZE の使用およびAthena EXPLAIN ステートメントの結果についてを参照してください。

ウィンドウ機能の最適化

ウィンドウ関数はリソースを大量に消費する操作であるため、クエリの実行が遅くなったり、「このスケールファクターではリソースを使い果たしました」というメッセージが表示され、クエリが失敗したりする可能性があります。ウィンドウ関数は、結果を計算するために操作したすべてのレコードをメモリに保持します。ウィンドウが非常に大きい場合、ウィンドウ関数のメモリが不足する可能性があります。

クエリが使用可能なメモリ制限内で実行されるように、ウィンドウ関数が処理するウィンドウのサイズを小さくしてください。そのようにするためには、PARTITIONED BY 句を追加するか、既存の分割節の範囲を絞り込めます。

ウィンドウ以外の関数を代用する

ウィンドウ関数を含むクエリは、ウィンドウ関数なしで書き直せる場合があります。たとえば、row_number を使用して上位 N レコードを検索する代わりに、ORDER BY および LIMIT を使用できます。row_number または rank を使用してレコードの重複排除を行う代わりに、max_by、min_by、任意 などの集計関数を使用できます。

たとえば、センサーからの更新を含むデータセットがあるとします。センサーは定期的にバッテリーの状態を報告し、位置情報などのメタデータを含みます。各センサーの最新のバッテリー状態とその位置を知りたい場合は、次のクエリを使用できます。

```
SELECT sensor_id,
       arbitrary(location) AS location,
       max_by(battery_status, updated_at) AS battery_status
FROM sensor_readings
GROUP BY sensor_id
```

位置などのメタデータはすべてのレコードで同じなので、arbitrary 関数を使用してグループから任意の値を選択できます。

max_by 関数を使用すると、最新のバッテリー状態を取得できます。max_by 関数は、別の列の最大値が見つかったレコードから列の値を選択します。この場合、グループ内の最終更新時刻を含むレコードのバッテリーステータスを返します。このクエリは、ウィンドウ関数を使用する同等のクエリよりも実行速度が速く、メモリ使用量も少なくて済みます。

集約の最適化

Athena が集約を実行すると、GROUP BY 句内の列を使用してレコードがワーカーノード全体に分散されます。レコードをグループと照合するタスクを可能な限り効率的に行うために、ノードはレコードをメモリに保持し、必要に応じてディスクに書き出そうとします。

GROUP BY 句に重複する列を含めないようにするのも良い考えです。列の数が少ないと必要なメモリも少なくなるため、使用する列の数が少ないグループを記述するクエリの方が効率的です。また、数値列は文字列よりもメモリ使用量が少なくなります。たとえば、数値のカテゴリ ID とカテゴリ名の両方を持つデータセットを集約する場合、GROUP BY 句にはカテゴリ ID 列のみを使用してください。

場合によっては、列が GROUP BY 句または集計式の一部である必要があるという事実を回避するために、クエリの GROUP BY 句に列が含まれることがあります。このルールに従わない場合、次のようなエラーメッセージが表示されることがあります。

EXPRESSION_NOT_AGGREGATE: 1:8 行目:「カテゴリ」は集合式であるか、GROUP BY 句に含まれている必要があります

GROUP BY 句に余分な列を追加しなくても済むように、次の例のように任意の関数を使用できます。

```
SELECT country_id,
       arbitrary(country_name) AS country_name,
       COUNT(*) AS city_count
FROM world_cities
GROUP BY country_id
```

ARBITRARY 関数はグループから任意の値を返します。この関数は、グループ内のすべてのレコードが 1 つの列に対して同じ値を持つことは把握しているがその値がグループを識別できない場合に便利です。

上位 N のクエリの最適化

ORDER BY 句は、クエリの結果をソートされた順序で返します。Athena は分散ソートを使用して、ソート操作を複数のノードで並行して実行します。

結果を厳密にソートする必要がない場合は、ORDER BY 句を追加しないでください。また、必ずしも必要ではない場合は、内部クエリへの ORDER BY の追加は避けてください。多くの場合、クエリプランナーは冗長なソートを削除できますが、これは保証されていません。このルールの例外は、内部クエリが上位 N の操作 (最新の N の値、または最も一般的な Nの値を検索するなど) を実行している場合です。

Athena が ORDER BY を LIMIT を使用して見た場合、実行されている N クエリが上位のクエリであることを認識し、それに応じて専用の操作を行います。

> **Note**
>
> Athena では、上位の N を使用する row_number のようなウィンドウ関数も多くの場合検出できますが、ORDER BY と LIMIT を使用するより単純なバージョンをお勧めします。詳細については、「ウィンドウ機能の最適化」を参照してください。

必要な列のみを含める

列が必ずしも必要ではない場合は、クエリに含めないでください。クエリが処理しなければならないデータが少ないほど、実行速度は速くなります。これにより、必要なメモリ量と、ノード間で送信す

る必要のあるデータ量の両方が削減されます。列形式のファイル形式を使用している場合、列の数を減らすと、Amazon S3 から読み取られるデータの量も減ります。

Athena には結果の列数に特別な制限はありませんが、クエリの実行方法によって、可能な列の合計サイズが制限されます。列を合わせたサイズには、名前とタイプが含まれます。

たとえば、次のエラーは、リレーションがリレーション記述子のサイズ制限を超えるために発生します。

GENERIC_INTERNAL_ERROR: io.airlift.bytecode.CompilationException

この問題を回避するには、クエリ内の列の数を減らす、またはサブクエリを作成して、より少ない量のデータを取得する JOIN を使用します。一番外側のクエリで SELECT * を実行するクエリがある場合は、* を必要な列のみのリストに変更する必要があります。

近似を使用したクエリの最適化

Athena では、近似集計関数をサポートしており、個別値、最も頻度の高い値、パーセンタイル (近似中央値を含む) のカウント、およびヒストグラムの作成を行えます。これらの関数は、正確な値が不要な場合に使用してください。

COUNT(DISTINCT col) 操作とは異なり、approx_distinct はメモリ使用量がはるかに少なく、実行速度も速くなります。同様に、ヒストグラムの代わりに numeric_histogram を使用すると、近似法を使用するため、メモリ使用量が少なくなります。

LIKE の最適化

LIKE を使用して一致する文字列を検索できますが、文字列が長い場合は計算量が多くなります。regexp_like 関数は、ほとんどの場合、より高速な代替手段であり、柔軟性にも優れています。

多くの場合、探している部分的な文字列を固定することで検索を最適化できます。たとえば、プレフィックスを探している場合は、'% *substr*%' の代わりに '*substr* %' を使用する方がはるかに優れています。または regexp_like、「^ *substr*」を使用している場合。

UNION の代わりに UNION ALL を使用する

UNION ALL と UNION は、2 つのクエリの結果を 1 つの結果にまとめる 2 つの方法です。 UNION ALL は最初のクエリのレコードを 2 番目のクエリと連結し、UNION は同じ処理を実行しますが、重複も削除します。UNION ではすべてのレコードを処理して重複を見つける必要があり、メモリと計算を大量に消費しますが、UNION ALL は比較的高速な操作です。レコードの重複排除が必要でない限り、UNION ALL はベストパフォーマンスを実現するために使用してください。

大きな結果セットには UNLOAD を使用する

クエリの結果が大きくなることが予想される場合 (たとえば、数万行以上など)、UNLOAD を使用して結果をエクスポートします。ほとんどの場合、これは通常のクエリを実行するよりも高速です。また、UNLOAD を使用すると、出力をより細かく制御できます。

クエリの実行が終了すると、Athena は結果を 1 つの非圧縮 CSV ファイルとして Amazon S3 に保存します。結果が圧縮されないだけでなく、操作を並列化できないため、UNLOAD よりも時間がかかります。これとは対照的に、UNLOAD は結果をワーカーノードから直接書き込み、計算クラスターの並列処理を最大限に活用します。さらに、結果を圧縮形式や JSON や Parquet などの他のファイル形式で書き込むように UNLOAD を構成できます。

詳細については、「UNLOAD」を参照してください。

CTAS または Glue ETL を使用して、頻繁に使用する集計をマテリアライズする

クエリの「マテリアライズ」とは、事前に計算された複雑なクエリ結果 (集計や結合など) を保存して後続のクエリで再利用することで、クエリのパフォーマンスを向上させる方法です。

クエリの多くに同じ結合や集計が含まれている場合は、共通のサブクエリを新しいテーブルとして作成し、そのテーブルに対してクエリを実行できます。クエリ結果からのテーブルの作成 (CTAS)、または Glue ETL などの専用 ETL ツールを使用して新しいテーブルを作成できます。

たとえば、オーダーデータセットのさまざまな側面を表示するウィジェットを含むダッシュボードがあるとします。各ウィジェットには独自のクエリがありますが、クエリはすべて同じ結合とフィルターを共有します。注文テーブルは品目テーブルと結合され、過去 3 か月のみ表示するフィルターがあります。これらのクエリに共通する機能がわかれば、ウィジェットが使用できる新しいテーブルを作成できます。これにより、重複が減り、パフォーマンスが向上します。欠点は、新しいテーブルを最新の状態に保つ必要があることです。

クエリ結果の再利用

同じクエリが短期間に複数回実行されることはよくあります。たとえば、複数のユーザーが同じデータダッシュボードを開いたときに発生する可能性があります。クエリを実行するときに、以前に計算した結果を再利用するよう Athena に指示できます。再利用できる結果の最大保管期間を指定します。同じクエリがその時間枠内で以前に実行された場合、Athena はクエリを再実行する代わりにそれらの結果を返します。詳細については、「Amazon Athena ユーザーガイド」の クエリの結果を再利用する、「AWS ビッグデータブログ」の「Amazon Athena クエリ結果の再利用によるコスト削減とクエリパフォーマンスの向上」を参照してください。

データを最適化するテクニック

パフォーマンスはクエリだけでなく、データセットの構成方法や使用するファイル形式と圧縮にも大きく依存します。

データのパーティション化

パーティショニングを行うと、テーブルが複数部分に分割され、日付、国、地域などのプロパティに基づいて関連データがまとめられます。パーティションキーは仮想列として機能します。パーティションキーはテーブルの作成時に定義し、クエリのフィルタリングに使用します。パーティションキー列をフィルタリングすると、一致するパーティションのデータのみが読み取られます。たとえば、データセットが日付で分割されていて、クエリに先週のみ一致するフィルターが設定されている場合、先週のデータのみが読み取られます。パーティショニングについての詳細は、「Athena でのデータのパーティション化」を参照してください。

クエリをサポートするパーティションキーを選択する

パーティショニングはクエリのパフォーマンスに大きな影響を与えるため、データセットとテーブルを設計するときは、パーティションの分割方法を慎重に検討してください。パーティションキーが多すぎると、データセットが断片化し、ファイルが多すぎたり、ファイルが小さすぎたりする可能性があります。逆に、パーティションキーが少なすぎたり、パーティショニングがまったく行われなかったりすると、クエリが必要以上に多くのデータをスキャンすることになります。

まれなクエリの最適化は避ける

最もよく使われるクエリを最適化し、まれであるクエリには最適化しないようにするのが良い戦略です。たとえば、クエリが日単位の期間を対象としている場合は、一部のクエリがそのレベルに絞り込まれていても、時間単位で分割しないでください。データに詳細なタイムスタンプ列がある場合、時間別にフィルタリングするまれなクエリではタイムスタンプ列を使用できます。まれなケースで必要以上のデータをスキャンしたとしても、まれなケースという理由で全体的なパフォーマンスを低下させることは、通常は良いトレードオフとは言えません。

クエリでスキャンするデータ量を減らしてパフォーマンスを向上させるには、列指向のファイル形式を使用し、レコードをソートしたままにしてください。時間ごとに分割する代わりに、レコードをタイムスタンプでソートしておきます。短い時間枠でのクエリでは、タイムスタンプによるソートは、時間ごとの分割とほぼ同じくらい効率的です。さらに、通常はタイムスタンプでソートしても、日単位でカウントされたタイムウィンドウでのクエリのパフォーマンスが低下することはありません。詳細については、「列指向ファイルフォーマットを使用する」を参照してください。

数万のパーティションを含むテーブルでのクエリは、すべてのパーティションキーに述語がある方がパフォーマンスが向上することに注意してください。これが、最も一般的なクエリ用にパーティショ

ンスキームを設計するもう一つの理由です。詳細については、「等式によるパーティションのクエリ」を参照してください。

パーティション射影を使用する

パーティション射影は Athena の機能の 1 つであり、パーティション情報を AWS Glue Data Catalog には格納せず、AWS Glue のテーブルのプロパティにルールとして格納します。Athena は、パーティション射影が設定されたテーブルに対してクエリを計画する際、テーブルのパーティション射影ルールを読み取ります。Athena は、AWS Glue Data Catalog でパーティションを検索する代わりに、クエリとルールに基づいてメモリに読み込むパーティションを計算します。

パーティション射影は、パーティション管理を簡素化するだけでなく、多数のパーティションを持つデータセットのパフォーマンスを向上させることができます。クエリにパーティションキーの特定の値の代わりに範囲が含まれている場合、カタログで一致するパーティションを検索すると、パーティションの数が多いほど時間がかかります。パーティション射影を使用すると、カタログにアクセスせずにフィルターをメモリ内で計算でき、はるかに高速になります。

特定の状況では、パーティション射影によってパフォーマンスが低下する可能性があります。一例として、テーブルが「スパースな」場合が挙げられます。スパーステーブルには、パーティション射影の設定で記述されたパーティションキー値のすべての順列に関するデータはありません。スパーステーブルでは、クエリから計算されたパーティションセットとパーティションプロジェクション設定は、データがない場合でもすべて Amazon S3 に一覧表示されます。

パーティション射影を使用する場合は、必ずすべてのパーティションキーに述語を含めてください。Amazon S3 の一覧表示が不要にならないように、指定できる値の範囲を絞り込んでください。100 万の値の範囲を持つパーティションキーと、そのパーティションキーにフィルターがないクエリを想像してください。クエリを実行するには、Athena は少なくとも 100 万回の Amazon S3 リストオペレーションを実行する必要があります。パーティション射影を使用するか、カタログにパーティション情報を保管するかに関係なく、特定の値に対してクエリを実行すると、クエリは最も速くなります。詳細については、「等式によるパーティションのクエリ」を参照してください。

パーティション射影用のテーブルを設定するときは、指定する範囲が適切であることを確認してください。クエリにパーティションキーの述語が含まれていない場合は、そのキーの範囲内のすべての値が使用されます。データセットが特定の日付に作成された場合は、その日付を任意の日付範囲の開始点として使用します。終了日の範囲として `NOW` を使用します。値の数が多い数値範囲は避け、代わりに注入された型の使用を検討してください。

パーティション射影の詳細については、「Amazon Athena でのパーティション射影」を参照してください。

パーティションインデックスの使用

パーティションインデックスは、多数のパーティションを持つテーブルのパーティションルックアップのパフォーマンスを向上させる AWS Glue Data Catalog の機能です。

カタログ内のパーティションのリストは、リレーショナルデータベースのテーブルのようなものです。このテーブルには、パーティションキー用の列と、パーティションの場所用の別の列があります。分割テーブルをクエリすると、このテーブルをスキャンしてパーティションの場所が検索されます。

リレーショナルデータベースと同様に、インデックスを追加することでクエリのパフォーマンスを向上できます。複数のインデックスを追加して、さまざまなクエリパターンをサポートできます。AWS Glue Data Catalog パーティションインデックスは、>、>= などの等価演算子と比較演算子の両方をサポートし、< は AND 演算子を結合します。詳細については、「AWS Glue 開発者ガイド」の「AWS Glue でのパーティションインデックスの使用」と、「AWS ビッグデータブログ」の「AWS Glue Data Catalog パーティションインデックスを使用して Amazon Athena クエリのパフォーマンスを向上させるETL ジョブを構成するには」を参照してください。

パーティションキーのタイプとして常に STRING を使用する

パーティションキーをクエリする場合、Athena ではパーティションフィルタリングを AWS Glue にプッシュダウンするためにパーティションキーのタイプ `STRING` が必要であることを覚えておいてください。パーティションの数が少なくない場合は、他のタイプを使用するとパフォーマンスが低下する可能性があります。パーティションキーの値が日付型または数値型の場合は、クエリ内の適切な型にキャストしてください。

古いパーティションや空のパーティションを削除する

Amazon S3 のパーティションから (Amazon S3 ライフサイクルを使用するなどして) データを削除する場合は、AWS Glue Data Catalog からそのパーティションエントリも削除する必要があります。クエリのプランニング中、クエリに一致するパーティションはすべて Amazon S3 に一覧表示されます。空のパーティションが多数ある場合、それらのパーティションを一覧表示することによるオーバーヘッドは悪影響をおよぼす可能性があります。

また、パーティションが何千もある場合は、不要になった古いデータのパーティションメタデータを削除することを検討してください。たとえば、クエリで 1 年以上前のデータが検索されない場合、古いパーティションのパーティションメタデータを定期的に削除できます。パーティションの数が数万に増えた場合、未使用のパーティションを削除すると、すべてのパーティションキーに述語が含まれていないクエリを高速化できます。クエリにすべてのパーティションキーに述語を含める方法については、「等式によるパーティションのクエリ」を参照してください。

等式によるパーティションのクエリ

すべてのパーティションキーに等価述語を含むクエリでは、パーティションメタデータを直接読み込めるため、実行速度が速くなります。1 つ以上のパーティションキーに述語がない場合や、述語で値の範囲を選択するクエリは避けてください。このようなクエリでは、すべてのパーティションのリストをフィルタリングして、一致する値を見つける必要があります。ほとんどのテーブルではオーバーヘッドは最小限ですが、パーティションが数万以上あるテーブルではオーバーヘッドが大きくなる可能性があります。

クエリを書き直してパーティションを等値でフィルタリングできない場合は、パーティション射影を試してみてください。詳細については、「パーティション射影を使用する」を参照してください。

パーティションのメンテナンスに MSCK REPAIR TABLE を使用しないでください

MSCK REPAIR TABLE は実行に時間がかかる場合があり、新しいパーティションを追加するだけで、古いパーティションは削除されないため、パーティションを管理する効率的な方法ではありません (「考慮事項と制約事項」を参照)。

パーティションは、AWS Glue Data Catalog API、ALTER TABLE ADD PARTITION または AWS Glue クローラーを使用して手動で管理する方が適切です。別の方法として、パーティション射影を使用できます。これにより、パーティションを完全に管理する必要がなくなります。詳細については、「Amazon Athena でのパーティション射影」を参照してください。

クエリがパーティションスキームと互換性があることを確認してください。

クエリがどのパーティションをスキャンするかは、EXPLAIN ステートメントを使用して事前に確認できます。クエリの前に EXPLAIN キーワードを付けてから、EXPLAIN 出力の下部にある各テーブルのソースフラグメント (たとえば、Fragment 2 [SOURCE]) を探します。右側がパーティションキーとして定義されている割り当てを探してください。下の行には、クエリの実行時にスキャンされるパーティションキーのすべての値のリストが含まれています。

たとえば、dt パーティションキーを含むテーブルに対してクエリを実行し、EXPLAIN のクエリのプレフィックスを付けたとします。クエリ内の値が日付で、フィルターで 3 日間の範囲を選択した場合、EXPLAIN 出力は次のようになります。

```
dt := dt:string:PARTITION_KEY
    :: [[2023-06-11], [2023-06-12], [2023-06-13]]
```

EXPLAIN 出力は、プランナーがクエリと一致するこのパーティションキーの値を 3 つ見つけたことを示しています。また、それらの値が何であるかも表示されます。EXPLAIN の使用の詳細について

は、「Athena での EXPLAIN および EXPLAIN ANALYZE の使用」と「Athena EXPLAIN ステートメントの結果について」を参照してください。

列指向ファイルフォーマットを使用する

Parquet や ORC のような列指向のファイル形式は、分散分析ワークロード向けに設計されています。データを行別ではなく列別に整理します。データを列形式で整理することには、次のような利点があります。

- クエリに必要な列のみが読み込まれる
- ロードする必要のあるデータの総量が減る
- 列の値はまとめて保存されるため、データを効率的に圧縮できる
- ファイルには、エンジンが不要なデータのロードをスキップできるようにするメタデータを含められる

ファイルメタデータの使用方法の例として、ファイルメタデータには、データページの最小値と最大値に関する情報を含められます。クエリされた値がメタデータに記載されている範囲にない場合は、ページをスキップできます。

このメタデータを使用してパフォーマンスを向上させる方法の 1 つは、ファイル内のデータを確実にソートすることです。たとえば、`created_at` エントリが短期間にあるレコードを検索するクエリがあるとします。`created_at` データが列でソートされている場合、Athena はファイルメタデータの最小値と最大値を使用して、データファイルの不要な部分をスキップできます。

列指向のファイル形式を使用する場合は、ファイルが小さすぎないことを確認してください。ファイルが多すぎにならないようにする で説明したように、小さなファイルが多いデータセットはパフォーマンスの問題を引き起こします。これは特に列指向ファイル形式の場合に当てはまります。小さなファイルの場合、列指向ファイル形式のオーバーヘッドの方がメリットよりも重要です。

Parquet と ORC は内部的には行グループ (Parquet) とストライプ (ORC) で整理されていることに注意してください。行グループのデフォルトサイズは 128 MB、デフォルトのストライプサイズは 64 MB です。列が多い場合は、行グループとストライプサイズを大きくしてパフォーマンスを向上できます。行グループまたはストライプサイズをデフォルト値より小さくすることはお勧めしません。

他のデータ形式を Parquet または ORC に変換するには、AWS Glue ETL または Athena を使用できます。ETL の Athena の使用に関する詳細については、「ETL およびデータ分析での CTAS および INSERT INTO の使用」を参照してください。

データを圧縮する

Athena は幅広い圧縮形式をサポートしています。圧縮データのクエリは、解凍前にスキャンされたバイト数に応じて支払われるため、高速かつ安価です。

gzip 形式は圧縮率が高く、他のツールやサービスを幅広くサポートしています。zstd (Zstandard) 形式は、パフォーマンスと圧縮率のバランスが取れた新しい圧縮形式です。

JSON や CSV データなどのテキストファイルを圧縮するときは、ファイル数とファイルサイズのバランスをとるようにしてください。ほとんどの圧縮形式では、リーダーはファイルを最初から読み取る必要があります。つまり、圧縮されたテキストファイルは一般的に並行して処理できません。大きな非圧縮ファイルは、クエリ処理中の並列性を高めるためにワーカー間で分割されることがよくありますが、これはほとんどの圧縮形式では不可能です。

ファイルが多すぎにならないようにする で説明したように、ファイルが多すぎたり少なすぎたりしない方がよいでしょう。ファイル数によってクエリを処理できるワーカーの数が制限されるため、このルールは圧縮ファイルの場合に特に当てはまります。

Athena での圧縮の使用の詳細については、「Athena での圧縮のサポート」を参照してください。

カーディナリティが高いキーの検索にはバケットを使用する

バケット処理とは、いずれかの列の値に基づいてレコードを個別のファイルに分散する手法です。これにより、同じ値を持つすべてのレコードが同じファイルにあることが保証されます。バケット化は、カーディナリティの高いキーがあり、クエリの多くがそのキーの特定の値を検索する場合に役立ちます。

たとえば、特定のユーザーのレコードセットをクエリするとします。データがユーザー ID 別にまとめられている場合、Athena は特定の ID のレコードを含むファイルと含まないファイルを事前に把握します。これにより、Athena はその ID を含められるファイルのみを読み取れるため、読み取るデータ量を大幅に削減できます。また、特定の ID のデータを検索するために必要となる計算時間も短縮されます。

バケット化のデメリット

クエリでデータがバケット化されている列で複数の値を頻繁に検索する場合、バケット化の価値は低くなります。クエリする値が多いほど、すべて、またはほとんどのファイルを読み取らなければならない可能性が高くなります。たとえば、バケットが 3 つあり、クエリが 3 つの異なる値を検索する場合、すべてのファイルを読み取る必要がある場合があります。バケット処理は、クエリが単一の値を検索するときに最も効果的です。

詳細については、「Athena におけるパーティション化とバケット化」を参照してください。

ファイルが多すぎにならないようにする

データセットが多数の小さなファイルで構成されていると、全体的なクエリパフォーマンスが低下します。Athena がクエリを計画すると、すべてのパーティションの場所が一覧表示されるため、時間がかかります。各ファイルの処理と要求には、計算上のオーバーヘッドもあります。したがって、Amazon S3 から 1 つの大きなファイルをロードする方が、多数の小さいファイルから同じレコードをロードするよりも高速です。

極端なケースでは、Amazon S3 のサービス制限が発生する可能性があります。Amazon S3 は、1 つのインデックスパーティションに対して 1 秒あたり最大 5,500 の要求をサポートします。最初は、バケットは単一のインデックスパーティションとして扱われますが、リクエストの負荷が増えると、複数のインデックスパーティションに分割できます。

Amazon S3 はリクエストパターンを調べ、キープレフィックスに基づいて分割します。データセットが何千ものファイルで構成されている場合、Athena からのリクエストはリクエストクォータを超える可能性があります。ファイルが少なくても、同じデータセットに対して複数のクエリを同時に実行すると、クォータを超える可能性があります。同じファイルにアクセスする他のアプリケーションが、リクエストの総数に影響する可能性があります。

リクエストレート `limit` を超えると、Amazon S3 は次のエラーを返します。このエラーは Athena のクエリのステータス情報に含まれています。

SlowDown: リクエスト率を下げてください

トラブルシューティングを行うには、まず、エラーの原因が単一のクエリなのか、同じファイルを読み取る複数のクエリなのかを判断します。後者の場合は、同時に実行されないようにクエリの実行を調整します。これを実現するには、アプリケーションにキューイングメカニズムを追加するか、リトライを行う必要があります。

1 つのクエリを実行しても エラーが発生する場合は、データファイルを組み合わせるか、クエリを変更して読み込むファイルの数を減らしてみてください。小さなファイルを結合する最適なタイミングは、ファイルが書き込まれる前です。そのためには、以下の方法を検討してください。

- ファイルを書き込むプロセスを変更して、より大きなファイルを書き込んでください。たとえば、レコードが書き込まれる前にレコードを長時間バッファリングできます。
- Amazon S3 上の場所にファイルを置き、Glue ETL などのツールを使用してファイルをより大きなファイルにまとめます。次に、サイズの大きいファイルをテーブルが指す場所に移動します。詳細

については、「AWS Glue デベロッパーガイド」の「大きなグループの入力ファイルの読み込み」か、および「AWS re:Post ナレッジセンター」の「より大きなファイルを出力するように AWS Glue ETL ジョブを構成するには」を参照してください。

- パーティションキーの数を減らしてください。パーティションキーが多すぎると、各パーティションのレコード数が少なくなり、小さなファイルが多すぎることがあります。作成するパーティションの決定については、「クエリをサポートするパーティションキーを選択する」を参照してください。

パーティションの外部にストレージ階層を追加することは避けてください

クエリプランニングのオーバーヘッドを回避するには、ファイルを各パーティションの場所にフラット構造で格納します。追加のディレクトリ階層は使用しないでください。

Athena がクエリを計画すると、クエリに一致するすべてのパーティション内のすべてのファイルが一覧表示されます。Amazon S3 自体にはディレクトリはありませんが、慣例として / スラッシュはディレクトリ区切り文字として解釈されます。Athena がパーティションの場所を一覧表示すると、見つかったすべてのディレクトリが再帰的に一覧表示されます。パーティション内のファイルを 1 つの階層にまとめると、リストが複数回表示されます。

すべてのファイルが直接パーティションの場所にある場合は、ほとんどの場合、リスト操作を 1 回実行するのみで済みます。ただし、Amazon S3 はリストオペレーションごとに 1000 個のオブジェクトしか返さないため、パーティションに 1000 を超えるファイルがある場合は、複数の連続リストオペレーションが必要です。また、1 つのパーティションに 1000 個を超えるファイルがあると、さらに深刻なパフォーマンスの問題が発生する可能性があります。詳細については、「ファイルが多すぎにならないようにする」を参照してください。

SymlinkTextInputFormat は必要な場合にのみ使用する

SymlinkTextInputFormat テクニックを使用すると、テーブルのファイルがパーティションにきちんと整理されていない状況を回避できます。たとえば、シンボリックリンクは、すべてのファイルが同じプレフィックスにある場合や、スキーマの異なるファイルが同じ場所にある場合に役立ちます。

ただし、シンボリックリンクを使用すると、クエリ実行に間接的なレベルが追加されます。これらのレベルのインダイレクションは、全体的なパフォーマンスに影響します。シンボリックリンク ファイルを読み取り、ファイルが定義する場所をリストする必要があります。これにより、通常の Hive テーブルでは不要な複数のラウンドトリップが Amazon S3 に追加されます。結論として、ファイルの再編成などのより適切なオプションが利用できない場合にのみ `SymlinkTextInputFormat` を使用してください。

追加リソース

Athena でのパフォーマンスチューニングの詳細については、次のリソースを参照してください。

- AWS ビッグデータブログ記事「Amazon Athena のパフォーマンスチューニング Tips トップ 10」を読む
- フェデレーテッドクエリのパフォーマンスを向上させるために述語プッシュダウンを使用する方法の記事については、「AWS Big Data Blog」の「Improve federated queries with predicate pushdown in Amazon Athena」を参照してください。
- Athena クエリエンジンでのパフォーマンス最適化に関する記事については、AWS ビッグデータブログで「Run queries 3x faster with up to 70% cost savings on the latest Amazon Athena engine」を参照してください。
- 「AWS ビッグデータブログ」では、Athena に関する他の記事もご覧いただけます。
- Amazon Athena タグを使用して、AWS re: Post について質問する
- AWS ナレッジセンターの Athena に関するトピックを参照する
- 連絡先 AWS Support (AWS Management Console で、[サポート]、[サポートセンター] の順にクリック)

Amazon S3 スロットリングの防止

スロットリングは、サービス、アプリケーション、またはシステムを使用する速度を制限するプロセスです。AWS では、スロットリングを使用して Amazon S3 サービスの過剰使用を防ぎ、すべてのユーザーの Amazon S3 の可用性と応答性を高めることができます。ただし、スロットリングによって Amazon S3 との間でデータを送受信できる速度が制限されるため、インタラクションが制限されるのを防ぐことを検討することが重要です。

サービスレベルでのスロットリングを軽減

サービスレベルでの Amazon S3 スロットリングを回避するには、使用状況を監視して Service Quotas を調整するか、パーティション化などの特定の手法を使用します。スロットリングの原因となる可能性があるいくつかの条件は、以下の通りです。

- [アカウントの API リクエスト制限の超過] — Amazon S3 には、アカウントタイプと使用状況に基づくデフォルトの API リクエスト制限があります。1 つのオブジェクトの 1 秒あたりの最大リクエスト数を超えると、Amazon S3 サービスの過負荷を防ぐためにリクエストが制限されることがあります。

- [データのパーティション分割が不十分] — データを適切に分割せずに大量のデータを転送すると、Amazon S3 がリクエストを制限する場合があります。パーティション分割の詳細については、このドキュメントの「パーティション分割を使用する」セクションを参照してください。
- [多数の小さいオブジェクト] — 可能であれば、小さいファイルを多数使用することを避けてください。Amazon S3 には パーティション化されたプレフィックスごとに 1 秒あたり 5,500 件の GET リクエスト という制限があり、Athena のクエリにもこれと同じ制限があります。単一のクエリで何百万もの小さなオブジェクトをスキャンする必要がある場合は、クエリが Amazon S3 によって制限される可能性が高くなります。

過度のスキャンを避けるには、AWS Glue ETL を使用してファイルを定期的に圧縮する、またはテーブルをパーティション化してパーティションキーフィルターを追加します。詳細については、以下のリソースを参照してください。

- より大きなファイルを出力するように AWS Glue ETL ジョブを設定する方法を教えてください。(AWS ナレッジセンター)
- 大規模なグループでの入力ファイルの読み込み (AWS Glue 開発者ガイド)

テーブルの最適化

スロットリングの問題が発生した場合は、データを構造化することが重要です。Amazon S3 は大量のデータを処理できますが、データの構造が原因でスロットリングが発生することがあります。

以下のセクションでは、スロットリングの問題を回避するために Amazon S3 でデータを構造化するいくつかの方法について提案します。

パーティション分割を使用する

パーティション分割を使用すると、いつでもアクセスする必要があるデータ量を制限することで、スロットリングを減らすことができます。特定の列のデータをパーティション分割することで、複数のオブジェクトにリクエストを均等に分散し、1 つのオブジェクトに対するリクエスト数を減らすことができます。スキャンする必要のあるデータ量を減らすことは、クエリのパフォーマンスの向上とコストの削減につながります。

テーブルを作成するときに、仮想列として機能するパーティションを定義できます。`CREATE TABLE` ステートメント内にパーティションを含むテーブルを作成するには、`PARTITIONED BY (column_name data_type)` 句を使用してデータを分割するキーを定義します。

クエリによってスキャンされるパーティションを制限するには、クエリの `WHERE` 句にそれらを述語として指定できます。したがって、フィルターとして頻繁に使用される列はパーティション分割に

適しています。一般的な方法では、時間間隔に基づいてデータをパーティション化します。これにより、通常、複数レベルのパーティション構成となります。

パーティション分割にはコストもかかることに注意してください。テーブル内のパーティション数を増やすと、パーティションメタデータの取得と処理に必要な時間も長くなります。そのため、パーティション分割が過剰になると、より計画的にパーティション分割を行うことで得られるメリットが得られない可能性があります。データが 1 つのパーティション値に大きく偏っていて、ほとんどのクエリがその値を使用している場合は、追加のオーバーヘッドが発生する可能性があります。

Athena でのパーティション分割の詳細については、「パーティション化とは」を参照してください。

データのバケット化

データを分割するもう 1 つの方法は、データを 1 つのパーティションにバケット化することです。バケット化するには、グループ化する行を含む 1 つまたは複数の列を指定します。次に、それらの行を複数のバケットに入れます。この方法では、読み取りが必要なバケットのみをクエリできるため、スキャンする必要のあるデータ行の数が減ります。

バケット化に使用する列を選択する際に、カーディナリティが高く (つまり、多数の異なる値があり)、均一に分布していて、データのフィルタリングによく使用する列を選択します。バケット化に適した列の例としては、ID 列などのプライマリキーがあります。

Athena でのバケット化の詳細については、「バケット化とは」を参照してください。

AWS Glue パーティションインデックスの使用

AWS Glue パーティションインデックスを使用すると、1 つ以上のパーティション値に基づくテーブル内のデータを整理できます。AWS Glue パーティションインデックスを使用すると、データ転送回数、データ処理量、およびクエリの処理時間を削減できます。

AWS Glue パーティションインデックスは、パーティションキーとその値など、テーブル内のパーティションに関する情報を含むメタデータファイルです。パーティションインデックスは Amazon S3 バケットに保存され、新しいパーティションがテーブルに追加されると AWS Glue によって自動的に更新されます。

AWS Glue パーティションインデックスが存在する場合は。クエリでは、テーブル内のすべてのパーティションをロードするのではなく、パーティションのサブセットを取得しようとします。クエリは、クエリに関連するデータのサブセットでのみ実行されます。

AWS Glue でテーブルを作成する際は、テーブルで定義されたパーティションキーの任意の組み合わせに対して、パーティションインデックスを作成できます。テーブルに 1 つ以上のパーティション

インデックスを作成したら、パーティションフィルタリングを有効にするプロパティをテーブルに追加する必要があります。その後、Athena からテーブルにクエリを実行できます。

AWS Glue でのパーティションインデックス作成のステップについては、「AWS Glue デベロッパーガイド」の「AWS Glue でパーティションインデックスを使用する」を参照してください。テーブルプロパティを追加してパーティションフィルタリングを有効にする方法については、「AWS Glue パーティションのインデックス作成とフィルタリング」を参照してください。

データ圧縮とファイル分割の使用

ファイルが最適なサイズである場合や、ファイルを論理グループに分割できる場合は、データ圧縮によりクエリを大幅に高速化できます。一般に、圧縮率が高いほど、データを圧縮および解凍するためにより多くの CPU サイクルが必要になります。Athena の場合は、デフォルトでデータを圧縮する Apache Parquet または Apache ORC のいずれかを使用することをお勧めします。Athena でのデータ圧縮の詳細については、「Athena での圧縮のサポート」を参照してください。

ファイルを分割すると、Athena は 1 つのファイルを読み取るタスクを複数のリーダーに分散できるため、並列処理が向上します。1 つのファイルを分割できない場合、他のリーダーがアイドル状態のときに 1 つのリーダーだけがファイルを読み取ることができます。Apache Parquet と Apache ORC は分割可能なファイルもサポートしています。

最適化された列指向データストアを使用する

データを列指向形式に変換すると、Athena クエリのパフォーマンスが大幅に向上します。列指向ファイルを生成する場合、検討すべき最適化手法の 1 つは、パーティションキーに基づいてデータを順序付けることです。

Apache Parquet と Apache ORC は、オープンソースの列指向データストアとしてよく使用されています。既存の Amazon S3 データソースをこれらの形式のいずれかに変換する方法については、「列指向形式への変換」を参照してください。

大きい Parquet ブロックサイズまたは ORC ストライプサイズの使用

Parquet と ORC には、調整して最適化できるデータストレージパラメータがあります。Parquet では、ブロックサイズを最適化できます。ORC では、ストライプサイズを最適化できます。ブロックまたはストライプが大きいほど、それぞれに保存できる行が多くなります。デフォルトでは、Parquet のブロックサイズは 128 MB で、ORC のストライプサイズは 64 MB です。

ORC ストライプが 8 MB (`hive.orc.max_buffer_size` のデフォルト値) 未満の場合、Athena は ORC ストライプ全体を読み取ります。これは、ストライプが小さい場合に、列の選択性と 1 秒あたりの入出力操作の間で Athena が行うトレードオフです。

列数が非常に多いテーブルがある場合、ブロックまたはストライプのサイズが小さいと、必要以上に多くのデータがスキャンされる可能性があります。このような場合は、ブロックサイズを大きくするほうが効率的です。

複合型で ORC を使用する

Parquet に保存されている複雑なデータ型 (例: `array`、`map`、`struct`) の列に対してクエリを実行する場合、現行では、Athena は指定された列のみを選択して読み取るのではなく、データの列全体を読み込みます。これは Athena における既知の問題です。回避策として、ORC の使用を検討してください。

圧縮アルゴリズムの選択

設定できるもう 1 つのパラメータは、データブロックの圧縮アルゴリズムです。Athena において、Parquet と ORC でサポートされている圧縮アルゴリズムについては、「Athena 圧縮サポート」を参照してください。

Athena での列指向ストレージ形式の最適化の詳細については、AWS ビッグデータのブログ記事「Amazon Athena のパフォーマンスチューニング Tips トップ 10」の「列指向データストア生成の最適化」セクションを参照してください。

Iceberg テーブル

Apache Iceberg は、Amazon S3 で最適に使用できるように設計された、非常に大規模な分析データセット用のオープンテーブル形式です。Iceberg テーブルを使用すると、Amazon S3 のスロットリングを減らすことができます。

Iceberg テーブルには次のような利点があります。

- Iceberg テーブルは、1 つまたは複数の列でパーティション化できます。これにより、データアクセスが最適化され、クエリでスキャンする必要のあるデータ量が減ります。
- Iceberg オブジェクトストレージモードでは Iceberg テーブルが Amazon S3 と連携するように最適化されるため、大量のデータや大量のクエリワークロードを処理できます。
- オブジェクトストレージモードの Iceberg テーブルは、スケーラブルで耐障害性および耐久性があり、スロットリングを減らすのに役立ちます。
- ACID トランザクションがサポートされているため、複数のユーザーがアトミックな方法で Amazon S3 オブジェクトを追加および削除できます。

Apache Iceberg の詳細については、「Apache Iceberg」を参照してください。Athena で Apache Iceberg テーブルを使用する方法については、「Iceberg テーブルの使用」を参照してください。

クエリの最適化

このセクションでは、Athena の SQL クエリを最適化する方法を提案しています。

LIMIT に ORDER BY 句を使用する

ORDER BY 句は、データをソートされた順序で返します。これには、Athena がすべてのデータ行を 1 つのワーカーノードに送信し、行をソートする必要があります。このタイプのクエリは、長時間実行される場合もあれば、失敗する場合もあります。

クエリの効率を上げるには、上位または下位の *N* 個の値を確認してから、LIMIT 句も使用してください。これにより、ソートと制限の両方を単一のワーカーノードではなく個々のワーカーノードにプッシュすることで、ソートのコストが大幅に削減されます。

JOIN 句の最適化

2 つのテーブルを結合すると、Athena は右側のテーブルをワーカーノードに配布し、左側のテーブルをストリーミングして結合を実行します。

このため、結合の左側には大きいテーブルを指定し、結合の右側に小さいテーブルを指定します。これにより、Athena はメモリの使用量を抑え、クエリの実行待ち時間を短縮できます。

また、以下の点に注意してください。

- 複数の JOIN コマンドを使用する場合は、テーブルを最大のものから最小のものまでの範囲で指定してください。
- クエリで必要でない限り、クロス結合は避けてください。

GROUP BY 句の最適化

GROUP BY オペレータは、GROUP BY 列に基づいて行をワーカーノードに分配します。これらの列はメモリ内で参照され、行が取り込まれると値が比較されます。GROUP BY 列が一致すると、値はまとめて集計されます。この処理の仕組みを考慮すると、列のカーディナリティが最も高いものから低いものの順に並べることをお勧めします。

文字列の代わりに数値を使用する

数値は文字列に比べてメモリ使用量が少なく、処理も速いため、可能な場合は文字列の代わりに数値を使用してください。

列の数を制限する

データを保存するのに必要なメモリの総量を減らすには、SELECT ステートメントで指定する列数を制限してください。

LIKE の代わりに正規表現を使用する

大規模な文字列に LIKE '%string%' 句などを含むクエリを使うと、計算量が非常に多くなる可能性があります。文字列の列で複数の値をフィルタリングする場合は、代わりに regexp_like() 関数や正規表現を使用します。これは、多数の値リストを比較する場合に特に便利です。

LIMIT 句の使用

クエリを実行する際に、すべての列を選択するのではなく、LIMIT 句を使用して必要な列のみを返します。これにより、クエリ実行パイプラインで処理されるデータセットのサイズが小さくなります。LIMIT 句は、文字列ベースの列数が多いテーブルをクエリする場合に便利です。LIMIT 句は、クエリに対して複数の結合や集計を実行するときにも役立ちます。

追加リソース

設計パターンのベストプラクティス: Amazon S3 のパフォーマンスの最適化 (Amazon Simple Storage Service ユーザーガイド) を参考にしてください。

Athena でのパフォーマンスのチューニング

Athena での圧縮のサポート

トピック

Athena は、複数の圧縮形式を使用するテーブルからの読み込みなど、データの読み書きのためのさまざまな圧縮形式をサポートしています。例えば、一部の Parquet ファイルが Snappy で圧縮され

ており、他の Parquet ファイルは GZIP で圧縮されているといった、Parquet ファイル形式を使用するテーブル内のデータも、Athena は正常に読み込むことができます。同様なことが ORC、テキストファイル、および JSON のストレージ形式に対しても当てはまります。

Athena は以下の圧縮形式をサポートしています。

- BZIP2 – Burrows-Wheeler アルゴリズムを使用する形式。
- DEFLATE – LZSS および Huffman コーディングをベースにした圧縮アルゴリズム。DEFLATE が適しているのは Avro ファイル形式のみです。
- GZIP – DEFLATE をベースにした圧縮アルゴリズム。Athena エンジンバージョン 2 および 3 の Hive テーブル、および Athena エンジンバージョン 2 の Iceberg テーブルの場合、GZIP は、Parquet およびテキストファイルストレージ形式のファイルのデフォルトの書き込み圧縮形式です。`tar.gz` 形式を使用するファイルはサポートされていません。
- LZ4 – データの最大圧縮率ではなく、圧縮と解凍速度に焦点を当てた、Lempel-Ziv 77 (LZ7) ファミリーのメンバー。LZ4 には以下のフレーミングフォーマットがあります。
  - LZ4 Raw/Unframed – LZ4 ブロック圧縮形式のフレームを使用しない標準的な実装。詳細については、GitHub の「LZ4 Block Format Description」(LZ4 ブロック形式の説明) を参照してください。
  - LZ4 Framed – LZ4 の一般的なフレーミング実装。詳細については、GitHub の「LZ4 Frame Format Description」(LZ4 フレームフォーマットの説明) を参照してください。
  - LZ4 Hadoop-Compatible – LZ4 の Apache Hadoop 向け実装。この実装は、LZ4 圧縮を BlockCompressorStream.java クラスでラップします。
- LZO – Lempel–Ziv–Oberhumer アルゴリズムを使用する形式。これは、データの最大圧縮率ではなく、圧縮と解凍の高速化に焦点を当てています。LZO には以下の 2 つの実装があります。
  - Standard LZO – 詳細については、Oberhumer ウェブサイトで LZO の「abstract」(抽象化) を参照してください。
  - LZO Hadoop-Compatible – LZO アルゴリズムを BlockCompressorStream.java クラスでラップした実装。
- SNAPPY – Lempel-Ziv 77 (LZ7) ファミリー内の圧縮アルゴリズムの一部。SNAPPY は、データの最大圧縮率ではなく、圧縮と解凍の高速化に焦点を当てています。
- ZLIB – ORC データストレージ形式のファイルで、書き込み圧縮にデフォルトで使用する、DEFLATE をベースにした圧縮形式 詳細については、GitHub の「zlib」ページを参照してください。
- ZSTD – Zstandard real-time data compression algorithm は、高い圧縮率を実現する高速圧縮アルゴリズムです。Zstandard (ZSTD) ライブラリは、BSD ライセンスにより、オープンソースソフト

ウェアとして提供されています。ZSTD は Iceberg テーブル用のデフォルトの圧縮です。ZSTD 圧縮データの書き込み時、Athena はディフォルトで ZSTD 圧縮レベル 3 を使用します。Athena での ZSTD 圧縮レベルの使用の詳細については、「Athena での ZSTD 圧縮レベルの使用」を参照してください。

圧縮形式の指定

CREATE TABLE または CTAS ステートメントを記述する際に圧縮プロパティを指定することで、Athena がこれらのテーブルへの書き込みに使用する圧縮タイプを指定できます。

- CTAS については、「CTAS テーブルのプロパティ」を参照してください。例については、「CTAS クエリの例」を参照してください。
- CREATE TABLE については、「ALTER TABLE SET TBLPROPERTIES」で圧縮テーブルのプロパティ一覧を参照してください。

圧縮なしを指定

CREATE TABLE 文は、非圧縮ファイルの書き込みをサポートします。非圧縮ファイルを書き込むには、次の構文を使用します。

- CREATE TABLE (テキストファイルまたは JSON) – TBLPROPERTIES で、`write.compression = NONE` を指定します。
- CREATE TABLE (Parquet) – TBLPROPERTIES で、`parquet.compression = UNCOMPRESSED` を指定します。
- CREATE TABLE (ORC) – TBLPROPERTIES で、`orc.compress = NONE` を指定します。

注意事項とリソース

- 現在、大文字のファイル拡張子 (`.GZ` または `.BZIP2` など) は、Athena では認識されません。大文字のファイル拡張子を持つデータセットを使用しないようにするか、データファイルの拡張子を小文字に変更します。
- CSV、TSV、および JSON のデータについては、Athena がファイル拡張子から圧縮タイプを判断します。ファイル拡張子がない場合、Athena はデータを非圧縮のプレーンテキストとして扱います。データが圧縮されている場合は、ファイル名に圧縮の拡張子 (gz など) が含まれていることを確認します。
- ZIP ファイル形式はサポートされていません。

- Athena からの Amazon Data Firehose ログのクエリでサポートされる形式には、GZIP 圧縮、または SNAPPY 圧縮を使用する ORC ファイルが含まれます。
- 圧縮の使用の詳細については、「AWS Big·Data·ブログ」の記事「Amazon Athena のパフォーマンスチューニング Tips トップ 10」のセクション 3「ファイルを圧縮· 分割する」 を参照してください。

ファイル形式ごとの Hive テーブルの圧縮サポート

Athena での Hive 圧縮サポートは、エンジンのバージョンに依存します。

Athena エンジンバージョン 3 での Hive 圧縮サポート

次の表に、Apache Hive 内のストレージファイル形式に対して Athena エンジンバージョン 3 でサポートされる、圧縮形式の概要を示します。テキストファイル形式には、TSV、CSV、JSON、およびテキスト用のカスタム SerDes が含まれます。セル内の「はい」または「いいえ」は、特に記載がない限り、読み込みオペレーションと書き込みオペレーションに等しく適用されます。このテーブルでは、CREATE TABLE、CTAS、INSERT INTO は書き込みオペレーションとみなされます。Athena での ZSTD 圧縮レベルの使用の詳細については、「Athena での ZSTD 圧縮レベルの使用」を参照してください。

	Avro	Ion	ORC	Parquet	テキストファイル
BZIP2	はい	はい	いいえ	いいえ	はい
DEFLATE	はい	いいえ	いいえ	いいえ	[いいえ]
GZIP	[いいえ]	はい	いいえ	はい	はい
LZ4	[いいえ]	はい	はい	はい	はい
LZO	[いいえ]	書き込む - いいえ 読み込む - はい	[いいえ]	はい	書き込む - いいえ 読み込む - はい
SNAPPY	はい	はい	はい	はい	はい

	Avro	Ion	ORC	Parquet	テキストファイル
ZLIB	いいえ	いいえ	はい	いいえ	[いいえ]
ZSTD	はい	はい	はい	はい	はい
なし	はい	はい	はい	はい	はい

Athena エンジンバージョン 2 での Hive 圧縮サポート

次の表に、Apache Hive 向け Athena エンジンバージョン 2 でサポートされる、圧縮形式の概要を示します。テキストファイル形式には、TSV、CSV、JSON、およびテキスト用のカスタム SerDes が含まれます。セル内の「はい」または「いいえ」は、特に記載がない限り、読み込みオペレーションと書き込みオペレーションに等しく適用されます。このテーブルでは、CREATE TABLE、CTAS、INSERT INTO は書き込みオペレーションとみなされます。

	Avro	Ion	ORC	Parquet	テキストファイル
BZIP2	はい	はい	いいえ	いいえ	はい
DEFLATE	はい	いいえ	いいえ	いいえ	[いいえ]
GZIP	[いいえ]	はい	いいえ	はい	はい
LZ4	いいえ	いいえ	はい	書き込む - はい 読み込む - いいえ	書き込む - いいえ 読み込む - はい
LZO	[いいえ]	書き込む - いいえ 読み込む - はい	[いいえ]	はい	書き込む - いいえ 読み込む - はい
SNAPPY	はい	はい	はい	はい	はい

	Avro	Ion	ORC	Parquet	テキストファイル
ZLIB	いいえ	いいえ	はい	いいえ	[いいえ]
ZSTD	[いいえ]	はい	はい	はい	はい
なし	はい	はい	はい	はい	はい

ファイル形式別の Iceberg テーブル圧縮サポート

Athena での Apache Iceberg 圧縮サポートは、エンジンのバージョンに依存します。

Athena エンジンバージョン 3 での Iceberg 圧縮サポート

次の表に、Apache Iceberg 内のストレージファイル形式に対して Athena エンジンバージョン 3 でサポートされる、圧縮形式の概要を示します。セル内の「はい」または「いいえ」は、特に記載がない限り、読み込みオペレーションと書き込みオペレーションに等しく適用されます。このテーブルでは、CREATE TABLE、CTAS、INSERT INTO は書き込みオペレーションとみなされます。Athena エンジンバージョン 3 における Iceberg のデフォルトのストレージ形式は Parquet です。Athena エンジンバージョン 3 における Iceberg のデフォルトの圧縮形式は ZSTD です。Athena での ZSTD 圧縮レベルの使用の詳細については、「Athena での ZSTD 圧縮レベルの使用」を参照してください。

	Avro	ORC	Parquet (デフォルト)
BZIP2	いいえ	いいえ	[いいえ]
GZIP	はい	いいえ	はい
LZ4	[いいえ]	はい	いいえ
SNAPPY	はい	はい	はい
ZLIB	[いいえ]	はい	いいえ
ZSTD	はい	はい	はい (デフォルト)
なし	はい (None または `Deflate` を指定)	はい	はい (None または `Uncompressed` を指定)

Athena エンジンバージョン 2 での Iceberg 圧縮サポート

次の表に、Apache Iceberg 向け Athena エンジンバージョン 2 でサポートされる、圧縮形式の概要を示します。セル内の「はい」または「いいえ」は、特に記載がない限り、読み込みオペレーションと書き込みオペレーションに等しく適用されます。このテーブルでは、CREATE TABLE、CTAS、INSERT INTO は書き込みオペレーションとみなされます。Athena エンジンバージョン 2 における Iceberg のデフォルトのストレージ形式は Parquet です。Athena エンジンバージョン 2 における Iceberg のデフォルトの圧縮形式は GZIP です。

	Avro (サポート外)	ORC (サポート外)	Parquet (デフォルト)
BZIP2	いいえ	いいえ	[いいえ]
GZIP	いいえ	[いいえ]	はい (デフォルト)
LZ4	いいえ	いいえ	[いいえ]
SNAPPY	いいえ	いいえ	はい
ZLIB	いいえ	いいえ	[いいえ]
ZSTD	いいえ	いいえ	はい
なし	いいえ	いいえ	はい

Athena での ZSTD 圧縮レベルの使用

Zstandard リアルタイムデータ圧縮アルゴリズムは、高い圧縮率を実現する高速圧縮アルゴリズムです。Zstandard (ZSTD) ライブラリは、オープンソースソフトウェアであり、BSD ライセンスを使用します。Athena は、ZSTD で圧縮された ORC、Parquet、およびテキストファイルでの、データの読み取りと書き込みをサポートしています。

ZSTD 圧縮レベルを使用すると、要件に応じて圧縮率や速度を調整できます。ZSTD ライブラリは、1 から 22 までの圧縮レベルをサポートします。Athena はデフォルトで ZSTD 圧縮レベル 3 を使用しています。

圧縮レベルは、圧縮速度や達成された圧縮量の間の細かいトレードオフを提供します。圧縮レベルが低くなるほど速度は速くなりますが、ファイルサイズは大きくなります。たとえば、速度が最も

重要である場合にはレベル 1 を使用し、サイズが最も重要な場合はレベル 22 を使用できます。レベル 3 は多数のユースケースに適しており、デフォルトです。19 以上のレベルは、より多くのメモリを必要とするため、注意してご使用ください。また、ZSTD ライブラリには、圧縮速度と圧縮率の範囲を広げる負の圧縮レベルも用意されています。詳細については、「Zstandard Compression RFC」(Zstandard 圧縮 RFC) を参照してください。

豊富な圧縮レベルにより、大幅に微調整できる機会が増えます。ただし、圧縮レベルを決定する際には、必ずデータを測定し、そのトレードオフを考慮してください。圧縮速度と圧縮データ サイズの間の合理的なトレードオフのためには、デフォルトレベルである 3 または 6～9 の範囲のレベルを使用することをお勧めします。サイズが最も重要で、圧縮速度が問題にならない場合は、レベル 20 以上を確保します。

考慮事項と制約事項

Athena で ZSTD 圧縮レベルを使用する際は、以下の点を考慮してください。

- ZTSD `compression_level` プロパティは、Athena エンジンバージョン 3 でのみサポートされます。
- ZSTD `compression_level` プロパティは、`ALTER TABLE`、`CREATE TABLE`、`CREATE TABLE AS` (CTAS)、および `UNLOAD` ステートメントでサポートされています。
- `compression_level` プロパティはオプションです。
- `compression_level` プロパティは ZSTD 圧縮でのみサポートされます。
- 可能な圧縮レベルは 1～22 です。
- デフォルトの圧縮レベルは 3 です。

Athena での Apache Hive ZSTD 圧縮サポートの詳細については、「ファイル形式ごとの Hive テーブルの圧縮サポート」を参照してください。Athena での Apache Iceberg ZSTD 圧縮のサポートに関する詳細については、「ファイル形式別の Iceberg テーブル圧縮サポート」を参照してください。

ZSTD 圧縮レベルの指定

`ALTER TABLE`、`CREATE TABLE`、`CREATE TABLE AS`、および `UNLOAD` ステートメントの ZSTD 圧縮レベルを指定するには、`compression_level` プロパティを使用します。ZSTD 圧縮自体を指定するには、ステートメントの構文が使用する個別の圧縮プロパティを使用する必要があります。

ALTER TABLE SET TBLPROPERTIES

ALTER TABLE SET TBLPROPERTIES ステートメントの `SET TBLPROPERTIES` 句では、`'write.compression' = ' ZSTD'` または `'parquet.compression' = 'ZSTD'` を使用

して ZSTD 圧縮を指定します。次に、compression_level プロパティを使用して 1 から 22 までの値を指定します (例: compression_level' = 5)。圧縮レベルプロパティを指定しない場合、圧縮レベルはデフォルトで 3 に設定されます。

例

次の例では、テーブル existing_table を変更して、ZSTD 圧縮および ZSTD 圧縮レベル 4 の Parquet ファイル形式を使用します。圧縮レベルの値は、整数ではなく文字列として入力する必要があることに注意してください。

```
ALTER TABLE existing_table
SET TBLPROPERTIES ('parquet.compression' = 'ZSTD', 'compression_level' = 4)
```

CREATE TABLE

CREATE TABLE ステートメントの TBLPROPERTIES 句では、write.compression' = 'ZSTD' または 'parquet.compression' = 'ZSTD' を指定してから compression_level = *compression_level* を使用して 1 から 22 までの値を指定します。compression_level プロパティを指定しない場合、デフォルトの圧縮レベルは 3 です。

例

次の例では、ZSTD 圧縮と ZSTD 圧縮レベル 4 を使用して Parquet ファイル形式でテーブルを作成します。

```
CREATE EXTERNAL TABLE new_table (
  `col0` string COMMENT '',
  `col1` string COMMENT ''
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
TBLPROPERTIES ('write.compression' = 'ZSTD', 'compression_level' = 4)
```

CREATE TABLE AS (CTAS)

CREATE TABLE AS ステートメントの WITH 句では、write_compression = 'ZSTD' または parquet_compression = 'ZSTD' を指定してから compression_level = *compression_level* を使用して 1 から 22 までの値を指定します。compression_level プロパティを指定しない場合、デフォルトの圧縮レベルは 3 です。

例

次の CTAS の例では、圧縮レベル 4 の ZSTD 圧縮を使用して、Parquet をファイル形式として指定しています。

```
CREATE TABLE new_table
WITH ( format = 'PARQUET', write_compression = 'ZSTD', compression_level = 4)
AS SELECT * FROM old_table
```

UNLOAD

UNLOAD ステートメントの WITH 句では、compression = 'ZSTD' を指定してから compression_level = *compression_level* を使用して 1 から 22 までの値を指定します。compression_level プロパティを指定しない場合、デフォルトの圧縮レベルは 3 です。

例

次の例では、Parquet ファイル形式、ZSTD 圧縮、および ZSTD 圧縮レベル 4 を使用して、クエリ結果を指定された場所にアンロードします。

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format = 'PARQUET', compression = 'ZSTD', compression_level = 4)
```

Athena リソースへのタグ付け

タグは、1 つのキーと 1 つの値で構成されており、どちらもお客様側が定義します。Athena リソースにタグを付けるときは、それにカスタムメタデータを割り当てます。タグを使用すると、AWS リソースを目的、所有者、環境などさまざまな方法で分類することができます。Athena では、ワークグループ、データカタログ、キャパシティ予約などのリソースはタグ付け可能なリソースです。たとえば、アカウントにワークグループの一連のタグを作成して、ワークグループの所有者を追跡したり、目的によってワークグループを識別したりできます。また、Billing and Cost Management コンソールでタグをコスト配分タグとして有効にすると、クエリの実行に関連するコストがそのコスト配分タグとともにコストと使用状況レポートに表示されます。AWS タグ付けのベストプラクティスを使用して、組織の要件に適合する一連の一貫したタグを作成することをお勧めします。

タグでの作業には、Athena コンソール、または API オペレーションを使用できます。

トピック

- タグの基本
- タグの制限
- コンソールでのワークグループのタグの操作
- タグオペレーションの使用
- タグベースの IAM アクセスコントロールポリシー

タグの基本

タグとは、Athena リソースに割り当てるラベルです。タグはそれぞれ、1 つのキーとオプションの 1 つの値で構成されており、どちらもお客様側が定義します。

タグを使用すると、さまざまな方法で AWS リソースを分類できます。たとえば、各ワークグループの所有者または目的を追跡しやすくするため、アカウントのワークグループに対して一連のタグを定義できます。

新しい Athena ワークグループまたはデータカタログを作成するときにタグを追加する、またはそれらに対してタグを追加、編集、または削除することができます。タグは、コンソールで編集できます。API オペレーションを使用してタグを編集するには、古いタグを削除して新しいタグを追加します。リソースを削除すると、リソースのタグも削除されます。

Athena は、リソースに自動でタグを割り当てません。タグのキーと値は編集でき、タグはリソースからいつでも削除できます。タグの値を空の文字列に設定することはできますが、タグの値を null に設定することはできません。同じリソースに重複したタグキーを追加しないでください。そのようにした場合には、Athena はエラーメッセージを発行します。TagResource アクションを使用して、既存のタグキーを使用してリソースにタグを付けると、新しいタグ値によって古い値が上書きされます。

IAM では、Amazon Web Services アカウント内のどのユーザーがタグを作成、編集、削除、またはリストする許可を持つかを制御できます。詳細については、「タグベースの IAM アクセスコントロールポリシー」を参照してください

Amazon Athena タグアクションの完全なリストについては、「Amazon Athena API リファレンス」で API アクション名を参照してください。

タグは請求に使用できます。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「請求へのタグの使用」を参照してください。

詳細については、「タグの制限」を参照してください

タグの制限

タグには、次の制約があります。

- Athena では、ワークグループとデータカタログにタグを付けることができます。クエリにタグを付けることはできません。
- リソースあたりのタグの最大数は 50 です。制限内に収まるようにするには、未使用のタグを確認して削除します。
- 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。同じリソースに重複したタグキーを同時に追加しないでください。そのようにした場合には、Athena はエラーメッセージを発行します。別の `TagResource` アクションで既存のタグキーを使用してリソースにタグを付けると、新しいタグ値によって古い値が上書きされます。
- タグキーの長さは、1 ~ 128 文字 (Unicode) (UTF-8) です。
- タグ値の長さは、0 ~ 256 文字 (Unicode) (UTF-8) です。

  タグの追加、編集、削除、一覧表示などのタグ付けオペレーションでは、ワークグループリソースの ARN を指定する必要があります。
- Athena では、文字、数字、UTF-8 で表現されたスペース、および + - = . _ : / @ の文字を使用できます。
- タグのキーと値は大文字と小文字が区別されます。
- タグキーの `"aws:"` プレフィックスは、AWS 用に予約されています。このプレフィックスが含まれるタグキーを編集したり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時には計算されません。
- 割り当てたタグは、お客様の Amazon Web Services アカウントだけに使用できます。

コンソールでのワークグループのタグの操作

Athena コンソールを使用すると、アカウントの各ワークグループでどのタグが使用されているかを確認できます。ワークグループでのみタグを表示できます。また、Athena コンソールを使用して、一度に 1 つのワークグループに対してタグの適用、編集、または削除を行うことができます。

作成したタグを使用してワークグループを検索できます。

トピック

- 個々のワークグループのタグを表示する

- 個々のワークグループでのタグの追加と削除

個々のワークグループのタグを表示する

Athena コンソールで個々のワークグループのタグを表示するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. ナビゲーションメニューで、[Workgroups] (ワークグループ) 選択し、ワークグループを選択します。
4. 次のいずれかを行います。
   - [タグ] タブを選択します。タグのリストが長い場合は、検索ボックスを使用します。
   - [Edit] (編集) を選択し、[Tags] (タグ) セクションまで下にスクロールします。

個々のワークグループでのタグの追加と削除

[Workgroups (ワークグループ)] タブから、個々のワークグループのタグを直接管理できます。

> **Note**
> ユーザーがコンソールでワークグループを作成するときにタグを追加する、または CreateWorkGroup アクションを使用するときにタグを渡すことができるようにするには、そ

のユーザーに TagResource アクションと CreateWorkGroup アクションへの IAM 許可を付与するようにしてください。

新しいワークグループを作成する際にタグを追加するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. ナビゲーションメニューで、[Workgroups] (ワークグループ) を選択します。
3. [Create workgroup] (ワークグループを作成する) をクリックして、必要に応じて値を入力します。詳細なステップについては、「ワークグループの作成」を参照してください。
4. [Tags] (タグ) セクションで、キーと値を指定して、1 つ以上のタグを追加します。同じワークグループに重複したタグキーを同時に追加しないでください。そのようにした場合には、Athena はエラーメッセージを発行します。詳細については、「タグの制限」を参照してください
5. 終了したら、[Create workgroup] (ワークグループを作成する) をクリックします。

既存のワークグループにタグを追加または編集する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. ナビゲーションペインで、[Global networks (グローバルネットワーク)] を選択します。
3. 変更するワークグループを選択します。
4. 次のいずれかを行います。
   - [タグ] タブを選択し、[タグ管理] を選択します。
   - [Edit] (編集) を選択し、[Tags] (タグ) セクションまで下にスクロールします。
5. タグごとにキーと値を指定します。詳細については、「タグの制限」を参照してください。
6. [Save] を選択します。

個々のワークグループからタグを削除する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. ナビゲーションペインで、[Global networks (グローバルネットワーク)] を選択します。
3. 変更するワークグループを選択します。
4. 次のいずれかを行います。
   - [タグ] タブを選択し、[タグ管理] を選択します。

- [Edit] (編集) を選択し、[Tags] (タグ) セクションまで下にスクロールします。

5. タグのリストで、削除するタグの [Remove] (削除) ボタンをクリックし、[Save] (保存) を選択します。

タグオペレーションの使用

次のタグオペレーションを使用して、リソースのタグを追加、削除、または一覧表示します。

API	CLI	アクションの説明
TagResource	tag-resource	指定した ARN を持つリソースの 1 つ以上のタグを追加または上書きします。
UntagResource	untag-resource	指定した ARN を持つリソースから 1 つ以上のタグを削除します。
ListTagsForResource	list-tags-for-resource	指定した ARN を持つリソースの 1 つ以上のタグを一覧表示します。

リソースの作成時のタグの追加

ワークグループまたはデータカタログの作成時にタグを追加するには、tags パラメータを CreateWorkGroup または CreateDataCatalog API オペレーション、あるいは AWS CLI の create-work-group または create-data-catalog コマンドで使用します。

API オペレーションを使用したタグの管理

このセクションの例では、タグ API オペレーションを使用してワークグループとデータカタログのタグを管理する方法を示します。例は、Java プログラミング言語で記述されています。

Example TagResource

次の例では、ワークグループ workgroupA に 2 つのタグを追加します。

```
List<Tag> tags = new ArrayList<>();
tags.add(new Tag().withKey("tagKey1").withValue("tagValue1"));
```

```
tags.add(new Tag().withKey("tagKey2").withValue("tagValue2"));

TagResourceRequest request = new TagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA")
    .withTags(tags);

client.tagResource(request);
```

次の例では、データカタログ datacatalogA に 2 つのタグを追加します。

```
List<Tag> tags = new ArrayList<>();
tags.add(new Tag().withKey("tagKey1").withValue("tagValue1"));
tags.add(new Tag().withKey("tagKey2").withValue("tagValue2"));

TagResourceRequest request = new TagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA")
    .withTags(tags);

client.tagResource(request);
```

> **Note**
>
> 同じリソースに重複したタグキーを追加しないでください。そのようにした場合には、Athena はエラーメッセージを発行します。別の TagResource アクションで既存のタグキーを使用してリソースにタグを付けると、新しいタグ値によって古い値が上書きされます。

Example UntagResource

次の例では、ワークグループ tagKey2 から workgroupA を削除します。

```
List<String> tagKeys = new ArrayList<>();
tagKeys.add("tagKey2");

UntagResourceRequest request = new UntagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA")
    .withTagKeys(tagKeys);

client.untagResource(request);
```

次の例では、データカタログ tagKey2 から datacatalogA を削除します。

```
List<String> tagKeys = new ArrayList<>();
tagKeys.add("tagKey2");

UntagResourceRequest request = new UntagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA")
    .withTagKeys(tagKeys);

client.untagResource(request);
```

Example ListTagsForResource

次の例では、ワークグループ workgroupA のタグを一覧表示します。

```
ListTagsForResourceRequest request = new ListTagsForResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA");

ListTagsForResourceResult result = client.listTagsForResource(request);

List<Tag> resultTags = result.getTags();
```

次の例では、データカタログ datacatalogA のタグを一覧表示します。

```
ListTagsForResourceRequest request = new ListTagsForResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA");

ListTagsForResourceResult result = client.listTagsForResource(request);

List<Tag> resultTags = result.getTags();
```

AWS CLI を使用したタグの管理

以下のセクションでは、AWS CLI を使用してデータカタログのタグを作成および管理する方法を示します。

リソースへのタグの追加: Tag-resource

tag-resource コマンドは、指定されたリソースに 1 つ以上のタグを追加します。

Syntax

```
aws athena tag-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name --tags
Key=string,Value=string Key=string,Value=string
```

--resource-arn パラメータは、タグを追加するリソースを指定します。--tags パラメータは、リソースにタグとして追加するスペース区切りのキーと値のペアのリストを指定します。

Example

次の例では、mydatacatalog データカタログにタグを追加します。

```
aws athena tag-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog --tags Key=Color,Value=Orange
 Key=Time,Value=Now
```

結果を表示するには、list-tags-for-resource コマンドを使用します。

create-data-catalog コマンドを使用する際にタグを追加する方法については、「カタログの登録: Create-data-catalog」(カタログを登録する: Create-data-catalog) を参照してください。

リソースのタグの一覧表示: List-tags-for-resource

list-tags-for-resource コマンドは、指定されたリソースのタグを一覧表示します。

Syntax

```
aws athena list-tags-for-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name
```

--resource-arn パラメータは、タグを一覧表示するリソースを指定します。

次の例では、mydatacatalog データカタログのタグを一覧表示します。

```
aws athena list-tags-for-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog
```

次のサンプル結果は JSON 形式です。

```
{
    "Tags": [
        {
            "Key": "Time",
            "Value": "Now"
```

```
        },
        {
            "Key": "Color",
            "Value": "Orange"
        }
    ]
}
```

リソースからのタグの削除: Untag-resource

untag-resource コマンドは、指定されたタグキーとその関連値を、指定されたリソースから削除します。

Syntax

```
aws athena untag-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name --tag-keys
key_name [key_name ...]
```

--resource-arn パラメータは、タグを削除するリソースを指定します。--tag-keys パラメータは、キー名のスペース区切りリストを取ります。untag-resource コマンドは、指定されたキー名ごとに、キーとその値の両方を削除します。

次の例では、Color カタログリソースから Time および mydatacatalog キーとそれぞれの値を削除します。

```
aws athena untag-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog --tag-keys Color Time
```

タグベースの IAM アクセスコントロールポリシー

タグがあると、リソースのタグに基づいてそのリソースへのアクセスを制御する Condition ブロックが含まれた IAM ポリシーを作成できます。

ワークグループのタグポリシーの例

Example 1. 基本的なタグ付けポリシー

以下の IAM ポリシーは、workgroupA という名前のワークグループのタグでクエリを実行し、それらとやり取りすることを可能にします。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
        }
    ]
```

```
}
```

Example 2: タグキーとタグ値のペアに基づいてワークグループに対するアクションを拒否するポリシーブロック

ワークグループなどのリソースに関連付けられたタグは、リソースタグと呼ばれます。リソースタグを使用すると、stack、production などのキーと値のペアでタグ付けされたワークグループに対してリストされているアクションを拒否する、次のようなポリシーブロックを作成できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "athena:GetWorkGroup",
                "athena:UpdateWorkGroup",
                "athena:DeleteWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stack": "production"
```

```
                }
            }
        }
    ]
}
```

Example 3. タグ変更アクションリクエストを指定のタグに制限するポリシーブロック

タグを変更するオペレーション (TagResource、UntagResource、タグを指定した CreateWorkGroup など) にパラメータとして渡されるタグは、リクエストタグと呼ばれます。次のポリシーブロックの例では、渡されたタグのいずれかがキー CreateWorkGroup と値 costcenter、1、または 2 を持つ場合にのみ、3 オペレーションを許可します。

> **Note**
>
> CreateWorkGroup オペレーションの一環としてタグを渡すことを IAM ロールに許可する場合は、そのロールに TagResource アクションと CreateWorkGroup アクションへの許可を付与するようにしてください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateWorkGroup",
                "athena:TagResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/costcenter": [
                        "1",
                        "2",
                        "3"
                    ]
                }
            }
        }
    ]
```

```
}
```

データカタログのタグポリシーの例

Example 1. 基本的なタグ付けポリシー

以下の IAM ポリシーは、datacatalogA という名前のデータカタログのタグとやり取りを可能にします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
```

```
                "athena:DeleteNamedQuery"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource"
            ],
            "Resource": "arn:aws:athena:us-
east-1:123456789012:datacatalog/datacatalogA"
        }
    ]
}
```

Example 2: タグキーとタグ値のペアに基づいてデータカタログに対するアクションを拒否するポリシーブロック

リソースタグを使用すると、特定のタグのキーと値のペアでタグ付けされたデータカタログに対して特定のアクションを拒否するポリシーブロックを作成できます。次のポリシーの例では、タグのキーと値のペア stack、production を持つデータカタログに対するアクションを拒否します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:UpdateDataCatalog",
```

```
                "athena:DeleteDataCatalog",
                "athena:GetDatabase",
                "athena:ListDatabases",
                "athena:GetTableMetadata",
                "athena:ListTableMetadata",
                "athena:StartQueryExecution",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stack": "production"
                }
            }
        }
    ]
}
```

Example 3. タグ変更アクションリクエストを指定のタグに制限するポリシーブロック

タグを変更するオペレーション (TagResource、UntagResource、タグを指定した CreateDataCatalog など) にパラメータとして渡されるタグは、リクエストタグと呼ばれます。次のポリシーブロックの例では、渡されたタグのいずれかがキー CreateDataCatalog と値 costcenter、1、または 2 を持つ場合にのみ、3 オペレーションを許可します。

> **Note**
>
> CreateDataCatalog オペレーションの一環としてタグを渡すことを IAM ロールに許可する場合は、そのロールに TagResource アクションと CreateDataCatalog アクションへの許可を付与するようにしてください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:TagResource"
```

```
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/costcenter": [
                        "1",
                        "2",
                        "3"
                    ]
                }
            }
        }
    ]
}
```

Service Quotas

> **Note**
>
> Service Quotas コンソールには、Amazon Athena のクォータに関する情報が提供されています。Service Quotas コンソールを使用して、調整可能なクォータに関してクォータの増加をリクエストすることもできます。スキーマの制限に関する AWS Glue の詳細は、「AWS Glue エンドポイントとクォータ」のページを参照してください。AWS サービスクォータの概要については、「AWS 全般のリファレンス」の「AWS サービスクォータ」を参照してください。

クエリ

アカウントには、Amazon Athena のクエリに関し以下のクォータが設定されています。詳細については、AWS 全般のリファレンス の「Amazon Athena エンドポイントとクォータ」ページを参照してください。

- アクティブな DDL クエリ – アクティブな DDL クエリの数。DDL クエリには、`CREATE TABLE` クエリと `ALTER TABLE ADD PARTITION` クエリが含まれます。
- DDL クエリのタイムアウト – DDL クエリが、キャンセルされずに実行可能な最大時間 (分単位)。
- アクティブな DML クエリ – アクティブな DML クエリの数。DML クエリには、`SELECT`、`CREATE TABLE AS` (CTAS)、および `INSERT INTO` クエリが含まれます。特定のクォータは、AWS リージョンによって変化します。

- DML クエリのタイムアウト – DML クエリが、キャンセルされずに実行可能な最大時間 (分単位)。このタイムアウトは、最大 240 分まで延長するようリクエストできます。

クォータの引き上げをリクエストするには、Athena Service Quotas コンソールを使用できます。

Athena は、全体的なサービス負荷と受信するリクエストの数に基づいてリソースを割り当てることによって、クエリを処理します。クエリは、一時的にキューに登録されてから実行される場合があります。アカウント設定によって許可されている場合に限り、非同期プロセスが物理リソースが使用可能になり次第キューからクエリを取り出して、それらを物理リソース上で実行します。

DML または DDL クエリのクォータには、実行中のクエリーとキューに登録されたクエリの両方が含まれます。例えば、DML クエリのクォータが 25 で、実行中のクエリとキューに登録されたクエリの合計が 26 になった場合、26 番目のクエリは TooManyRequestsException エラーを引き起こします。

> **Note**
>
> Athena で実行するクエリに対して同時実行を直接制御したい場合は、キャパシティ予約を使用できます。詳細については、「クエリ処理キャパシティの管理」を参照してください。

クエリ文字列の長さ

許容される最大クエリ文字列の長さは、262144 バイトです。文字列は UTF-8 でエンコードされます。これは調整可能なクォータではありません。ただし、長いクエリを複数の小さなクエリに分割することで、この制限を回避することができます。詳細については、AWS ナレッジセンターの「Athena でクエリ文字列の最大長を増やすにはどうすればよいですか?」を参照してください。

ワークグループ

Athena ワークグループを使用する場合は、以下の点に注意してください。

- Athena のサービスクォータは、アカウント内のすべてのワークグループで共有されます。
- アカウントでリージョンごとに作成できるワークグループの最大数は 1000 です。
- ワークグループ内の準備済みステートメントの最大数は 1,000 です。
- ワークグループあたりのタグの最大数は 50 です。詳細については、「タグの制限」を参照してください。

データベース、テーブル、パーティション

- Athena で AWS Glue Data Catalog を使用している場合、テーブル、データベース、およびパーティションのサービスクォータ (アカウントあたりのデータベースまたはテーブルの最大数など) については、「AWS Glue エンドポイントとクォータ」を参照してください。
  - Athena では、1,000 万のパーティションを持つ AWS Glue テーブルへのクエリがサポートされていますが、1 回のスキャンで読み取れるのは、100 万のパーティションまでです。
- AWS Glue Data Catalog を使用しない場合、テーブルあたりのパーティション数は 20,000 です。クォータの引き上げをリクエストできます。

Amazon S3 バケット

Amazon S3 バケットを使用する場合は、以下の点に注意してください。

- Amazon S3 には、アカウントあたり 100 バケットのデフォルトサービスクォータがあります。
- Athena では、結果を口グするための個別のバケットが必要です。
- AWS アカウントごとに Amazon S3 バケットのクォータ引き上げを最大 1,000 個までリクエストできます。

アカウントあたりの API コールのクォータ

Athena API では、アカウントあたり (クエリあたりではありません) の API へのコール数に、以下のデフォルトクォータがあります。

API 名	1 秒あたりのデフォルトコール数	バーストキャパシティ
`BatchGetNamedQuery` , `ListNamedQueries` , `ListQueryExecutions`	5	最大 10
`CreateNamedQuery` , `DeleteNamedQuery` , `GetNamedQuery`	5	最大 20
`BatchGetQueryExecution`	20	最大 40
`StartQueryExecution` , `StopQueryExecution`	20	最大 80

API 名	1 秒あたりのデフォルトコール数	バーストキャパシティ
`GetQueryExecution` , `GetQueryResults`	100	最大 200

例えば、`StartQueryExecution` の場合は、1 秒あたり最大 20 回呼び出すことができます。さらに、API が 4 秒間呼び出しを行わなかった場合、アカウントのバーストキャパシティーが最大 80 回まで累積されます。この場合、アプリケーションはバーストモードでこの API を最大 80 回呼び出すことができます。

これらの API のいずれかを使用しており、1 秒あたりのコール数のデフォルトクォータ、またはアカウントのバーストキャパシティを超過した場合、Athena API は「ClientError: An error occurred (ThrottlingException) when calling the <API_name> operation: Rate exceeded.」に類似したエラーを発行します。1 秒あたりの呼び出し数、またはこのアカウントの API のバーストキャパシティーを減らしてください。

アカウント API 呼び出しごとの Athena クォータは、Athena Service Quotas コンソールでは変更できません。Athena API 呼び出しのクォータ引き上げをリクエストするには、AWS Support サービスの上限緩和のページに移動し、フォームに記入して送信します。

Athena エンジンのバージョニング

Athena は、パフォーマンスと機能の改善、およびコードの修正を反映するために、新しいエンジンバージョンをリリースすることがあります。新しいエンジンバージョンが利用可能になると、Athena がコンソールと AWS Health Dashboard で通知を行います。AWS Health Dashboard は、AWS のサービス、またはアカウントに影響を及ぼす可能性があるイベントについて通知します。AWS Health Dashboard の詳細については、「AWS Health Dashboard の使用開始」を参照してください。

エンジンのバージョン管理は、ワークグループごとに設定されます。ワークグループを使用して、クエリが使用するクエリエンジンと、Athena がワークグループを自動的にアップグレードできるかどうかを制御できます。使用中のクエリエンジンは、クエリエディタとワークグループの詳細ページに表示され、Athena API を使用して取得することもできます。

- デフォルトで、ワークグループは自動アップグレードするように設定されています。自動アップグレードするようにワークグループを設定すると、非互換性を検出した場合を除き、Athena がワークグループをアップグレードします。

- 特定のバージョンを使用するようにワークグループを設定すると、Athena はワークグループのバージョンを変更しません。

どちらの場合も、Athena はバージョンが利用できなくなった時点でワークグループをアップグレードします。Athena は、エンジンバージョンが提供されなくなる時期に関する通知を、AWS Health Dashboard 経由で行います。AWS Health Dashboard は、AWS のサービス、またはアカウントに影響を及ぼす可能性があるイベントについて通知します。AWS Health Dashboard の詳細については、「AWS Health Dashboard の使用開始」を参照してください。

新しいエンジンバージョンの使用を開始すると、非互換が原因でクエリの小さなサブセットが破損する場合があります。下位互換性のない変更は、新しい Athena バージョンのリリース時に発表されます。新しいエンジンを使用するテストワークグループを作成する、または既存のワークグループを試験的にアップグレードすることによって、アップグレード前にワークグループを使用してクエリをテストするようにしてください。詳細については、「エンジンバージョンをアップグレードする前のクエリのテスト」を参照してください。

トピック

- Athena エンジンバージョンの変更
- Athena エンジンバージョンリファレンス

Athena エンジンバージョンの変更

Athena は、パフォーマンスと機能の改善、およびコードの修正を反映するために、新しいエンジンバージョンをリリースすることがあります。新しいエンジンバージョンが利用可能になると、Athena はコンソールで通知を行います。ユーザーは、アップグレードするタイミングを Athena に決定させる、またはワークグループごとに Athena エンジンのバージョンを手動で指定する選択ができます。

トピック

- ワークグループのクエリエンジンバージョンの確認
- Athena コンソールでのエンジンバージョンの変更
- AWS CLI を使用したエンジンバージョンの変更
- ワークグループの作成時におけるエンジンバージョンの指定
- エンジンバージョンをアップグレードする前のクエリのテスト
- 失敗するクエリのトラブルシューティング

ワークグループのクエリエンジンバージョンの確認

[Workgroups] (ワークグループ) ページから、任意のワークグループの現在のエンジンバージョンを確認することができます。

ワークグループの現在のエンジンバージョンを確認する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
4. [Workgroups] (ワークグループ) ページで、目的のワークグループを探します。ワークグループの [Query engine version] (クエリエンジンのバージョン) 列には、クエリエンジンのバージョンが表示されています。

Athena コンソールでのエンジンバージョンの変更

新しいエンジンバージョンが利用可能になると、アップグレードするタイミングを Athena に決定させる、またはワークグループが使用する Athena のエンジンバージョンを手動で指定することを選択できます。使用できるバージョンが 1 つしかない場合は、別のバージョンを指定できません。

> **Note**
>
> ワークグループのエンジンバージョンを変更するには、ワークグループで `athena:ListEngineVersions` アクションを実行する許可が必要です。IAM ポリシーの例については、「ワークグループのポリシーの例」を参照してください。

Athena にワークグループをアップグレードするタイミングを決定させる

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
4. ワークグループのリストで、設定するワークグループへのリンクを選択します。
5. [編集] を選択します。
6. [Query engine version] (クエリエンジンのバージョン) セクションにある [Update query engine] (クエリエンジンを更新) で、[Automatic] (自動) を選択し、ワークグループのアップグレードタイミングを Athena が選択できるようにします。これはデフォルトの設定です。
7. [Save changes] (変更の保存) をクリックします。

   ワークグループのリストでは、ワークグループの [Query engine update status] (クエリエンジンの更新ステータス) に [Automatic] (自動) と表示されます。

エンジンのバージョンを手動で選択する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
4. ワークグループのリストで、設定するワークグループへのリンクを選択します。
5. [編集] を選択します。
6. 手動でエンジンバージョンを選択するには、[Query engine version] (クエリエンジンのバージョン) セクションにある [Update query engine] (クエリエンジンを更新) で、[Manual] (手動) を選択します。

7. ワークグループで使用するエンジンのバージョンを自分で選択する場合は、[Query engine version] (クエリエンジンのバージョン) オプションを使用します。他に使用できるエンジンバージョンがない場合は、別のエンジンバージョンを指定することはできません。
8. [Save changes] (変更の保存) をクリックします。

   ワークグループのリストでは、ワークグループの [Query engine update status] (クエリエンジンの更新ステータス) に [Manual] (手動) と表示されます。

AWS CLI を使用したエンジンバージョンの変更

AWS CLI を使用してエンジンバージョンを変更するには、次の例にある構文を使用します。

```
aws athena update-work-group --work-group workgroup-name --configuration-updates
 EngineVersion={SelectedEngineVersion='Athena engine version 3'}
```

ワークグループの作成時におけるエンジンバージョンの指定

ワークグループを作成するときは、ワークグループが使用するエンジンバージョンを指定する、または Athena にワークグループをアップグレードするタイミングを決定させることができます。利用可能な新しいエンジンバージョンがある場合のベストプラクティスは、他のワークグループをアップグレードする前に、別にワークグループを作成して新しいエンジンのテストを行うことです。ワークグループのエンジンバージョンを指定するには、ワークグループに対する `athena:ListEngineVersions` 許可が必要です。IAM ポリシーの例については、「ワークグループのポリシーの例」を参照してください。

ワークグループの作成時にエンジンバージョンを指定する

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
4. [Workgroups] (ワークグループ) ページで、[Create workgroup] (ワークグループを作成する) をクリックします。
5. [Create workgroup] (ワークグループの作成) ページにある [Query engine version] (クエリエンジンのバージョン) セクションで、以下のいずれかを実行します。

- ワークグループをアップグレードするタイミングを Athena に選択させる場合は、[Automatic] (自動) を選択します。これはデフォルトの設定です。
- 別の使用可能なエンジンバージョンを手動で選択する場合は、[Manual] (手動) を選択します。

6. 必要に応じて、他のフィールドに情報を入力します。他のフィールドについては、「ワークグループの作成」を参照してください。
7. [Create workgroup] (ワークグループの作成) を選択します。

エンジンバージョンをアップグレードする前のクエリのテスト

ワークグループが新しいエンジンバージョンにアップグレードされると、互換性の問題が原因となり一部のクエリが破損する場合があります。エンジンバージョンのアップグレードがスムーズに行われることを確実にするには、事前にクエリをテストすることができます。

エンジンバージョンのアップグレード前にクエリをテストする

1. 使用しているワークグループのエンジンバージョンを確認します。使用しているエンジンのバージョンが、[Workgroups] (ワークグループ) ページで、対象のワークグループの [Query engine version] (クエリエンジンのバージョン) 列に表示されます。詳細については、「ワークグループのクエリエンジンバージョンの確認」を参照してください。
2. 新しいエンジンバージョンを使用するテストワークグループを作成します。詳細については、「」を参照してくださいワークグループの作成時におけるエンジンバージョンの指定
3. 新しいワークグループを使用して、テストするクエリを実行します。
4. クエリが失敗した場合は、「Athena エンジンバージョンリファレンス」を使用して、クエリに影響を及ぼしている可能性がある破壊的変更をチェックします。変更には、クエリ構文の更新が必要になるものもあります。
5. それでもクエリが失敗する場合は、AWS Support に連絡してサポートを受けてください。Adding a change AWS Management Console で、[Support] (サポート)、[Support Center] (サポートセンター) の順にクリックします。または、[Amazon Athena] タグを使用して AWS re:Post に質問します。

失敗するクエリのトラブルシューティング

エンジンバージョンのアップグレード後にクエリが失敗した場合は、「Athena エンジンバージョンリファレンス」を使用して、クエリの構文に影響を及ぼす可能性がある変更を含めた破壊的変更をチェックしてください。

それでもクエリが失敗する場合は、AWS Support に連絡してサポートを受けてください。AWS Management Console で、[Support] (サポート)、[Support Center] (サポートセンター) の順にクリックします。または、[Amazon Athena] タグを使用して AWS re:Post に質問します。

Athena エンジンバージョンリファレンス

このセクションでは、Athena クエリエンジンに対する変更を一覧表示します。

トピック

- Athena エンジンバージョン 3
- Athena エンジンバージョン 2

Athena エンジンバージョン 3

エンジンバージョン 3 では、Athena はオープンソースのソフトウェア管理に継続的な統合アプローチを導入しました。これにより、Trino および Presto プロジェクトでの同時実行性が向上するため、Athena エンジン内で統合およびチューニングされたコミュニティの改善にすばやくアクセスできるようになります。

Athena エンジンバージョン 3 のこのリリースでは、Athena エンジンバージョン 2 のすべての機能がサポートされています。このドキュメントでは、Athena エンジンバージョン 2 と Athena エンジンバージョン 3 の主な違いについて説明します。詳細については、AWS Big Data Blog の記事、「Upgrade to Athena engine version 3 to increase query performance and access more analytics features」を参照してください。

- 開始
- 改善点と新機能
  - 追加された機能
  - 追加された関数
  - フォーマンスの向上
  - 信頼性の強化

開始

はじめに、Athena エンジンバージョン 3 を使用する新しい Athena ワークグループを作成するか、バージョン 3 を使用するように既存のワークグループを設定するかを選択します。Athena のどのワークグループでも、クエリの送信を中断することなく、エンジンバージョン 2 からエンジンバージョン 3 にアップグレードできます。

詳細については、「Athena エンジンバージョンの変更」を参照してください。

改善点と新機能

記載されている機能や更新には、Athena 自体の改善点と、オープンソースの Trino から組み込まれた機能性の改善点が含まれています。SQL クエリの演算子および関数を網羅するリストについては、「Trino documentation」(Trino ドキュメント) を参照してください。

追加された機能

Apache Spark バケットアルゴリズムのサポート

Athena は Spark ハッシュアルゴリズムによって生成されたバケットを読み込むことができます。データが元々 Spark ハッシュアルゴリズムによって書き込まれたものであることを指定するには、`CREATE TABLE` ステートメントの `TBLPROPERTIES` 句に `('bucketing_format'='spark')` を入れます。このプロパティが指定されていない場合は、Hive ハッシュアルゴリズムが使用されます。

```
CREATE EXTERNAL TABLE `spark_bucket_table`(
  `id` int,
  `name` string
  )
CLUSTERED BY (`name`)
INTO 8 BUCKETS
```

```
STORED AS PARQUET
LOCATION
  's3://DOC-EXAMPLE-BUCKET/to/bucketed/table/'
TBLPROPERTIES ('bucketing_format'='spark')
```

追加された関数

このセクションの関数は、Athena エンジンバージョン 3 で新たに追加されました。

集計関数

listagg(x, separator) – 連結された入力値を区切り文字列で区切って返します。

```
SELECT listagg(value, ',') WITHIN GROUP (ORDER BY value) csv_value
FROM (VALUES 'a', 'c', 'b') t(value);
```

配列関数

contains_sequence(x, seq) – 配列 x にすべての配列シーケンスが (すべての値が同じ順序で) シーケンシャルサブセットとして含まれている場合、true を返します。

```
SELECT contains_sequence(ARRAY [1,2,3,4,5,6], ARRAY[1,2]);
```

バイナリ関数

murmur3(binary) – バイナリの 128 ビット MurmurHash3 ハッシュを計算します。

```
SELECT murmur3(from_base64('aaaaaa'));
```

変換関数

format_number(number) – 単位記号を使用してフォーマットされた文字列を返します。

```
SELECT format_number(123456); -- '123K'
```

```
SELECT format_number(1000000); -- '1M'
```

日付および時刻関数

timezone_hour(timestamp) – タイムスタンプからオフセットされたタイムゾーンの時間を返します。

```
SELECT EXTRACT(TIMEZONE_HOUR FROM TIMESTAMP '2020-05-10 12:34:56 +08:35');
```

timezone_minute(timestamp) – タイムスタンプからオフセットされたタイムゾーンの分数を返します。

```
SELECT EXTRACT(TIMEZONE_MINUTE FROM TIMESTAMP '2020-05-10 12:34:56 +08:35');
```

地理空間関数

to_encoded_polyline(Geometry) – ライン文字列またはマルチポイントをポリラインにエンコードします。

```
SELECT to_encoded_polyline(ST_GeometryFromText(
   'LINESTRING (-120.2 38.5, -120.95 40.7, -126.453 43.252)'));
```

from_encoded_polyline(varchar) – ポリラインをライン文字列にデコードします。

```
SELECT ST_AsText(from_encoded_polyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
```

to_geojson_geometry(SphericalGeography) – 指定された球面ジオグラフィを GeoJSON 形式で返します。

```
SELECT to_geojson_geometry(to_spherical_geography(ST_GeometryFromText(
   'LINESTRING (0 0, 1 2, 3 4)')));
```

from_geojson_geometry(varchar) – 非ジオメトリのキー/値を取り除いて、GeoJSON 表現から球面ジオグラフィ型のオブジェクトを返します。`Feature` および `FeatureCollection` はサポートされていません。

```
SELECT
 from_geojson_geometry(to_geojson_geometry(to_spherical_geography(ST_GeometryFromText(
   'LINESTRING (0 0, 1 2, 3 4)'))));
```

geometry_nearest_points(Geometry, Geometry) – 各ジオメトリ上の互いに最も近いポイントを返します。いずれかのジオメトリが空の場合は NULL を返します。それ以外の場合は、ジオメトリ上にある任意の 2 つのポイント間の距離が最小となる 2 つの `Point` オブジェクトを含む行を返します。最初のポイントは Geometry の 1 番目の引数から、2 番目のポイントは Geometry の 2 番目の引数からのものです。最小距離が同じペアが複数ある場合は、任意に 1 つのペアが選択されます。

```
SELECT geometry_nearest_points(ST_GeometryFromText(
   'LINESTRING (50 100, 50 200)'), ST_GeometryFromText(
   'LINESTRING (10 10, 20 20)'));
```

ダイジェスト関数の設定

make_set_digest(x) – x のすべての入力値を setdigest にまとめます。

```
SELECT make_set_digest(value) FROM (VALUES 1, 2, 3) T(value);
```

文字列関数

soundex(char) – char の発音表現を含む文字列を返します。

```
SELECT name
FROM nation
WHERE SOUNDEX(name) = SOUNDEX('CHYNA'); -- CHINA
```

concat_ws(string0, string1,..., stringN) –`string0` を区切り文字として使用し、`string1, string2, ..., stringN` の結果を返します。`string0` が null である場合、戻り値は null です。区切り文字の後の引数に指定された null 値は、すべてスキップされます。

```
SELECT concat_ws(',', 'def', 'pqr', 'mno');
```

Window 関数

GROUPS – グループに基づくウィンドウフレームのサポートが追加されました。

```
SELECT array_agg(a) OVER(
   ORDER BY a ASC NULLS FIRST GROUPS BETWEEN 1 PRECEDING AND 2 FOLLOWING)
FROM (VALUES 3, 3, 3, 2, 2, 1, null, null) T(a);
```

フォーマンスの向上

Athena エンジンバージョン 3 でのパフォーマンスの向上には、次のものが含まれています。

- AWS Glue テーブルメタデータの取得の高速化 - 複数のテーブルを含むクエリでは、クエリ計画時間を短縮できます。
- RIGHT JOIN の動的フィルタリング – 次の例のように、等価結合条件を持つ右結合に対して動的フィルタリングが有効になりました。

```
SELECT *
FROM lineitem RIGHT JOIN tpch.tiny.supplier
ON lineitem.suppkey = supplier.suppkey
WHERE supplier.name = 'abc';
```

- 大きいサイズのプリペアドステートメント - デフォルトの HTTP リクエスト/レスポンスヘッダーのサイズを 2 MB に増やし、大きいサイズのプリペアドステートメントを使用できるようにしました。
- approx_percentile() — この `approx_percentile` 関数は、分布から近似分位値を取得するために `qdigest` の代わりに `tdigest` を使用するようになりました。その結果、パフォーマンスが向上し、メモリ使用量が少なくなります。この変更により、この関数は Athena エンジンバージョン 2 とは異なる結果を返しますので、ご注意してください。詳細については、「prox_percentile 関数は異なる結果を返します。」を参照してください。

信頼性の強化

Athena エンジンバージョン 3 の一般的なエンジンメモリ使用量および追跡が改善されました。クエリの規模が大きい場合でも、ノードクラッシュによる障害の影響を受けにくくなります。

クエリ構文の機能強化

INTERSECT ALL – `INTERSECT ALL` のサポートが追加されました。

```
SELECT * FROM (VALUES 1, 2, 3, 4) INTERSECT ALL SELECT * FROM (VALUES 3, 4);
```

EXCEPT ALL – `EXCEPT ALL` のサポートが追加されました。

```
SELECT * FROM (VALUES 1, 2, 3, 4) EXCEPT ALL SELECT * FROM (VALUES 3, 4);
```

RANGE PRECEDING – ウィンドウ関数の `RANGE PRECEDING` のサポートが追加されました。

```
SELECT sum(x) over (order by x range 1 preceding)
FROM (values (1), (1), (2), (2)) t(x);
```

MATCH_RECOGNIZE – 次の例のように、行パターン照合のサポートが追加されました。

```
SELECT m.id AS row_id, m.match, m.val, m.label
```

```
FROM (VALUES(1, 90),(2, 80),(3, 70),(4, 70)) t(id, value)
MATCH_RECOGNIZE (
        ORDER BY id
        MEASURES match_number() AS match,
        RUNNING LAST(value) AS val,
        classifier() AS label
        ALL ROWS PER MATCH
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (() | A) DEFINE A AS true
) AS m;
```

データ形式およびデータ型の機能強化

Athena エンジンバージョン 3 では、次のデータ形式とデータ型の機能が強化されています。

- LZ4 および ZSTD – LZ4 および ZSTD で圧縮した Parquet データを読み込むためのサポートが追加されました。ZSTD で圧縮した ORC データを書き込むためのサポートが追加されました。
- Symlink ベースのテーブル – Avro ファイルで Symlink ベースのテーブルを作成するためのサポートが追加されました。以下に例を示します。

  ```
  CREATE TABLE test_avro_symlink
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
  ...
  INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
  ```

- SphericalGeography – SphericalGeography 型は、地理座標 (測地座標、緯度/経度、または経度/緯度とも呼ばれる) で表される空間機能のネイティブサポートを提供します。地理座標は、角度単位 (度) で表される球面座標です。

  to_spherical_geography 関数は、次の例のように、地理 (平面) 座標から地理 (球面) 座標を返します。

  ```
  SELECT to_spherical_geography(ST_GeometryFromText(
     'LINESTRING (-40.2 28.9, -40.2 31.9, -37.2 31.9)'));
  ```

重要な変更

Athena エンジンバージョン 2 から Athena エンジンバージョン 3 に移行すると、特定の変更がテーブルスキーマ、構文、またはデータ型の使用に影響を及ぼす可能性があります。このセクションでは、関連するエラーメッセージを一覧表示し、推奨される回避策を示します。

クエリ構文の変更

IGNORE NULLS は value 以外のウィンドウ関数では使用できません

エラーメッセージ: [bool_or 関数の null treatment 句を指定できません。]

原因: IGNORE NULLS は、value 関数の first_value、last_value、nth_value、lead、および lag でのみ使用できるようになりました。この変更は ANSI SQL 仕様に準拠するために行われました。

推奨される解決策: クエリ文字列で、value 以外のウィンドウ関数から IGNORE NULLS を削除します。

CONCAT 関数には 2 つ以上の引数が必要です

エラーメッセージ:「INVALID_FUNCTION_ARGUMENT: There must be two or more concatenation arguments」(INVALID_FUNCTION_ARGUMENT: 2 つ以上の連結引数が必要です)

原因: 以前は、CONCAT 文字列関数は 1 つの引数を受け入れていました。Athena エンジンバージョン 3 では、CONCAT 関数には最低 2 つの引数が必要です。

推奨される解決策: CONCAT(str) の出現箇所を CONCAT(str, '') に変更してください。

Athena エンジンバージョン 3 では、関数に含められる引数は最大 127 個です。詳細については、「関数呼び出しの引数が多すぎます」を参照してください。

prox_percentile 関数は異なる結果を返します。

Athena エンジンバージョン 3 では、approx_percentile 関数が Athena エンジンバージョン 2 とは異なる結果を返します。

エラーメッセージ: なし

原因: この approx_percentile 関数はバージョンが変更される可能性があります。

> **Important**
>
> approx_percentile 関数の出力は近似値であり、その近似値はバージョンごとに変更される可能性があるため、重要なアプリケーションでは approx_percentile 関数に依存しないでください。

推奨される解決策: approx_percentile の Athena エンジンバージョン 2 の動作に近づけるには、Athena エンジンバージョン 3 の別の関数セットを使用できます。例えば、Athena エンジンバージョン 2 では次のクエリがあるとします。

```
SELECT approx_percentile(somecol, 2E-1)
```

Athena エンジンバージョン 3 の出力に近似させるには、次の例のように qdigest_agg 関数と value_at_quantile 関数を試すことができます。ただし、この回避策を使用しても、同じ動作が保証されるわけではないことに注意してください。

```
SELECT value_at_quantile(qdigest_agg(somecol, 1), 2E-1)
```

地理空間関数では、varbinary 入力がサポートされていません

エラーメッセージ:「FUNCTION_NOT_FOUND for st_XXX」(st_XXX に対応する関数が見つかりませんでした)

原因: 一部の地理空間関数では、従来の VARBINARY 入力の型やテキスト関連の関数シグネチャがサポートされなくなりました。

推奨される解決策: 地理空間関数を使用して、入力の型をサポートされているタイプに変換してください。サポートされている入力の型は、エラーメッセージに示されています。

GROUP BY 句では、ネストされた列は二重引用符で囲む必要があります

エラーメッセージ: ["*column_name*"。"*nested_column*" は集計式であるか GROUP BY 句に表示される必要があります]

原因: Athena エンジンバージョン 3 では、GROUP BY 句内のネストされた列名を二重引用符で囲む必要があります。たとえば、次のクエリでは、GROUP BY 句で user.name が二重引用符で囲まれていないためエラーが発生します。

```
SELECT "user"."name" FROM dataset
GROUP BY user.name
```

推奨される解決策: 次の例のように、GROUP BY 句内のネストされた列名を二重引用符で囲みます。

```
SELECT "user"."name" FROM dataset
```

```
GROUP BY "user"."name"
```

Iceberg テーブルで OPTIMIZE を使用すると予期しない FilterNode エラーが発生する

エラーメッセージ:「Unexpected FilterNode found in plan; probably connector was not able to handle provided WHERE expression」

原因: Iceberg テーブルで実行された `OPTIMIZE` ステートメントで、フィルター式に非パーティション列を含む `WHERE` 句が使用されていました。

推奨される解決策: `OPTIMIZE` ステートメントは、パーティションによるフィルタリングのみをサポートします。パーティションテーブルで `OPTIMIZE` を実行する場合、`WHERE` 句にはパーティション列のみを含めてください。非パーティションテーブルで `OPTIMIZE` を実行する場合は、`WHERE` 句を指定しないでください。

Log() 関数の引数の順序

Athena エンジンバージョン 2 では、`log( )` 関数の引数の順序は `log(value, base)` でした。Athena エンジンバージョン 3 では、これは SQL 標準に準拠するように `log(base, value)` に変更されました。

Minute() 関数では、間隔値 year to month がサポートされていません

エラーメッセージ:「Unexpected parameters (interval year to month) for function minute.」(関数 minute の予期しないパラメータ (間隔値 year to month)) 期待パラメータ: minute(timestamp with time zone)、minute(time with time zone)、minute(timestamp)、minute(time)、minute(interval day to second)。

原因: Athena エンジンバージョン 3 では、ANSI SQL 仕様に従って `EXTRACT` の型チェックがより正確に行われました。

推奨される解決策: クエリを更新して、型が推奨された関数シグネチャと一致することを確認してください。

ORDER BY 式は SELECT リストに表示する必要があります。

エラーメッセージ:「For SELECT DISTINCT, ORDER BY expressions must appear in SELECT list」(SELECT DISTINCT、ORDER BY 式は SELECT リストに表示する必要があります)

原因: `SELECT` 句に不適切なテーブルエイリアスが使われています。

推奨される解決策: ORDER BY 式のすべての列に SELECT DISTINCT 句の適切なリファレンスが含まれていることを再確認してください。

サブクエリから返された複数の列を比較するとクエリが失敗する

エラーメッセージの例: [サブクエリの値式と結果は次と同じタイプであることが必要です: row(varchar, varchar) と row(row(varchar, varchar))]

原因: Athena エンジンバージョン 3 における構文の更新により、クエリがサブクエリから返された複数の値を比較しようとした際にサブクエリ SELECT のステートメントが列のリストを括弧で囲んでいる場合に、このエラーが発生します。次の例をご覧ください。

```
SELECT *
FROM table1
WHERE (t1_col1, t1_col2)
IN (SELECT (t2_col1, t2_col2) FROM table2)
```

解決策: Athena エンジンバージョン 3 では、次の更新されたクエリ例のように、サブクエリ SELECT のステートメントにある列リストの周りの括弧を削除します。

```
SELECT *
FROM table1
WHERE (t1_col1, t1_col2)
IN (SELECT t2_col1, t2_col2 FROM table2)
```

SKIP は、DML クエリの予約語です。

この単語 SKIP は、SELECT のような DML クエリの予約語になりました。DML クエリの識別子として SKIP を使用するには、二重引用符で囲みます。

Athena の予約語の詳細については、「予約済みキーワード」を参照してください。

SYSTEM_TIME および SYSTEM_VERSION 句は、タイムトラベルのために非推奨となりました。

エラーメッセージ:「mismatched input 'SYSTEM_TIME'.」(入力 'SYSTEM_TIME' が一致しません)「Expecting: 'TIMESTAMP', 'VERSION'」(期待値: 'TIMESTAMP'、'VERSION')

原因: Athena エンジンバージョン 2 では、Iceberg テーブルがタイムスタンプとバージョンのタイムトラベルに FOR SYSTEM_TIME AS OF および FOR SYSTEM_VERSION AS OF 句を使用していました。Athena エンジンバージョン 3 では、FOR TIMESTAMP AS OF および FOR VERSION AS OF 句が使用されています。

推奨される解決策: 次の例のように、タイムトラベルオペレーションに TIMESTAMP AS OF および VERSION AS OF 句を使用するように SQL クエリを更新してください。

タイムスタンプによるタイムトラベル:

```
SELECT * FROM TABLE FOR TIMESTAMP AS OF (current_timestamp - interval '1' day)
```

バージョンによるタイムトラベル:

```
SELECT * FROM TABLE FOR VERSION AS OF 949530903748831860
```

配列コンストラクターの引数が多すぎる

エラーメッセージ: TOO_MANY_ARGUMENTS: 配列コンストラクターの引数が多すぎます。

原因: 配列コンストラクターの最大要素数が 254 に設定されています。

推奨される解決策: 次の例のように、要素をそれぞれ 254 以下の要素を持つ複数の配列に分割し、CONCAT 関数を使用して配列を連結してください。

```
CONCAT(
ARRAY[x1,x2,x3...x254],
ARRAY[y1,y2,y3...y254],
...
)
```

長さがゼロの区切り記号付き識別子は使用できません

エラーメッセージ:「Zero-length delimited identifier not allowed.」(長さがゼロの区切り記号付き識別子は使用できません)

原因: クエリが列のエイリアスとして空の文字列を使用しました。

推奨される解決策: クエリを更新して、列に空でないエイリアスを使用してください。

データ処理の変更

バケット検証

エラーメッセージ: HIVE_INVALID_BUCKET_FILES: Hive テーブルが壊れています。

原因: テーブルが破損している可能性があります。バケット化されたテーブルについてのクエリの正確性を確保するために、Athena エンジンバージョン 3 ではバケット化されたテーブルに対する追加の検証が有効になっています。これにより、クエリの正確性を確保し、ランタイムにおける予期しないエラーを回避できます。

推奨される解決策: Athena エンジンバージョン 3 を使用してテーブルを再作成してください。

構造体を JSON にキャストするとフィールド名が返されるようになりました

Athena エンジンバージョン 3 の SELECT クエリで `struct` を JSON にキャストすると、キャストによって値 (例: `null`) だけでなく、フィールド名と値 (例: `useragent":null`) の両方が返されるようになりました。

Iceberg テーブルの列レベルのセキュリティ強制の変更

エラーメッセージ:「Access Denied: Cannot select from columns」(アクセスが拒否されました: 列から選択できません)

原因: Iceberg テーブルは Athena の外部で作成され、0.13.0 より前のバージョンの Apache Iceberg SDK を使用しています。以前の SDK バージョンでは、AWS Glue に列が入力されないため、Lake Formation はアクセスが許可されている列を特定できませんでした。

推奨される解決策: Athena ALTER TABLE SET PROPERTIES ステートメントを使用して更新を実行するか、最新の Iceberg SDK を使用してテーブルを修正し、AWS Glue の列情報を更新してください。

データ型 List の Nulls が UDF に反映されるようになりました

エラーメッセージ:「Null Pointer Exception」(Null ポインタ例外)

原因: UDF コネクタを使用しており、ユーザー定義の Lambda 関数を実装している場合に、この問題が発生する可能性があります。

Athena エンジンバージョン 2 では、ユーザー定義の関数に渡されたデータ型 List の nulls が除外されました。Athena エンジンバージョン 3 では、nulls が保持されて UDF に渡されるようになりました。これにより、UDF がチェックすることなく null 要素を逆参照しようとすると、null ポインタ例外が発生する可能性があります。

たとえば、DynamoDB などの元のデータソースにデータ `[null, 1, null, 2, 3, 4]` がある場合、次のデータがユーザー定義の Lambda 関数に渡されます。

Athena エンジンバージョン 2: `[1,2,3,4]`

Athena エンジンバージョン 3: `[null, 1, null, 2, 3, 4]`

推奨される解決策: ユーザー定義の Lambda 関数がデータ型 list の null 要素を処理するようにしてください。

文字配列の部分文字列に埋め込みスペースが含まれなくなりました

エラーメッセージ: エラーはスローされませんでしたが、返された文字列には埋め込みスペースが含まれていません。たとえば、`substr(char[20],1,100)` は 100 文字ではなく 20 文字の長さの文字列を返すようになりました。

推奨される解決策: アクションは不要です。

サポートされていない列型 decimal の強制

エラーメッセージ: HIVE_CURSOR_ERROR: Parquet ファイル: s3://DOC-EXAMPLE-BUCKET/*path*/*file_name*.parquet の読み取りに失敗、または Parquet 列 ([*column_name*]) でサポートされていない列タイプ (varchar) です

原因: Athena エンジンバージョン 2 は、`varchar` から 10 進数へのデータ型の強制変換を試みた際に、時々成功しました (しかし、頻繁に失敗しました)。Athena エンジンバージョン 3 は、値の読み取りを試みる前に型に互換性があるかどうかをチェックする型検証機能を備えているため、このような強制変換の試みは常に失敗するようになりました。

推奨される解決策: Athena エンジンバージョン 2 と Athena エンジンバージョン 3 の両方で、Parquet ファイル内の 10 進数列について `varchar` ではなく数値データ型を使用するように AWS Glue のスキーマを変更してください。データを再クロールして新しい列のデータ型が 10 進数型であることを確認するか、Athena でテーブルを手動で再作成し、構文 `decimal(`*precision*`,` *scale*`)` を使用して列のデータ型として decimal を指定します。

浮動小数点数または倍精度浮動小数点数の NaN 値を bigint にキャストできなくなりました

エラーメッセージ:INVALID_CAST_ARGUMENT: Cannot cast real/double NaN to bigint

原因: Athena エンジンバージョン 3 では、`bigint` として `NaN` を 0 にキャストできなくなりました。

推奨される解決策: `bigint` にキャストする場合は、`NaN` の値が `float` 列または `double` 列に存在しないことを確認してください。

uuid() 関数の戻り型に関する変更

次の問題は、テーブルとビューの両方に影響します。

エラーメッセージ:[サポート対象外の Hive タイプ: uuid]

原因: Athena エンジンバージョン 2 では `uuid()` 関数は文字列を返しましたが、Athena エンジンバージョン 3 では UUID (タイプ 4) をランダムに生成した疑似文字列を返します。Athena では UUID 列のデータ型はサポートされていないため、Athena エンジンバージョン 3 では `uuid()` 関数を CTAS クエリで直接使用して UUID 列を生成することはできなくなりました。

たとえば、次の `CREATE TABLE` ステートメントは Athena エンジンバージョン 2 では正常に完了しますが、Athena エンジンバージョン 3 では [NOT_SUPPORTED: サポート対象外の Hive タイプ: uuid] が返されます。

```
CREATE TABLE uuid_table AS
   SELECT uuid() AS myuuid
```

たとえば、次の `CREATE VIEW` ステートメントは Athena エンジンバージョン 2 では正常に完了しますが、Athena エンジンバージョン 3 では [myuuid 列に無効な列タイプ: サポート対象外の Hive タイプ: uuid] が返されます。

```
CREATE VIEW uuid_view AS
   SELECT uuid() AS myuuid
```

Athena エンジンバージョン 2 で作成されたビューを Athena エンジンバージョン 3 でクエリすると、次のようなエラーが発生します。

[VIEW_IS_STALE: 1:15 行目:ビュー 「awsdatacatalog.mydatabase.uuid_view」が古くなっているか無効な状態です: クエリビューから位置 0 に射影された uuid 型の列 [muuid] を、ビュー定義に保存されている varcharar 型の列 [muuid] に強制変換できません]

推奨される解決方法: テーブルまたはビューを作成するときには、次の例のように、`cast()` 関数を使用して `uuid()` の出力を`varchar` に変換します。

```
CREATE TABLE uuid_table AS
   SELECT CAST(uuid() AS VARCHAR) AS myuuid
```

```
CREATE VIEW uuid_view AS
```

```
    SELECT CAST(uuid() AS VARCHAR) AS myuuid
```

CHAR と VARCHAR の強制に関する問題

Athena エンジンバージョン 3 の varchar および char で強制上の問題が発生した場合は、このセクションにある回避策を使用してください。これらの回避策を使用できない場合は、「AWS Support」にお問い合わせください。

CHAR 入力と VARCHAR 入力の混在による CONCAT 関数の失敗

問題: 次のクエリは Athena エンジンバージョン 2 では成功します。

```
SELECT concat(CAST('abc' AS VARCHAR(20)), '12', CAST('a' AS CHAR(1)))
```

しかしながら、Athena エンジンバージョン 3 では、同じクエリが次のように失敗します。

エラーメッセージ: FUNCTION_NOT_FOUND: 1:8 行目:concat 関数に予期しないパラメータ (varchar (20)、varchar (2)、char (1))。期待値: concat(char(x), char(y))、concat(array(E), E) E、concat(E, array(E)) E、concat(array(E)) E、concat(varchar)、concat(varbinary)

推奨される解決方法: concat 関数を使用するときは、char または varchar にキャストし、両方を混合しないでください。

CHAR 入力と VARCHAR 入力による SQL || 連結の失敗

Athena エンジンバージョン 3 では、二重の縦線 || 連結演算子では、入力が varchar である必要があります。入力に、varchar と char タイプを組み合わせて使用することはできません。

エラーメッセージ: [TYPE_NOT_FOUND: 1:26 行目:不明なタイプ:char (65537)]

原因: || を使用して char と varchar を連結するクエリで、次の例のようなエラーが発生する可能性があります。

```
SELECT CAST('a' AS CHAR) || CAST('b' AS VARCHAR)
```

推奨される解決方法: 次の例のようにして varchar と varchar を連結します。

```
SELECT CAST('a' AS VARCHAR) || CAST('b' AS VARCHAR)
```

CHAR および VARCHAR UNION クエリの失敗

エラーメッセージ: NOT_SUPPORTED: サポート対象外の Hive の型: char(65536)。サポートされている CHAR タイプ: CHAR (<= 255)

原因: 次の例のように、クエリが `char` と `varchar` を組み合わせようとしています。

```
CREATE TABLE t1 (c1) AS SELECT CAST('a' as CHAR) as c1 UNION ALL SELECT CAST('b' AS
 VARCHAR) AS c1
```

推奨される解決方法: このクエリ例では、`'a'` を `char` ではなく `varchar` にキャストします。

CHAR または VARCHAR の強制による不要な空白スペース

Athena エンジンバージョン 3 では、配列または単一の列の構成で `char(X)` と `varchar` のデータが単一の型に強制的に変換される場合、ターゲットの型は `char(65535)` になり各フィールドの末尾には不要なスペースが多く含まれます。

原因: Athena エンジンバージョン 3 では、強制的に `varchar` と `char(X)` を `char(65535)` に変換し、その後データの右にスペースを埋め込みます。

推奨される解決策: 明示的に各フィールドを `varchar` にキャストします。

タイムスタンプの変更

タイムゾーン付きの Timestamp を varchar の動作変更にキャストしています

Athena エンジンバージョン 2 では、タイムゾーン付きの `Timestamp` を `varchar` にキャストすると、一部のタイムゾーンリテラルが変更されました (例: `US/Eastern` が `America/New_York` に変更)。この動作は、Athena エンジンバージョン 3 では発生しません。

Date タイムスタンプのオーバーフローでエラーがスローされます

エラーメッセージ:「Millis overflow: XXX」(Millis オーバーフロー: XXX)

原因: Athena エンジンバージョン 2 では ISO 8601 の日付のオーバーフローがチェックされなかったため、一部の日付では負のタイムスタンプが生成されました。Athena エンジンバージョン 3 では、このオーバーフローがチェックされ、例外がスローされます。

推奨される解決策: タイムスタンプが範囲内にあることを確認してください。

TIME では政治タイムゾーンはサポートされていません

エラーメッセージ:「INVALID LITERAL」(リテラルが無効です)

原因: `SELECT TIME '13:21:32.424 America/Los_Angeles'` などのクエリ。

推奨される解決策: `TIME` を含む政治タイムゾーンの使用を避けてください。

Timestamp 列の精度の不一致により、シリアル化エラーが発生します

エラーメッセージ:「SERIALIZATION_ERROR: Could not serialize column '*COLUMNZ*' of type 'timestamp(3)' at position *X*:*Y*」(型 'timestamp(3)' の列 'COLUMNZ' を位置 X:Y でシリアル化できませんでした)

COLUMNZ は、問題の原因となる列の出力名です。*X*:*Y* という数字は、出力内の列の位置を示します。

原因: Athena エンジンバージョン 3 では、データ内のタイムスタンプの精度が、テーブル仕様の列のデータ型に指定されている精度と同じかどうかがチェックされます。現在、この精度は常に 3 です。データの精度が 3 より大きい場合、クエリは失敗し、記載されているエラーが表示されます。

推奨される解決策: データをチェックして、タイムスタンプの精度がミリ秒であることを確認してください。

Iceberg テーブルの UNLOAD クエリと CTAS クエリにあるタイムスタンプの精度が正しくない

エラーメッセージ: [timestamp(6) のタイムスタンプの精度が正しくありません。設定されている精度はミリ秒単位です]

原因: Athena エンジンバージョン 3 では、データ内のタイムスタンプの精度が、テーブル仕様の列のデータ型に指定されている精度と同じかどうかがチェックされます。現在、この精度は常に 3 です。データの精度がこれより大きい場合 (たとえば、ミリ秒ではなくマイクロ秒)、記載されているエラーが表示されてクエリが失敗する可能性があります。

解決策: この問題を回避するには、次の Iceberg テーブルを作成する CTAS の例のように、まずタイムスタンプの精度を 6 に CAST (キャスト) します。Iceberg で Timestamp precision (3) がサポートされていないというエラーを回避するために、精度を 3 ではなく 6 に指定する必要があることに注意してください。

```
CREATE TABLE my_iceberg_ctas
WITH (table_type = 'ICEBERG', location = 's3://DOC-EXAMPLE-BUCKET/table_ctas/',
format = 'PARQUET')
```

```
AS SELECT id, CAST(dt AS timestamp(6)) AS "dt"
FROM my_iceberg
```

次に、Athena はタイムスタンプ 6 をサポートしていないので、その値を (例えば、ビュー内の) タイムスタンプに再度キャストします。次の例では、my_iceberg_ctas テーブルからビューを作成します。

```
CREATE OR REPLACE VIEW my_iceberg_ctas_view AS
SELECT cast(dt AS timestamp) AS dt
FROM my_iceberg_ctas
```

ORC ファイルで Long 型をタイムスタンプとして読み込む、または書き込むと誤った形式の ORC ファイルとしてエラーが発生するようになりました

エラーメッセージ:「Error opening Hive split 'FILE (SPLIT POSITION)' Malformed ORC file.」(Hive 分割 'FILE (SPLIT POSITION)' を開く際のエラー。不正な ORC ファイル) Cannot read SQL type timestamp from ORC stream .long_type of type LONG (Hive で分割された「FILE (SPLIT POSITION)」を開くときにエラーが発生しました。誤った形式の ORC ファイルです。LONG 型の ORC ストリーム .long_type から SQL 型のタイムスタンプを読み込むことができません)」

原因: Athena エンジンバージョン 3 では、データ型が Long から Timestamp または Timestamp から Long への暗黙的な強制を拒否するようになりました。以前は、Long 値はあたかも工ポックミリ秒であるかのように、暗黙的にタイムスタンプに変換されていました。

推奨される解決策: from_unixtime 関数を使用して列を明示的にキャストするか、from_unixtime 関数を使用して将来のクエリで追加の列を作成してください。

時刻と間隔値 year to month はサポートされていません

エラーメッセージ:「TYPE MISMATCH」(型の不一致)

原因: Athena エンジンバージョン 3 では、時刻と間隔値 year to month がサポートされていません (例: SELECT TIME '01:00' + INTERVAL '3' MONTH)。

int96 Parquet 形式のタイムスタンプのオーバーフロー

エラーメッセージ:「Invalid timeOfDayNanos」(timeOfDayNanos が無効です)

原因: int96 Parquet 形式のタイムスタンプのオーバーフロー。

推奨される解決策: 問題のある特定のファイルを特定してください。次に、よく知られている最新の Parquet ライブラリを使用してデータファイルを再度生成するか、Athena CTAS を使用してくださ

い。問題が解決しない場合は、Athena サポートに連絡して、データファイルの生成方法をお知らせください。

文字列からタイムスタンプにキャストする際に日付と時刻の値の間に必要なスペース

エラーメッセージ: [INVALID_CAST_ARGUMENT: 値はタイムスタンプにキャストできません。]

原因: Athena エンジンバージョン 3 では、cast (キャスト) する入力文字列にある日付と時刻の値の間の有効な区切り記号としてハイフンを使用できなくなりました。たとえば、次のクエリは Athena エンジンバージョン 2 では動作しますが、Athena エンジンバージョン 3 では動作しません:

```
SELECT CAST('2021-06-06-23:38:46' AS timestamp) AS this_time
```

推奨される解決策: Athena エンジンバージョン 3 では、次の例のように、日付と時刻の間にあるハイフンをスペースに置き換えます。

```
SELECT CAST('2021-06-06 23:38:46' AS timestamp) AS this_time
```

to_iso8601() タイムスタンプの戻り値に関する変更

エラーメッセージ: なし

原因: Athena エンジンバージョン 2 では、to_iso8601 関数は、関数に渡された値にタイムゾーンが含まれていなくてもタイムゾーンを含むタイムスタンプを返します。Athena エンジンバージョン 3 では、to_iso8601 関数は、渡された引数にタイムゾーンが含まれている際にのみタイムゾーンを含むタイムスタンプを返します。

たとえば、次のクエリは現在の日付を to_iso8601 関数に 2 回渡します。1 回目はタイムゾーンを含むタイムスタンプとして、2 回目はタイムスタンプとして渡します。

```
SELECT TO_ISO8601(CAST(CURRENT_DATE AS TIMESTAMP WITH TIME ZONE)),
 TO_ISO8601(CAST(CURRENT_DATE AS TIMESTAMP))
```

次の出力は、各エンジンにおけるクエリの結果を示しています。

Athena エンジンバージョン 2:

#	_col0	_col1
1	2023-02-24T00:00:00.000Z	2023-02-24T00:00:00.000Z

Athena エンジンバージョン 3:

#	_col0	_col1
1	2023-02-24T00:00:00.000Z	2023-02-24T00:00:00.000

推奨される解決策: 前述の動作を再現するには、次の例のように、タイムスタンプ値を to_iso8601 に渡す前に with_timezone 関数に渡します:

```
SELECT to_iso8601(with_timezone(TIMESTAMP '2023-01-01 00:00:00.000', 'UTC'))
```

結果

#	_col0
1	2023-01-01T00:00:00.000Z

at_timezone() の最初のパラメータに日付を指定する必要がある

問題: Athena エンジンバージョン 3 では、at_timezone 関数は time_with_timezone の値を最初のパラメータとして受け取れません。

原因: 日付情報がないと、渡された値が夏時間か標準時かを判断できません。たとえば、渡された値が太平洋夏時間 (PDT) か太平洋標準時 (PST) かを判断する方法がないため、at_timezone('12:00:00 UTC', 'America/Los_Angeles') はあいまいです。

制限事項

Athena エンジンバージョン 3 には次の制約事項があります。

- クエリパフォーマンス – Athena エンジンバージョン 3 ではクエリの多くが高速に実行されますが、一部のクエリ計画は Athena エンジンバージョン 2 とは異なる場合があります。その結果、一部のクエリではレイテンシーやコストが異なる場合があります。
- Trino および Presto コネクタ – Trino コネクタも Presto コネクタもサポートされていません。データソースの接続には Amazon Athena の横串検索を使用します。詳細については、「Amazon Athena 横串検索の使用」を参照してください。

- フォールトトレラントの実行 – Trino フォールトトレラントの実行 (Trino Tardigrade) はサポートされていません。
- 関数パラメータの制限 — 関数は 127 個を超えるパラメータを取れません。詳細については、「関数呼び出しの引数が多すぎます」を参照してください。

リソース制限によるクエリの失敗が発生しないことを確実にするため、Athena エンジンバージョン 2 に以下の制限が導入されました。これらの制限は、ユーザー設定可能な制限ではありません。

- 結果要素の数 – `min(col, n)`、`max(col, n)`、`min_by(col1, col2, n)`、および `max_by(col1, col2, n)` 関数では、結果要素 n の数が 10,000 以下に制限されます。
- GROUPING SETS – グループ化セット内のスライスの最大数は 2,048 です。
- テキストファイルの最大行長 — テキストファイルのデフォルト最大行長は 200 MB です。
- シーケンス関数の最大結果サイズ – シーケンス関数の最大結果サイズは 50,000 エントリです。例えば、`SELECT sequence(0,45000,1)` は成功しますが、`SELECT sequence(0,55000,1)` は「The result of the sequence function must not have more than 50000 entries」というエラーメッセージを伴って失敗します。この制限は、タイムスタンプを含むシーケンス関数のすべての入力タイプに適用されます。

Athena エンジンバージョン 2

Athena エンジンバージョン 2 には、次の変更が行われました。

- 改善点と新機能
  - グループ化、結合、およびサブクエリの改善
  - データ型の拡張
  - 追加された関数
  - フォーマンスの向上
  - JSON 関連の改善
- 重要な変更
  - バグ修正
  - 地理空間関数への変更
  - ANSI SQL コンプライアンス
  - 置き換えられた関数
  - 制限

改善点と新機能

- EXPLAIN および EXPLAIN ANALYZE – Athena で EXPLAIN ステートメントを使用して、SQL クエリの実行プランを表示できます。EXPLAIN ANALYZE では、SQL クエリの分散実行プランと各オペレーションのコストを表示します。詳細については、「Athena での EXPLAIN および EXPLAIN ANALYZE の使用」を参照してください。
- フェデレーティッドクエリ – フェデレーティッドクエリは、Athena エンジンバージョン 2 でサポートされています。詳細については、「Amazon Athena 横串検索の使用」を参照してください。
- 地理空間関数 – 25 以上の地理空間関数が追加されました。詳細については、「Athena エンジンバージョン 2 の新しい地理空間関数」を参照してください。
- ネストされたスキーマ – ネストされたスキーマの読み取りに対するサポートが追加されました。これは、コストを削減します。
- 準備済みステートメント – 同じクエリを、異なるクエリパラメータを使用して繰り返し実行するには、準備済みステートメントを使用します。準備済みステートメントでは、パラメータがプレースホルダになっており、その値を実行時に渡すことができます。準備済みステートメントは、SQL インジェクション攻撃を防ぐのに役立ちます。詳細については、「パラメータ化されたクエリの使用」を参照してください。
- スキーマ進化のサポート – Parquet 形式のデータに対するスキーマ進化のサポートが追加されました。
  - パーティションスキーマがテーブルスキーマとは異なるパーティションからの配列、マップ、または行型の列の読み込みに対するサポートが追加されました。これは、パーティションの作成後にテーブルスキーマが更新された場合に発生する可能性があります。変更された列型には互換性がある必要があります。行型については、末尾のフィールドを追加またはドロップすることができますが、対応するフィールド (序数順) に同じ名前が付けられている必要があります。
  - ORC ファイルに、フィールドが欠落している構造体の列を使用できるようになりました。これにより、ORC ファイルを書き換えることなくテーブルスキーマを変更することが可能になります。
  - ORC 構造体の列が序数ではなく名前でマップされるようになりました。これにより、ORC ファイル内にある欠落した、または余分な構造体フィールドが正しく処理されます。
- SQL OFFSET — SQLの SELECT ステートメントで、OFFSET 句がサポートされるようになりました。詳細については、「SELECT」を参照してください。
- UNLOAD ステートメント – UNLOAD ステートメントを使用して、SELECT クエリの出力を PARQUET、ORC、AVRO、および JSON 形式で記述することができます。詳細については、「UNLOAD」を参照してください。

グループ化、結合、およびサブクエリの改善

- 複雑なグループ化 – 複雑なグループ化オペレーションのサポートが追加されました。
- 相関サブクエリ – IN 述語の相関サブクエリ、および強制を必要とする相関サブクエリのサポートが追加されました。
- CROSS JOIN – LATERAL 導出テーブルに対する CROSS JOIN のサポートが追加されました。
- GROUPING SETS – GROUPING SETS を使用するクエリのための集計における ORDER BY 句のサポートが追加されました。
- Lambda 式 – Lambda 式の行フィールドの逆参照に対するサポートが追加されました。
- 準結合での Null 値 – 準結合の左側 (つまり、サブクエリでの IN 述語) での null 値のサポートが追加されました。
- 空間結合 – ブロードキャスト空間結合と空間左結合のサポートが追加されました。
- ディスクへのスピル – メモリ集約型の INNER JOIN および LEFT JOIN オペレーションでは、Athena が中間オペレーション結果をディスクにオフロードします。これにより、大量のメモリを必要とするクエリの実行が可能になります。

データ型の拡張

- INTEGER 用の INT – INTEGER データ型のエイリアスとしての INT のサポートが追加されました。
- INTERVAL 型 – INTERVAL 型へのキャスティングのサポートが追加されました。
- IPADDRESS – DML クエリ内の IP アドレスを表すための新しい IPADDRESS 型を追加しました。VARBINARY 型と IPADDRESS 型間におけるキャスティングのサポートが追加されました。DDL クエリでは、IPADDRESS 型が認識されません。
- IS DISTINCT FROM – JSON および IPADDRESS 型に対する IS DISTINCT FROM のサポートが追加されました。
- Null 等価チェック – ARRAY、MAP、および ROW データ構造における null 値の等価チェックがサポートされるようになりました。例えば、ARRAY ['1', '3', null] = ARRAY ['1', '2', null] 式は false を返します。これまで、null 要素は「comparison not supported」というエラーメッセージを返していました。
- 行型の強制 – フィールド名を問わない行型間での強制が可能になりました。これまでは、ソースとなる型のフィールド名がターゲットとなる型のものと一致する場合、またはターゲットとなる型に匿名化されたフィールド名がある場合にのみ、行型を別の型に強制することが可能でした。
- 時間の減算 – すべての TIME および TIMESTAMP 型に減算を実装しました。

- Unicode – 文字列リテラルでのエスケープされた Unicode シーケンスのサポートが追加されました。
- VARBINAY の連結 – VARBINARY 値の連結に対するサポートが追加されました。

  ウィンドウ値関数 – ウィンドウ値関数で IGNORE NULLS および RESPECT NULLS をサポートするようになりました。

追加の関数入力タイプ

以下の関数が追加の入力タイプを許可するようになりました。各関数の詳細については、対応する Presto ドキュメントへのリンクを参照してください。

- approx_distinct() – approx_distinct() 関数が次の型をサポートするようになりました: INTEGER、SMALLINT、TINYINT、DECIMAL、REAL、DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIME、TIME WITH TIME ZONE、IPADDRESS、および CHAR。
- Avg()、sum() – avg() および sum() 集計関数が INTERVAL データ型をサポートするようになりました。
- Lpad()、rpad() – lpad および rpad 関数が VARBINARY 入力で機能するようになりました。
- Min()、max() – min() および max() 集計関数が、クエリ分析時間での不明な入力タイプを許可するようになったため、NULL リテラルでのこれらの関数の使用が可能になりました。
- regexp_replace() – 置換ごとに Lambda 関数を実行できる regexp_replace() 関数のバリアントが追加されました。
- Sequence() – 暗黙的な one-day ステップ増分でのバリアントを含む、sequence() 関数のための DATE バリアントが追加されました。
- ST_Area() – ST_Area() 地理空間関数が、すべてのジオメトリ型をサポートするようになりました。
- Substr() – substr 関数が VARBINARY 入力で動作するようになりました。
- zip_with() – 長さが一致しない配列を zip_with() で使用できるようになりました。欠落した位置指定は null で埋められます。これまで、異なる長さの配列が渡された場合はエラーが発生していました。この変更により、配列を同じ長さにするために追加された値から、もともと null であった値を区別することが困難になる可能性があります。

追加された関数

次のリストには、Athena エンジンバージョン 2 から採用された新しい関数が記載されています。このリストに地理空間関数は含まれていません。地理空間関数のリストについては、「Athena エンジンバージョン 2 の新しい地理空間関数」を参照してください。

各関数の詳細については、対応する Presto ドキュメントへのリンクを参照してください。

集計関数

reduce_agg()

配列関数と演算子

array_sort() – Lambda 関数をコンパレータとして受け取る、この関数のバリアントが追加されました。

ngrams()

バイナリ関数と演算子

from_big_endian_32()

from_ieee754_32()

from_ieee754_64()

hmac_md5()

hmac_sha1()

hmac_sha256()

hmac_sha512()

spooky_hash_v2_32()

spooky_hash_v2_64()

to_big_endian_32()

to_ieee754_32()

to_ieee754_64()

日付/時刻関数と演算子

millisecond()

parse_duration()

to_milliseconds()

マッピング関数と演算子

multimap_from_entries()

数学関数と演算子

inverse_normal_cdf()

wilson_interval_lower()

wilson_interval_upper()

分位点ダイジェスト関数

分位点ダイジェスト関数と `qdigest` 分位点ダイジェスト型を追加しました。

文字列関数および演算子

hamming_distance()

split_to_multimap()

フォーマンスの向上

Athena エンジンバージョン 2 では、以下の機能のパフォーマンスが改善されました。

クエリパフォーマンス

- ブロードキャスト参加のパフォーマンス – ワーカーノードに動的パーティションプルーニングを適用することで、ブロードキャスト参加のパフォーマンスが向上しました。
- バケット化されたテーブル – 書き込まれるデータが既に適切にパーティション化されている場合 (例えば、出力がバケット化された結合からの場合など) におけるバケット化されたテーブルへの書き込みのパフォーマンスが改善されました。
- DISTINCT – `DISTINCT` を使用する一部のクエリのパフォーマンスが改善されました。

  動的フィルタリングとパーティションプルーニング - これらの改善によってパフォーマンスが向上し、クエリでスキャンされるデータ量が削減されます。

- フィルタ演算と射影演算 – 可能な場合には、フィルタ演算と射影演算が常に列で処理されるようになりました。エンジンは、辞書エンコーディングが有効であれば、それを自動的に利用します。
- Gathering exchange – Gathering exchange でのクエリのパフォーマンスが改善されました。
- グローバル集計 – フィルタリングされたグローバル集計を実行する一部のクエリのパフォーマンスが改善されました。
- GROUPING SETS、CUBE、ROLLUP – 1 つのクエリに列のセットを複数集約できるので、`GROUPING SETS`、`CUBE`、または `ROLLUP` に関連するクエリのパフォーマンスが向上します。
- きわめて選択的なフィルタ – きわめて選択的なフィルタを使用したクエリのパフォーマンスが改善されました。
- JOIN および AGGREGATE オペレーション – `JOIN` および `AGGREGATE` オペレーションのパフォーマンスが強化されました。
- LIKE – `information_schema` テーブルの列で `LIKE` 述語を使用するクエリのパフォーマンスが改善されました。
- ORDER BY および LIMIT – 不必要なデータ交換を回避するため、`ORDER BY` と `LIMIT` が関与するクエリのプラン、パフォーマンス、およびメモリ使用量が改善されました。
- ORDER BY – `ORDER BY` オペレーションがデフォルトで分散されるようになり、より大きな `ORDER BY` 句の使用が可能になりました。
- ROW 型の変換 – `ROW` 型を変換する時のパフォーマンスが改善されました。
- 構造型 – 構造型を処理し、スキャン、結合、集約、またはテーブル書き込みを含むクエリのパフォーマンスが改善されました。
- テーブルのスキャン – 特定のケースで発生するテーブルスキャンの重複を回避するための、最適化ルールが導入されました。
- UNION – `UNION` クエリのパフォーマンスが改善されました。

クエリプランのパフォーマンス

- プランパフォーマンス – 多数の列を持つ複数のテーブルを結合するクエリのプランパフォーマンスが改善されました。
- 述語評価 – プランにおける述語プッシュダウン中の述語評価のパフォーマンスが改善されました。
- キャスティングのための述語プッシュダウンのサポート – values list の値が列の型を一致させるためのキャスティングを必要とする *`<column>`* `IN` *`<values list>`* 述語に対する述語プッシュダウンがサポートされます。

- 述語推論とプッシュダウン – 述語推論とプッシュダウンが *<symbol>* IN *<subquery>* 述語を使用するクエリに拡張されました。
- タイムアウト – クエリプランニングのタイムアウトをまれに引き起こす可能性のあるバグを修正しました。

結合のパフォーマンス

- マップ列での結合 – マップ列を含む結合と集計のパフォーマンスが改善されました。
- 非等価条件のみでの結合 – ハッシュ結合の代わりにネステッドループ結合を使用することで、非等価条件のみでの結合のパフォーマンスを改善しました。
- 外部結合 – 外部結合が関与するクエリに対して、結合分布型が自動的に選択されるようになりました。
- 関数の範囲指定による結合 – 条件が関数の範囲指定 (`a JOIN b ON b.x < f(a.x) AND b.x > g(a.x)` など) である場合の結合のパフォーマンスが改善されました。
- ディスクへのスピル – ディスクへのスピルに関連するバグとメモリの問題を修正し、`JOIN` オペレーションのパフォーマンスを向上させると同時にメモリエラーを削減しました。

サブクエリのパフォーマンス

- 相関 EXISTS サブクエリ – 相関 `EXISTS` サブクエリのパフォーマンスが改善されました。
- 等価述語を使用する相関サブクエリ – 等価述語が含まれる相関サブクエリのサポートが改善されました。
- 非等価述語を使用する相関サブクエリ – 不等価述語が含まれる相関サブクエリのパフォーマンスが改善されました。
- サブクエリに対する count(*) 集計 – 既知の安定したカーディナリティでのサブクエリでの `count(*)` 集計のパフォーマンスが改善されました。
- 外部クエリのフィルタのプロパゲーション – 外部クエリからのフィルタをサブクエリにプロパゲートできる場合における相関サブクエリのパフォーマンスが改善されました。

関数のパフォーマンス

- 集約 Window 関数 – 集約 Window 関数のパフォーマンスが改善されました。
- element_at() – マップに対する `element_at()` の改善されたパフォーマンスが、マップのサイズに比例するのではなく、定数時間になります。

- Grouping() – `grouping()` が関与するクエリのパフォーマンスが改善されました。
- JSON キャスティング – JSON から ARRAY または MAP 型へのキャスティングのパフォーマンスが改善されました。
- マップを返す関数 – マップを返す関数のパフォーマンスが改善されました。
- マップからマップへのキャスティング – マップからマップへのキャスティングのパフォーマンスが改善されました。
- min() および max() – `min()` および `max()` 関数が、不要なオブジェクトの作成を回避するように最適化されました。これにより、ガベージコレクションのオーバーヘッドが軽減されます。
- row_number() – `row_number()` を使用するクエリで、生成された行数のフィルタが後に続くもののパフォーマンスとメモリ使用量が改善されました。
- Window 関数 – 同一の `PARTITION BY` と `ORDER BY` 句を持つウィンドウ関数が含まれるクエリのパフォーマンスが改善されました。
- Window 関数 – 類似した仕様を持つ特定の Window 関数 (LAG など) のパフォーマンスが改善されました。

地理空間のパフォーマンス

- ジオメトリのシリアル化 – ジオメトリ値のシリアル化パフォーマンスが改善されました。
- 地理空間関数 – `ST_Intersects()`、`ST_Contains()`、`ST_Touches()`、`ST_Within()`、`ST_Overlaps()`、`ST_Dis` および `array_intersect()` のパフォーマンスが改善されました。
- ST_Distance() – `ST_Distance()` 関数が関与する結合クエリのパフォーマンスが改善されました。
- ST_Intersection() – 座標軸に平行な矩形 (`ST_Envelope()` および `bing_tile_polygon()` 関数で生成されたポリゴンなど) のための `ST_Intersection()` 関数が最適化されました。

JSON 関連の改善

マップ関数

- すべてのケースにおける O(n) から O(1) のマップサブスクリプトのパフォーマンスが改善されました。これまで、この改善を利用できるのは特定の関数とリーダーによって作成されたマップのみでした。
- `map_from_entries()` 関数と `map_entries()` 関数を追加しました。

キャスティング

- REAL、TINYINT、または SMALLINT から JSON にキャストする機能が追加されました。
- 今後は、JSON が ROW にすべてのフィールドを含まない場合でも、JSON を ROW にキャストできるようになります。
- CAST(json_parse(...) AS ...) のパフォーマンスが改善されました。
- JSON から ARRAY または MAP 型へのキャスティングのパフォーマンスが改善されました。

新しい JSON 関数

- is_json_scalar()

重要な変更

破壊的変更には、バグ修正、地理空間関数への変更、置き換えられた関数、および制限の導入が含まれます。ANSI SQL コンプライアンスにおける改善により、非標準的な動作に依存するクエリが破損する可能性があります。

バグ修正

以下の変更により、クエリが正常に実行されても結果が不正確になる原因となっていた動作上の問題が修正されます。

- Parquet の fixed_len_byte_array 列が DECIMAL として受け付けられるようになりました – fixed_len_byte_array タイプの Parquet 列が、Parquet スキーマで DECIMAL として注釈付けされている場合は、それらに対するクエリが成功し正しい値が返されます。DECIMAL の注釈がない fixed_len_byte_array 列に対するクエリはエラーとなります。これまでは、DECIMAL の注釈がない fixed_len_byte_array 列へのクエリは成功するものの、意味のない値が返されていました。
- json_parse() が末尾の文字を無視しなくなりました – これまでは、[1,2]abc などの入力が [1,2] として正常に解析されていました。末尾の文字を使用することにより、「Cannot convert '[1, 2]abc' to JSON」というエラーメッセージが表示されるようになります。
- round() 小数点精度が修正されました – x が DECIMAL、または x がスケール 0 の DECIMAL で d が負の整数である場合に、round(x, d) が x を正しく四捨五入するようになりました。これまで、これらのケースでは四捨五入が行われていませんでした。

- round(x, d) および truncate(x, d) – `round(x, d)` および `truncate(x, d)` 関数のシグネチャのパラメータ d が INTEGER 型になりました。これまでは、d が BIGINT 型である場合がありました。
- 重複したキーを持つ map() – `map()` が重複したキーに関するエラーを生成するようになり、エラーなしで破損したマップを作成することがなくなりました。現在重複キーを使用してマップ値を構築するクエリは、今後エラーを伴って失敗するようになります。
- map_from_entries() が null エントリでエラーを生成します – 入力配列に null エントリが含まれている場合、`map_from_entries()` がエラーを生成するようになりました。NULL を値として渡すことでマップを構築するクエリは失敗するようになります。
- テーブル – サポートされていないパーティションタイプを持つテーブルは作成できなくなりました。
- 統計関数における数値安定性の改善 – 統計関数 `corr()`、`covar_samp()`、`regr_intercept()`、および `regr_slope()` の数値安定性が改善されました。
- Parquet で定義されているタイムスタンプの精度が尊重されるようになりました – 今後は、TIMESTAMP 値の精度と、Parquet スキーマの TIMESTAMP 列で定義される精度を一致させる必要があります。これらの精度が一致しないと、不適切なタイムスタンプが生成されます。
- タイムゾーン情報 – タイムゾーン情報が Java 1.8 SDK の java.time パッケージを使用して計算されるようになりました。
- INTERVAL_DAY_TO_SECOND および INTERVAL_YEAR_TO_MONTH データ型の SUM – `SUM(NULL)` を直接使用することができなくなりました。`SUM(NULL)` を使用するには、NULL を BIGINT、DECIMAL、REAL、DOUBLE、INTERVAL_DAY_TO_SECOND または INTERVAL_YEAR_TO_MONTH といったデータ型にキャストしてください。

地理空間関数への変更

地理空間関数に対して行われた変更は以下のとおりです。

- 関数名の変更 – 一部の関数名が変更されました。詳細については、「Athena エンジンバージョン 2 における地理空間関数名の変更」を参照してください。
- VARBINARY 入力 – VARBINARY 型が、地理空間関数への入力に直接サポートされなくなりました。例えば、ジオメトリの面積を直接計算するには、ジオメトリを VARCHAR または GEOMETRY の形式で入力することが必要になります。回避策は、以下の例にあるように、変換関数を使用することです。

- 周知のバイナリ (WKB) 形式での VARBINARY 入力の面積を計算するために ST_area() を使用するには、以下の例のように、まず入力を ST_GeomFromBinary() に渡します。

```
ST_area(ST_GeomFromBinary(<wkb_varbinary_value>))
```

- レガシーバイナリ形式での VARBINARY 入力の面積を計算するために ST_area() を使用するには、以下の例のように、まず同じ入力を ST_GeomFromLegacyBinary() 関数に渡します。

```
ST_area(ST_GeomFromLegacyBinary(<legacy_varbinary_value>))
```

- ST_ExteriorRing() および ST_Polygon() – ST_ExteriorRing() および ST_Polygon() が、入力としてポリゴンのみを受け入れるようになりました。これまで、これらの関数は他のジオメトリを誤って受け入れていました。
- ST_Distance() – SQL/MM 仕様で必須とされているとおり、入力の 1 つが空のジオメトリである場合、ST_Distance() 関数が NULL を返すようになりました。これまでは、NaN が返されました。

ANSI SQL コンプライアンス

ANSI SQL 標準に従うため、以下の構文と動作の問題が修正されました。

- Cast() オペレーション – REAL または DOUBLE から DECIMAL への cast() オペレーションが SQL 標準に準拠するようになりました。例えば、`cast (double '100000000000000000000000000000000' as decimal(38))` はこれまで `100000000000000005366162204393472` を返していましたが、今後は `100000000000000000000000000000000000` を返します。
- JOIN ... USING – `JOIN ... USING` が 標準 SQL セマンティクスに準拠するようになりました。これまで、`JOIN ... USING` では列内のテーブル名を修飾することが必要で、出力に両方のテーブルからの列が存在しました。今後はテーブルの修飾が無効になり、列は出力に 1 つしか存在しなくなります。
- ROW 型のリテラルが排除されました – ROW 型のリテラル形式 `ROW<int, int>(1, 2)` はサポートされなくなりました。その代わりに構文 `ROW(1 int, 2 int)` を使用してください。
- グループ化された集計セマンティクス – グループ化された集計は等価セマンティクスではなく `IS NOT DISTINCT FROM` セマンティクスを使用します。今後は、NaN 浮動小数点値でのグループ化において、グループ化された集計が正しい結果を返し、パフォーマンスが改善されます。null が含まれるマップ、リスト、および行型のグループ化がサポートされます。

- 引用符付きの型は使用できなくなりました – ANSI SQL 標準に従い、データ型を引用符で囲むことができなくなりました。例えば、`SELECT "date" '2020-02-02'` は有効なクエリではなくなります。その代わりに、構文 `SELECT date '2020-02-02'` を使用してください。
- 匿名化された行フィールドへのアクセス – 匿名化された行フィールドに構文 `[.field0, .field1, ...]` を使用してアクセスすることができなくなりました。
- 複雑なグループ化操作 – 複雑なグループ化操作である `GROUPING SETS`、`CUBE`、および `ROLLUP` では、入力列で構成される式のグループ化がサポートされていません。列名のみが許可されます。

置き換えられた関数

以下の関数はサポートされなくなり、同じ結果を生成する構文に置き換えられました。

- information_schema.__internal_partitions__ – Athena エンジンバージョン 2 では、今後 `__internal_partitions__` の使用はサポートされません。これに相当する構文には、`SELECT * FROM "<table_name>$partitions"` または `SHOW PARTITIONS` を使用してください。詳細については、「特定のテーブルのパーティションのリスト化」を参照してください。
- 置き換えられた地理空間関数 – 名前が変更された地理空間関数のリストについては、「Athena エンジンバージョン 2 における地理空間関数名の変更」を参照してください。

制限

リソース制限によるクエリの失敗が発生しないことを確実にするため、Athena エンジンバージョン 2 に以下の制限が導入されました。これらの制限は、ユーザー設定可能な制限ではありません。

- 結果要素の数 – `min(col, n)`、`max(col, n)`、`min_by(col1, col2, n)`、および `max_by(col1, col2, n)` 関数では、結果要素 n の数が 10,000 以下に制限されます。
- GROUPING SETS – グループ化セット内のスライスの最大数は 2,048 です。
- テキストファイルの最大行長 — テキストファイルのデフォルト最大行長は 200 MB です。
- シーケンス関数の最大結果サイズ – シーケンス関数の最大結果サイズは 50,000 エントリです。例えば、`SELECT sequence(0,45000,1)` は成功しますが、`SELECT sequence(0,55000,1)` は「The result of the sequence function must not have more than 50000 entries」というエラーメッセージを伴って失敗します。この制限は、タイムスタンプを含むシーケンス関数のすべての入力タイプに適用されます。

Athena の SQL リファレンス

Amazon Athena では、データ定義言語 (DDL) とデータ操作言語 (DML) のステートメント、関数、演算子、およびデータ型のサブセットがサポートされています。いくつかの例外を除き、Athena DDL は HiveQL DDL に基づいており、Athena DML は Trino に基づいています。Athena エンジンバージョンの詳細については、「Athena エンジンのバージョニング」を参照してください。

トピック

- Amazon Athena のデータ型
- DML クエリ、関数、および演算子
- DDL ステートメント
- Amazon Athena での SQL クエリに関する考慮事項と制約事項

Amazon Athena のデータ型

`CREATE TABLE` を実行するときは、列名と、各列に含めることができるデータ型を指定します。作成するテーブルは、AWS Glue Data Catalog に保存されます。

他のクエリエンジンとの相互運用性を促進するため、Athena は `CREATE TABLE` などの DDL ステートメントに Apache Hive データ型名を使用します。`SELECT`、`CTAS`、および `INSERT INTO` などの DML クエリの場合、Athena は Trino データ型名を使用します。以下の表は、Athena でサポートされるデータ型を示しています。DDL 型と DML 型が名前、可用性、構文といった点で異なる場合は、個別の列に表示されます。

DDL	DML	説明
BOOLEAN		値は、`true` および `false` です。
TINYINT		2 の補数形式の 8 ビット符号付き整数で、最小値は -2^7、最大値は 2^7-1 です。
SMALLINT		2 の補数形式の 16 ビット符号付き整数で、最小値は -2^{15}、最大値は $2^{15}-1$ です。
INT, INTEGER		2 の補数形式の 32 ビット符号付き整数で、最小値は -2^{31}、最小値は -263、最大値は $2^{31}-1$ です。

DDL	DML	説明
BIGINT		2 の補数形式の 64 ビット符号付き整数で、最小値は -2^{63}、最小値は -263、最大値は 2^{63}-1 です。
FLOAT	REAL	32 ビットの符号付き単精度浮動小数点数です。範囲は、1.402846638528807e-45 から 3.40282348528860e+38 (正または負) です。IEEE 浮動小数点数演算標準 (IEEE 754) に従います。
DOUBLE		64 ビットの符号付き倍精度浮動小数点数です。範囲は、4.94065645841246544e-324d から 1.7976931 3486231570e+308d (正または負) です。IEEE 浮動小数点数演算標準 (IEEE 754) に従います。
DECIMAL(*precision* , *scale*)		*precision* は桁の合計数です。*scale* (オプション) は小数点以下の桁数で、デフォルトは 0 です。たとえば、`decimal(11,5)` 、`decimal(15)` のタイプ定義を使用します。*precision* の最大値は 38 で、*scale* の最大値も 38 です。
CHAR, CHAR(*length*)		固定長の文字データで、char(10) のように、1 から 255 の長さが指定されています。*length* が指定されている場合、文字列は読み取り時に指定された長さで切り捨てられます。基盤となるデータ文字列がそれより長い場合、基盤となるデータ文字列はそのまま変更されません。 詳細については、「CHAR Hive データ型」を参照してください。
STRING	VARCHAR	可変長文字データです。
VARCHAR(*length*)		最大読み取り長が設定された可変長文字データです。文字列は、読み取り時に指定された長さで切り捨てられます。基盤となるデータ文字列がそれより長い場合、基盤となるデータ文字列はそのまま変更されません。
BINARY	VARBINARY	可変長バイナリデータです。

DDL	DML	説明
TIME		ミリ秒精度の時刻です。
利用不可	TIME(*precision*)	特定の精度の時刻です。`TIME(3)` は `TIME` と同等です。
利用不可	TIME WITH TIME ZONE	タイムゾーン内の時刻です。タイムゾーンは、UTC からのオフセットとして指定する必要があります。
DATE		年、月、日を使用したカレンダー日付です。
TIMESTAMP	TIMESTAMP WITHOUT TIME ZONE	カレンダー日付とミリ秒精度の時刻です。
利用不可	TIMESTAMP (*precision*), TIMESTAMP (*precision*) WITHOUT TIME ZONE	カレンダー日付と特定の精度の時刻です。`TIMESTAMP(3)` は `TIMESTAMP` と同等です。
利用不可	TIMESTAMP WITH TIME ZONE	タイムゾーン内のカレンダー日付と時刻です。タイムゾーンは、UTC からのオフセット、IANA タイムゾーン名、または UTC、UT、Z、GMT を使用して指定できます。
利用不可	TIMESTAMP (*precision*) WITH TIME ZONE	タイムゾーン内のカレンダー日付と特定の精度の時刻です。
利用不可	INTERVAL YEAR TO MONTH	1 か月以上の間隔です。
利用不可	INTERVAL DAY TO SECOND	1 秒、1 分、1 時間、または 1 日以上の間隔です。

DDL	DML	説明
ARRAY<*element_type* >	ARRAY[*element_type*]	値の配列です。すべての値は同じ型である必要があります。
MAP<*key_type*, *value_type* >	MAP(*key_type*, *value_type*)	キーを使用して値を検索できるマップです。すべてのキーに同じ値が必要であり、すべての値に同じ値が必要です。
STRUCT<*field_name_1* :*field_type_1* , *field_name_2* :*field_type_2* , ...>	ROW(*field_name_1* *field_type_1* , *field_name_2* *field_type_2* , ...)	名前が付けられたフィールドとその値からなるデータ構造です。
利用不可	JSON	JSON 値型です。これは、JSON オブジェクト、JSON 配列、JSON 番号、JSON 文字列、`true`、`false`、または `null` にすることができます。
利用不可	UUID	UUID (Universally Unique IDentifier) です。
利用不可	IPADDRESS	IPv4 または IPv6 アドレスです。
利用不可	HyperLogLog	これらのデータ型は、近似関数の内部をサポートします。各型の詳細については、Trino ドキュメント内の対応する項目へのリンクを参照してください。
	P4HyperLogLog	
	SetDigest	
	QDigest	
	TDigest	

データ型の例

以下の表は、DML データ型のリテラル例を示しています。

データ型	例
BOOLEAN	`true` `false`
TINYINT	`TINYINT '123'`
SMALLINT	`SMALLINT '123'`
INT, INTEGER	`123456790`
BIGINT	`BIGINT '1234567890'` `2147483648`
REAL	`'123456.78'`
DOUBLE	`1.234`
DECIMAL(*precision* , *scale*)	`DECIMAL '123.456'`
CHAR, CHAR(*length*)	`CHAR 'hello world'`, `CHAR 'hello ''world''!'`
VARCHAR, VARCHAR(*length*)	`VARCHAR 'hello world'`, `VARCHAR 'hello ''world''!'`
VARBINARY	`X'00 01 02'`
TIME, TIME(*precision*)	`TIME '10:11:12'` , `TIME '10:11:12.345'`
TIME WITH TIME ZONE	`TIME '10:11:12.345 -06:00'`
DATE	`DATE '2024-03-25'`
TIMESTAMP, TIMESTAMP WITHOUT TIME ZONE, TIMESTAMP(*precision*), TIMESTAMP(*precision*) WITHOUT TIME ZONE	`TIMESTAMP '2024-03-25 11:12:13'` , `TIMESTAMP '2024-03-25 11:12:13.456'`

データ型	例
TIMESTAMP WITH TIME ZONE, TIMESTAMP (*precision*) WITH TIME ZONE	`TIMESTAMP '2024-03-25 11:12:13.456 Europe/Berlin'`
INTERVAL YEAR TO MONTH	`INTERVAL '3' MONTH`
INTERVAL DAY TO SECOND	`INTERVAL '2' DAY`
ARRAY[*element_type*]	`ARRAY['one', 'two', 'three']`
MAP(*key_type*, *value_type*)	`MAP(ARRAY['one', 'two', 'three'], ARRAY[1, 2, 3])` マップは、キーの配列と値の配列から作成されることに注意してください。

ROW(*field_name_1 field_type_1* , *field_name_2 field_type_2* , ...)	`ROW('one', 'two', 'three')`
	
	この方法で作成された行には列名がないことに注意してください。列名を追加するには、以下の例にあるように、CAST を使用できます。
	
	`CAST(ROW(1, 2, 3) AS ROW(one INT, two INT, three INT))`

JSON	`JSON '{"one":1, "two": 2, "three": 3}'`
UUID	`UUID '12345678-90ab-cdef-1234-567890abcdef'`

IPADDRESS	`IPADDRESS '10.0.0.1'`
	
	`IPADDRESS '2001:db8::1'`

データ型に関する考慮事項

サイズ制限

サイズ制限を指定しないデータ型については、単一の行内にあるすべてのデータに対して 32MB という実用的制限が設定されていることに注意してください。詳細については、「Amazon Athena での SQL クエリに関する考慮事項と制約事項」の Row or column size limitation を参照してください。

CHAR と VARCHAR

CHAR(*n*) 値には、常に *n* の文字数があります。たとえば、CHAR(7) に「abc」をキャストすると、末尾に 4 つのスペースが追加されます。

CHAR 値の比較には、先頭と末尾のスペースが含まれます。

CHAR または VARCHAR に長さが指定されている場合、文字列は読み取り時に指定された長さで切り捨てられます。基盤となるデータ文字列がそれより長い場合、基盤となるデータ文字列はそのまま変更されません。

CHAR または VARCHAR で一重引用符を工スケープするには、追加の一重引用符を使用します。

DML クエリ内の文字列に文字列以外のデータ型をキャストするには、VARCHAR データ型にキャストします。

substr 関数を使用して CHAR データ型から指定された長さの従属文字列を返すには、まず CHAR 値を VARCHAR としてキャストする必要があります。以下の例では、col1 が CHAR データ型を使用します。

```
substr(CAST(col1 AS VARCHAR), 1, 4)
```

DECIMAL

特定の 10 進値を含む行を選択する場合など、SELECT クエリのリテラルとして 10 進値を指定するには、以下の例にあるように、クエリ内で DECIMAL 型を指定し、一重引用符内のリテラルとして 10 進値をリストすることができます。

```
SELECT * FROM my_table
WHERE decimal_value = DECIMAL '0.12'
```

```
SELECT DECIMAL '44.6' + DECIMAL '77.2'
```

タイムスタンプデータの使用

このセクションでは、Athena でのタイムスタンプデータを使用する場合のいくつかの考慮事項について説明します。

> **Note**
>
> Athena エンジンバージョン 2 から Athena エンジンバージョン 3 では、タイムスタンプの処理が多少変わりました。Athena エンジンバージョン 3 で発生する可能性のあるタイムスタンプ関連のエラーと推奨される解決策については、Athena エンジンバージョン 3 リファレンスにある「タイムスタンプの変更」を参照してください。

タイムスタンプデータを Amazon S3 オブジェクトに書き込むためのフォーマット

タイムスタンプデータを Amazon S3 オブジェクトに書き込む際の形式は、使用する列データ型と SerDe ライブラリの両方によって決まります。

- タイプ `DATE` のテーブル列がある場合、Athena はデータの対応する列またはプロパティが ISO 形式 `YYYY-MM-DD` の文字列、または Parquet や ORC のような組み込みの日付型であることを期待します。
- タイプ `TIME` のテーブル列がある場合、Athena はデータの対応する列またはプロパティが ISO 形式 `HH:MM:SS` の文字列、または Parquet や ORC のような組み込みの時間型であることを期待します。
- タイプ `TIMESTAMP` のテーブル列がある場合、、Athena はデータの対応する列またはプロパティが、その形式 `YYYY-MM-DD HH:MM:SS.SSS` の文字列 (日付と時刻の間のスペースに注意)、または Parquet、ORC、Ion のような組み込みの時間型であることを期待します。

> **Note**
>
> OpenCSVSerDe のタイムスタンプは例外であり、ミリ秒単位の精度の UNIX エポックとしてエンコードする必要があります。

時間分割されたデータがレコードのタイムスタンプフィールドと一致することを確認します

データの作成者は、パーティションの値がパーティション内のデータと一致していることを確認する必要があります。たとえば、データに `timestamp` プロパティがあり、Firehose を使用してデータ

を Amazon S3 にロードする場合、Firehose のデフォルトパーティショニングは実経過時間ベースであることから、動的パーティショニングを使用する必要があります。

パーティションキーのデータ型として文字列を使用

パフォーマンス上の理由から、パーティションキーのデータ型として STRING を使用することをお勧めします。Athena では DATE タイプを使用した場合、その形式 YYYY-MM-DD のパーティション値を日付として認識しますが、これによってパフォーマンスが低下する可能性があります。この理由から、パーティションキーには STRING データ型を使用することをお勧めします。

同じく時間分割されているタイムスタンプフィールドのクエリを作成する方法

時間分割されたタイムスタンプフィールドのクエリを作成する方法は、クエリするテーブルの種類によって異なります。

Hive テーブル

Athena で最も一般的に使用されている Hive テーブルでは、クエリエンジンは列とパーティションキーの関係を認識しません。このため、クエリには必ず列とパーティションキーの両方に述語を追加する必要があります。

たとえば、event_time 列と event_date パーティションキーがあり、23:00 から 03:00 までのイベントをクエリしたいとします。この場合、次の例のように、クエリに列とパーティションキーの両方に述語を含める必要があります。

```
WHERE event_time BETWEEN start_time AND end_time
  AND event_date BETWEEN start_time_date AND end_time_date
```

Iceberg テーブルの使用

Iceberg テーブルでは、計算されたパーティション値を使用できるため、クエリが簡単になります。たとえば、Iceberg テーブルが次のような PARTITIONED BY 句で作成されたとします。

```
PARTITIONED BY (event_date month(event_time))
```

この場合、クエリエンジンは event_time 述語の値に基づいてパーティションを自動的に削除します。このため、クエリでは次の例のように event_time の述語を指定するだけです。

```
WHERE event_time BETWEEN start_time AND end_time
```

詳細については、「Iceberg テーブルの作成」を参照してください。

DML クエリ、関数、および演算子

Athena DML クエリエンジンは、基本的に Trino と Presto の構文をサポートし、独自の改良が加えられています。Athena は Trino と Presto のすべての機能をサポートしているわけではありません。詳細については、このセクション、および「考慮事項と制約事項」で特定のステートメントに関するトピックを参照してください。関数の詳細については、「Amazon Athena の関数」を参照してください。Athena エンジンバージョンの詳細については、「Athena エンジンのバージョニング」を参照してください。

DDL ステートメントの詳細については、「DDL ステートメント」を参照してください。サポートされていない DDL ステートメントのリストについては、「サポートされない DDL」を参照してください。

SELECT

0 個以上のテーブルからデータの行を取得します。

> **Note**
>
> このトピックでは、参照用に概要情報を提供します。SELECT と SQL 言語の使用に関する包括的な情報は、このドキュメントでは説明しません。Athena に固有の SQL の使用については、「Amazon Athena での SQL クエリに関する考慮事項と制約事項」および「Amazon Athena を使用した SQL クエリの実行」を参照してください。データベースの作成、テーブルの作成、および Athena のテーブルに対する SELECT クエリの実行の例については、開始を参照してください。

概要

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ OFFSET count [ ROW | ROWS ] ]
```

```
[ LIMIT [ count | ALL ] ]
```

> **Note**
>
> SQL SELECT ステートメントの予約語は、二重引用符で囲む必要があります。詳細については、「SQL SELECT ステートメントの予約キーワードのリスト」を参照してください。

パラメータ

[WITH with_query [, ...]]

WITH を使用すると、ネストされたクエリをフラット化したり、サブクエリを簡素化できます。

WITH 句を使用した再帰クエリの作成は、Athena エンジンバージョン 3 以降でサポートされています。最大再帰深度は 10 です。

WITH 句はクエリの SELECT リストに先行し、SELECT クエリ内で使用する 1 つ以上のサブクエリを定義します。

各サブクエリは、ビュー定義と同じように、一時テーブルを定義します。一時テーブルは FROM 句で参照できます。このテーブルはクエリを実行時のみに使用します。

with_query 構文は次のとおりです。

```
subquery_table_name [ ( column_name [, ...] ) ] AS (subquery)
```

各パラメータの意味は次のとおりです。

- subquery_table_name は、WITH 句のサブクエリの結果を定義する一時テーブルの一意な名前です。各 subquery には、FROM 句で参照可能なテーブル名を付ける必要があります。
- column_name [, ...] は、出力列名の省略可能なリストです。列名の数は、subquery で定義した列数以下でなければなりません。
- subquery は、任意のクエリステートメントです。

[ALL | DISTINCT] select_expression

select_expression は、選択する行を決定します。select_expression には、次のいずれかの形式が使用されます。

```
expression [ [ AS ] column_alias ] [, ...]
```

```
row_expression.* [ AS ( column_alias [, ...] ) ]
```

```
relation.*
```

```
*
```

- expression [[AS] column_alias] 構文は出力列を指定します。オプションの [AS] column_alias 構文は、出力の列に使用されるカスタム見出し名を指定します。
- row_expression.* [AS (column_alias [, ...])] の場合、row_expression は データ型 ROW の任意の式です。行のフィールドは、結果に含まれる出力列を定義します。
- relation.* の場合、relation の列が結果に含まれます。この構文では列のエイリアスは使用できません。
- アスタリスク * は、すべての列を結果セットに含めることを指定します。
- 結果セット内の列の順序は、select 式による指定の順序と同じです。select 式が複数の列を返す場合、列の順序はソースリレーションまたは行タイプの式で使用されている順序に従います。
- 列のエイリアスを指定すると、そのエイリアスは既存の列または行のフィールド名よりも優先されます。select 式に列名がない場合は、インデックスが 0 の匿名の列名 (_col0、_col1、_col2, ...) が出力に表示されます。
- ALL はデフォルトです。ALL は、それを省略した場合と同じように扱われます。すべての列のすべての行が選択され、重複も含まれます。
- DISTINCT は、列に重複する値が含まれているときに、個別の値のみを返すために使用します。

FROM from_item [, ...]

クエリへの入力を示します。from_item は、以下に示すように、ビュー、結合コンストラクト、サブクエリのいずれかです。

from_item は以下のいずれかです。

- table_name [[AS] alias [(column_alias [, ...])]]

  table_name は行の選択元であるターゲットテーブルの名前であり、alias は SELECT ステートメントの出力に渡す名前です。column_alias は指定した alias の列を定義します。

-または-

- join_type from_item [ON join_condition | USING (join_column [, ...])]

  join_type は以下のいずれかです。

  - [INNER] JOIN
  - LEFT [OUTER] JOIN
  - RIGHT [OUTER] JOIN
  - FULL [OUTER] JOIN
  - CROSS JOIN
  - ON join_condition | USING (join_column [, ...])。join_condition では、複数のテーブルにおいて結合キーの列名を指定できます。join_column を使用するには、join_column が両方のテーブルに存在している必要があります。

[WHERE condition]

指定した condition に従って結果をフィルタリングします。通常、condition には次の構文が含まれています。

```
column_name operator value [[[AND | OR] column_name operator value] ...]
```

*###*は、比較演算子 =、>、<、>=、<=、<>、!= のいずれかになります。

次のサブクエリ式も、WHERE 句で使用できます。

- [NOT] BETWEEN *integer_A* AND *integer_B* - 次の例のように、2 つの整数間の範囲を指定します。列のデータ型が varchar の場合、最初に列を整数にキャストする必要があります。

  ```
  SELECT DISTINCT processid FROM "webdata"."impressions"
  WHERE cast(processid as int) BETWEEN 1500 and 1800
  ORDER BY processid
  ```

- [NOT] LIKE *value* - 指定したパターンを検索します。次の例のように、パーセント記号 (%) をワイルドカード文字として使用します。

  ```
  SELECT * FROM "webdata"."impressions"
  WHERE referrer LIKE '%.org'
  ```

- [NOT] IN (*value*[, *value*[, ...]) - 次の例のように、列で使用できる値のリストを指定します。

```
SELECT * FROM "webdata"."impressions"
WHERE referrer IN ('example.com','example.net','example.org')
```

[GROUP BY [ALL | DISTINCT] grouping_expressions [, ...]]

SELECT ステートメントの出力を、一致する値を持つ行に分割します。

ALL と DISTINCT は、重複したグループ化セットごとに個別の出力行を生成するかどうかを決定します。省略すると、ALL が使用されます。

grouping_expressions では、複雑なグループ化オペレーションを実行できます。複雑なグループ化オペレーションを使用して、複数の列セットの集計を必要とする分析を単一のクエリで実行できます。

grouping_expressions 要素には、SUM、AVG、COUNT など、入力列に対して実行される任意の関数を指定できます。

GROUP BY 式は、SELECT ステートメントの出力に表示されない入力列名で出力をグループ化できます。

すべての出力式は、集計関数であるか、GROUP BY 句に存在する列であることが必要です。

単一のクエリを使用して、複数の列セットの集計を必要とする分析を実行できます。

Athena は、GROUPING SETS、CUBE、および ROLLUP を使用する複雑な集計をサポートしています。GROUP BY GROUPING SETS で、グループ化する列の複数のリストを指定します。GROUP BY CUBE で、特定の列のセットに対して、すべての可能なグループ化セットを生成します。GROUP BY ROLLUP で、特定の列のセットに対して、すべての可能な小計を生成します。複雑なグループ化オペレーションでは、入力列で構成される式でのグループ化がサポートされていません。列名のみが許可されます。

通常、UNION ALL を使用しても、これらの GROUP BY オペレーションと同じ結果を達成できます。ただし、GROUP BY を使用するクエリでは、データの読み取りが 1 回で済むという利点があります。UNION ALL は基となるデータを 3 回読み取るため、データソースが変わりやすい場合は、不整合な結果が生成されることがあります。

[HAVING condition]

集計関数と GROUP BY 句で使用します。どのグループを選択するかを制御します。condition を満たさないグループは排除されます。このフィルタ処理は、グループや集計の計算後に行われます。

[{ UNION | INTERSECT | EXCEPT } [ALL | DISTINCT] union_query]]

UNION、INTERSECT、および EXCEPT は、複数の SELECT ステートメントの結果を 1 つのクエリに結合します。ALL または DISTINCT は、最終的な結果セットに含められる行の一意性を制御します。

UNION は、最初のクエリから得られた行と、2 番目のクエリから得られた行を結合します。重複を排除するため、UNION はメモリを消費するハッシュテーブルを構築します。パフォーマンス向上のため、クエリに重複を排除する必要がない場合は UNION ALL の使用を検討してください。複数の UNION 句は左から右に処理されます。ただし、括弧を使用して処理の順序を明示的に定義することもできます。

INTERSECT は、最初のクエリと 2 番目のクエリ両方の結果に存在する行のみを返します。

EXCEPT は、最初のクエリの結果からの行を返し、2 番目のクエリで見つかった行は排除されます。

ALL は、行が同一の場合でも、すべての行が含まれる処理を実行します。

DISTINCT は、統合された結果セットに一意の行のみが含まれるようにします。

[ORDER BY expression [ASC | DESC] [NULLS FIRST | NULLS LAST] [, ...]]

結果セットを 1 つ以上の出力 expression でソートします。

句に複数の式が含まれている場合、結果セットは最初の expression に従ってソートされます。次に、最初の式で一致した値がある行に 2 番目の expression が適用されます。以下、同様です。

各 expression では、SELECT の出力列を指定するか、出力列の位置を 1 から始まる序数で指定できます。

ORDER BY は、GROUP BY または HAVING 句の後で、最後のステップとして評価されます。ASC と DESC は、結果のソートを昇順にするか、降順にするかを決定します。

デフォルトの null 順序は、昇順または降順のソート順に関係なしに NULLS LAST です。

[OFFSET count [ROW | ROWS]]

OFFSET 句を使用して、結果セットの先頭の行をいくつか破棄します。ORDER BY 句が存在する場合、OFFSET 句はソートされた結果セットに対して評価されます。スキップされた行が破棄された後もセットはソートされたままになります。クエリに ORDER BY 句がない場合、破棄され

る行は任意です。OFFSET で指定したカウントが結果セットのサイズに等しいかそれを超える場合、最終結果は空になります。

LIMIT [count | ALL]

結果セットの行数を count に制限します。LIMIT ALL は LIMIT 句を省略した場合と同じです。クエリに ORDER BY 句がない場合は、任意の結果になります。

TABLESAMPLE [BERNOULLI | SYSTEM] (percentage)

サンプリング方法に基づいてテーブルから行を選択する演算子 (オプション) です。

BERNOULLI は、percentage の確率でテーブルサンプルに存在する各行を選択します。テーブルのすべての物理ブロックがスキャンされ、サンプルの percentage とランタイムに計算されるランダム値の比較に基づいて、特定の行がスキップされます。

SYSTEM では、テーブルがデータの論理セグメントに分割され、この詳細度でテーブルがサンプリングされます。

特定のセグメントのすべての行が選択されるか、サンプルの percentage とランタイムに計算されたランダム値の比較に基づいて当該セグメントがスキップされます。SYSTEM サンプリングはコネクタに依存します。この方法では、独立したサンプリング確率は保証されません。

[UNNEST (array_or_map) [WITH ORDINALITY]]

配列またはマップをリレーションに展開します。配列は単一の列に展開されます。マップは 2 つの列 (キー、値) に展開されます。

UNNEST に複数の引数を使用できます。これらの引数は、複数の列に展開され、各列の行数は最大の基数引数と同じになります。

その他の列には NULL が埋め込まれます。

WITH ORDINALITY 句は、序数列を末尾に追加します。

通常、UNNEST は JOIN と一緒に使用し、JOIN の左側のリレーションの列を参照できます。

Amazon S3 内にあるソースデータのファイルの場所の取得

テーブル行にあるデータの Amazon S3 ファイルの場所を確認するには、以下の例にあるように、SELECT クエリで "$path" を使用できます。

```
SELECT "$path" FROM "my_database"."my_table" WHERE year=2019;
```

このクエリは以下のような結果を返します。

```
s3://DOC-EXAMPLE-BUCKET/datasets_mytable/year=2019/data_file1.json
```

テーブル内にあるデータについて S3 ファイル名パスのソートされた一意のリストを返すには、以下の例にあるように、SELECT DISTINCT と ORDER BY を使用できます。

```
SELECT DISTINCT "$path" AS data_source_file
FROM sampledb.elb_logs
ORDER By data_source_file ASC
```

パスなしでファイル名のみを返すには、以下の例にあるように、"$path" をパラメータとして regexp_extract 関数に渡すことができます。

```
SELECT DISTINCT regexp_extract("$path", '[^/]+$') AS data_source_file
FROM sampledb.elb_logs
ORDER By data_source_file ASC
```

特定のファイルからのデータを返すには、以下の例にあるように、WHERE 句でそのファイルを指定します。

```
SELECT *,"$path" FROM my_database.my_table WHERE "$path" = 's3://DOC-EXAMPLE-BUCKET/
my_table/my_partition/file-01.csv'
```

詳細と例については、ナレッジセンターの記事、「Athena テーブルの行の Amazon S3 ソースファイルを表示する方法を教えてください」を参照してください。

> **Note**
>
> Athena では、Hive または Iceberg の非表示のメタデータ
> 列 $bucket、$file_modified_time、$file_size、および $partition は、ビューでサポートされていません。

一重引用符のエスケープ

一重引用符をエスケープするには、以下の例のように、その前に別の一重引用符を付けます。これを二重引用符と混同しないでください。

```
Select 'O''Reilly'
```

結果

```
O'Reilly
```

追加リソース

Athena で SELECT ステートメントを使用する方法の詳細については、以下のリソースを参照してください。

詳細情報の内容	参照先
Athena でクエリを実行する	Amazon Athena を使用した SQL クエリの実行
SELECT を使用してテーブルを作成する	クエリ結果からのテーブルの作成 (CTAS)
SELECT クエリからのデータを別のテーブルに挿入する	INSERT INTO
SELECT ステートメントで組み込みの関数を使用する	Amazon Athena の関数
SELECT ステートメントでユーザー定義の関数を使用する	ユーザー定義関数を使用したクエリ
データカタログのメタデータをクエリする	AWS Glue Data Catalog のクエリ

INSERT INTO

ソーステーブルで実行される SELECT クエリステートメント、またはステートメントの一部として提供される VALUES のセットに基づいて、送信先テーブルに新しい行を挿入します。ソーステーブルが CSV や JSON などの形式の基盤データに基づくもので、宛先テーブルが Parquet や ORC など別の形式に基づいている場合は、INSERT INTO クエリを使用して、選択したデータを宛先テーブルの形式に変換できます。

考慮事項と制約事項

Athena で INSERT クエリを使用するときは、以下の点を考慮してください。

- Amazon S3 で暗号化された基盤データがあるテーブルに対して INSERT クエリを実行する場合、INSERT クエリが書き込む出力ファイルはデフォルトで暗号化されません。暗号化されたデータを含むテーブルに挿入する場合は、INSERT クエリの結果を暗号化することをお勧めします。

  コンソールを使用したクエリ結果の暗号化の詳細については、「Amazon S3 に保存された Athena のクエリ結果の暗号化」を参照してください。AWS CLI または Athena API を使用した暗号化を有効にするには、StartQueryExecution アクションの EncryptionConfiguration プロパティを使用して、要件に沿った Amazon S3 暗号化オプションを指定します。
- INSERT INTO ステートメントの場合、予想されるバケット所有者の設定は、Amazon S3 内の送信先テーブルのロケーションには適用されません。予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。詳細については、「Athena コンソールを使用したクエリ結果の場所の指定」を参照してください。
- ACID 準拠の INSERT INTO ステートメントについては、Iceberg テーブルデータの更新 の INSERT INTO セクションを参照してください。

サポートされる形式と SerDes

次の形式と SerDes を使用して、データから作成されたテーブルに対して INSERT クエリを実行できます。

データ形式	SerDe
Avro	org.apache.hadoop.hive.serde2.avro.AvroSerDe
Ion	com.amazon.ionhiveserde.IonHiveSerDe
JSON	org.apache.hive.hcatalog.data.JsonSerDe
ORC	org.apache.hadoop.hive.ql.io.orc.OrcSerde
Parquet	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
テキストファイル	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe **Note** CSV、TSV、およびカスタム区切りファイルがサポートされています。

バケット化されたテーブルはサポートされていません

INSERT INTO はバケット化されたテーブルではサポートされていません。詳細については、「Athena におけるパーティション化とバケット化」を参照してください。

フェデレーティッドクエリはサポートされていません

横串検索では、INSERT INTO はサポートされていません。これを試みた場合、「This operation is currently not supported for external catalogs (この操作は現在、外部カタログではサポートされていません)」というエラーメッセージが表示されることがあります。横串検索の詳細については、「Amazon Athena 横串検索の使用」を参照してください。

パーティション

INSERT INTO または CREATE TABLE AS SELECT クエリでパーティションを使用するときは、このセクションのポイントを考慮してください。

制限

この INSERT INTO ステートメントは、送信先テーブルへの最大 100 個のパーティションの書き込みをサポートします。100 個を超えるパーティションを持つテーブルから SELECT 句を使用して行を追加する場合、SELECT クエリは、100 個以下のパーティションに制限されない限り失敗します。

この制限を回避する方法については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。

列の順序

INSERT INTO または CREATE TABLE AS SELECT ステートメントは、SELECT ステートメントで射影された列のリストの最後の列がパーティションされた列であることを期待します。

ソーステーブルがパーティションされていない場合、または宛先テーブルとは異なる列でパーティションされている場合、INSERT INTO *destination_table* SELECT * FROM *source_table* のようなクエリは、ソーステーブルの最後の列の値が宛先テーブルのパーティション列の値であると見なします。パーティションされていないテーブルからパーティションテーブルを作成するときは、この点に留意してください。

リソース

パーティションでの INSERT INTO の使用に関する詳細については、以下のリソースを参照してください。

- パーティションされたデータをパーティションテーブルに挿入する方法については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。
- パーティションされていないデータをパーティションテーブルに挿入する方法については、「ETL およびデータ分析での CTAS および INSERT INTO の使用」を参照してください。

Amazon S3 に書き込まれるファイル

Athena は、INSERT コマンドの結果として、Amazon S3 のソースデータの場所にファイルを書き込みます。INSERT オペレーションごとに、既存のファイルを追加するのではなく、新しいファイルが作成されます。ファイルの場所は、テーブルの構造と SELECT クエリ (存在する場合) に応じて異なります。Athena は、INSERT クエリごとにデータマニフェストファイルを生成します。マニフェストは、クエリが書き込んだファイルを追跡します。これは、Amazon S3 にある Athena のクエリ結果の場所に保存されます。詳細については、「クエリ出力ファイルの識別」を参照してください。

トランザクション集約型更新の回避

INSERT INTO を使用して Amazon S3 内のテーブルに行を追加しても、Athena は既存のファイルの書き換えや変更を行いません。その代わりに、Athena は 1 つ、または複数の新しいファイルとして行を書き込みます。小さなファイルが多数存在するテーブルはクエリパフォーマンスの低下につながり、PutObject や GetObject などの書き込みおよび読み取り操作は Amazon S3 のコスト増加につながるため、INSERT INTO を使用するときは以下のオプションを検討してください。

- 行の大規模なバッチに対する INSERT INTO 操作の実行頻度を減らす。
- データインジェスト量が多い場合は、Amazon Data Firehose といったサービスの使用を検討する。
- もとより INSERT INTO を使用しないようにする。その代わりに、より大きなファイルに行を蓄積し、Athena が行をクエリできる Amazon S3 に直接アップロードします。

孤立したファイルの検索

CTAS または INSERT INTO ステートメントが失敗すると、孤立したデータがデータの場所に残され、後続のクエリで読み取られる場合があります。検査または削除する孤立したファイルを見つけるには、Athena に用意されているデータマニフェストファイルを使用して、書き込まれるファイルのリストを追跡できます。詳細については、「クエリ出力ファイルの識別」および「DataManifestLocation」を参照してください。

INSERT INTO...SELECT

1 つのテーブル、source_table に対して実行するクエリを指定します。これにより、2 番目のテーブル、destination_table に挿入する行が決定されます。SELECT クエリが source_table での列を指定する場合、その列は destination_table の列と正確に一致する必要があります。

SELECT クエリの詳細については、「SELECT」を参照してください。

概要

```
INSERT INTO destination_table
SELECT select_query
FROM source_table_or_view
```

例

vancouver_pageviews テーブル内のすべての行を選択し、canada_pageviews テーブルに挿入します。

```
INSERT INTO canada_pageviews
SELECT *
FROM vancouver_pageviews;
```

date 列の値が 2019-07-01 ~ 2019-07-31 の vancouver_pageviews テーブル内の行のみを選択し、canada_july_pageviews に挿入します。

```
INSERT INTO canada_july_pageviews
SELECT *
FROM vancouver_pageviews
WHERE date
    BETWEEN date '2019-07-01'
        AND '2019-07-31';
```

cities_world テーブルの city と state 列で、country 列の値が usa の行だけを選択して、cities_usa テーブルの city と state 列に挿入します。

```
INSERT INTO cities_usa (city,state)
SELECT city,state
FROM cities_world
```

```
    WHERE country='usa'
```

INSERT INTO...VALUES

列と値を指定して、既存のテーブルに行を挿入します。指定された列と関連するデータ型は、宛先テーブルの列およびデータ型と正確に一致する必要があります。

> **Important**
>
> Athena は INSERT オペレーションごとにファイルを生成するため、VALUES を使用した行の挿入は推奨されません。これにより、多数の小さなファイルが作成され、テーブルのクエリパフォーマンスが低下する可能性があります。INSERT クエリが作成するファイルを識別するには、データマニフェストファイルを調べます。詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。

概要

```
INSERT INTO destination_table [(col1,col2,...)]
VALUES (col1value,col2value,...)[,
       (col1value,col2value,...)][,
       ...]
```

例

次の例では、都市テーブルには、id、city、state、state_motto の 3 つの列があります。id 列は INT 型で、他のすべての列は VARCHAR 型です。

cities テーブルに 1 つの行を挿入し、すべての列の値を指定します。

```
INSERT INTO cities
VALUES (1,'Lansing','MI','Si quaeris peninsulam amoenam circumspice')
```

cities テーブルに 2 行を挿入します。

```
INSERT INTO cities
VALUES (1,'Lansing','MI','Si quaeris peninsulam amoenam circumspice'),
       (3,'Boise','ID','Esto perpetua')
```

DELETE

Apache Iceberg テーブルの行を削除します。DELETE はトランザクション型で、Apache Iceberg テーブルでのみサポートされています。

概要

Iceberg テーブルから行を削除するには、次の構文を使用します。

```
DELETE FROM [db_name.]table_name [WHERE predicate]
```

詳細と例については、Iceberg テーブルデータの更新 の DELETE セクションを参照してください。

UPDATE

Apache Iceberg テーブルの行を更新します。UPDATE はトランザクションであり、Apache Iceberg テーブルでのみサポートされています。

概要

Iceberg テーブルの行を更新するには、次の構文を使用します。

```
UPDATE [db_name.]table_name SET xx=yy[,...] [WHERE predicate]
```

詳細と例については、Iceberg テーブルデータの更新 の UPDATE セクションを参照してください。

MERGE INTO

Apache Iceberg テーブルに条件付きで行を更新、削除、または挿入します。単一のステートメントで、更新、削除、挿入のアクションを組み合わせることができます。

> **Note**
>
> MERGE INTO はトランザクションで、Athena エンジンバージョン 3 の Apache Iceberg テーブルでのみサポートされています。

概要

Iceberg テーブルから条件付きで行を更新、削除、または挿入するには、次の構文を使用します。

```
MERGE INTO target_table [ [ AS ]  target_alias ]
USING { source_table | query } [ [ AS ] source_alias ]
ON search_condition
when_clause [...]
```

when_clause は次のいずれかです。

```
WHEN MATCHED [ AND condition ]
    THEN DELETE
```

```
WHEN MATCHED [ AND condition ]
    THEN UPDATE SET ( column = expression [, ...] )
```

```
WHEN NOT MATCHED [ AND condition ]
    THEN INSERT (column_name[, column_name ...]) VALUES (expression, ...)
```

MERGE は、異なる MATCHED の条件を持つ任意の数の WHEN 句をサポートします。条件句は、MATCHED ステートと一致条件によって選択された最初の WHEN 句の DELETE、UPDATE または INSERT 演算を実行します。

ソース行ごとに、WHEN 句が順番に処理されます。最初に一致した WHEN 句のみが実行されます。後続の句は無視されます。1 つのターゲットテーブル行が複数のソース行と一致すると、ユーザーエラーが発生します。

ソース行がどの WHEN 句にも一致せず、WHEN NOT MATCHED 句がない場合、そのソース行は無視されます。

UPDATE 演算を含む WHEN 句では、列値の式はターゲットまたはソースの任意のフィールドを参照できます。NOT MATCHED の場合、INSERT の式はソースの任意のフィールドを参照できます。

例

次の例では、行が最初のテーブルに存在しない場合、2 番目のテーブルの行を最初のテーブルにマージしています。VALUES 句にリストされている列の先頭には、ソーステーブルのエイリアスを付ける必要があることに注意してください。INSERT 句にリストされているターゲット列には、このプレジックスを付けないでください。

```
MERGE INTO iceberg_table_sample as ice1
USING iceberg2_table_sample as ice2
```

```
ON ice1.col1 = ice2.col1
WHEN NOT MATCHED
THEN INSERT (col1)
      VALUES (ice2.col1)
```

その他の MERGE INTO 例については、「Iceberg テーブルデータの更新」を参照してください。

OPTIMIZE

関連する削除ファイルのサイズと数に基づいて、データファイルをより最適化されたレイアウトに書き換えることにより、Apache Iceberg テーブル内の行を最適化します。

> **Note**
>
> OPTIMIZE は Apache Iceberg テーブルでのみサポートされているトランザクションです。

構文

次の構文の概要は、Iceberg テーブルのデータレイアウトを最適化する方法を示しています。

```
OPTIMIZE [db_name.]table_name REWRITE DATA USING BIN_PACK
  [WHERE predicate]
```

> **Note**
>
> WHERE 句の##にはパーティション列のみを使用できます。非パーティション列を指定すると、クエリは失敗します。

圧縮アクションは、書き換えプロセス中にスキャンしたデータ量の分だけ課金されます。REWRITE DATA アクションは、述語を使用して、一致する行を含むファイルを選択します。ファイル内のいずれかの行が述語と一致する場合、ファイルは最適化のために選択されます。したがって、WHERE 句を指定することで、圧縮オペレーションの影響を受けるファイルの数を制御できます。

圧縮のプロパティの設定

圧縮用に選択するファイルのサイズと、圧縮後の結果のファイルサイズを制御するには、テーブルプロパティパラメータを使用します。ALTER TABLE SET PROPERTIES コマンドを使用して、関連するテーブルプロパティを設定できます。

追加リソース

Iceberg テーブルの最適化

VACUUM

VACUUM ステートメントは、不要になったデータファイルを削除することにより、Apache Iceberg テーブルのテーブルメンテナンスを行います。

> **Note**
>
> VACUUM はトランザクションで、Athena エンジンバージョン 3 の Apache Iceberg テーブルでのみサポートされています。

VACUUM ステートメントを Iceberg テーブルで実行すると、不要になったデータファイルを削除し、メタデータのサイズとストレージの消費量を削減できます。VACUUM ステートメントは Amazon S3 に API 呼び出しを行うため、Amazon S3 への関連するリクエストに対して料金がかかることに注意してください。

> **Warning**
>
> スナップショット有効期限切れオペレーションを実行すると、期限切れのスナップショットへのタイムトラベルができなくなります。

概要

Iceberg テーブルで不要になったデータファイルを削除するには、次の構文を使用します。

```
VACUUM [database_name.]target_table
```

アンダースコアで始まる名前 (例: _mytable) を持つテーブルで VACUUM を実行するには、次の例にあるようにテーブル名をバックティックで囲みます。テーブル名の前にデータベース名を付ける場合は、データベース名をバックティックで囲まないでください。バックティックの代わりに二重引用符を使用しても機能しないことに注意してください。

この動作は VACUUM に固有の動作です。CREATE および INSERT INTO ステートメントでは、アンダースコアで始まるテーブル名にバックティックは必要ありません。

```
VACUUM `_mytable`
VACUUM my_database.`_mytable`
```

また、VACUUM は、Iceberg データが Amazon S3 バケットではなく Amazon S3 フォルダに格納されることを想定していることにも注意してください。例えば、Iceberg データが s3://DOC-EXAMPLE-BUCKET/myicebergfolder/ ではなく s3://DOC-EXAMPLE-BUCKET/ に格納されている場合、VACUUM ステートメントは「GENERIC_INTERNAL_ERROR: ファイルシステムロケーション: s3://DOC-EXAMPLE-BUCKET にパスがありません」というエラーで失敗します。

実行されたオペレーション

VACUUM では以下のオペレーションを実行します。

- vacuum_max_snapshot_age_seconds テーブルプロパティで指定された時間よりも古いスナップショットを削除します。デフォルトでは、このプロパティは 432,000 秒 (5 日間) に設定されています。
- 保持期間に含まれていないスナップショットのうち、vacuum_min_snapshots_to_keep テーブルプロパティで指定された数を超えているスナップショットを削除します。デフォルト は 1 です。

  これらのテーブルプロパティは CREATE TABLE ステートメントで指定できます。テーブルを作成した後、ALTER TABLE SET PROPERTIES ステートメントを使用してテーブルを更新できます。
- スナップショットを削除した結果としてアクセスできなかったメタデータおよびデータファイルをすべて削除します。vacuum_max_metadata_files_to_keep テーブルプロパティを設定すると、保持する古いメタデータファイルの数を設定できます。デフォルト値は 100 です。
- vacuum_max_snapshot_age_seconds テーブルプロパティで指定された時間より古い孤立ファイルを削除します。孤立ファイルとは、テーブルのデータディレクトリにある、テーブルステートの一部ではないファイルです。

Athena で Apache Iceberg テーブルを作成および管理する方法については、「Iceberg テーブルの作成」および「Iceberg テーブルの管理」を参照してください。

Athena での EXPLAIN および EXPLAIN ANALYZE の使用

EXPLAIN ステートメントは、指定された SQL ステートメントの論理実行または分散実行プランを表示、または SQL ステートメントを検証します。結果はテキスト形式での出力、またはグラフへのレンダリングのためのデータ形式での出力が可能です。

> **Note**
>
> EXPLAIN 構文を使用せずに、Athena コンソールでクエリの論理プランと分散プランのグラフィック表現を表示できます。詳細については、「SQL クエリの実行プランの表示」を参照してください。

EXPLAIN ANALYZE ステートメントでは、指定した SQL ステートメントの分散実行プランと、SQL クエリ内の各オペレーションに関する計算コストの両方を表示します。結果はテキスト形式または JSON 形式で出力することができます。

考慮事項と制約事項

Athena の EXPLAIN および EXPLAIN ANALYZE ステートメントには、以下の制限があります。

- EXPLAIN クエリはデータをスキャンしないため、Athena はそれらに対する料金を請求しません。ただし、EXPLAIN クエリはテーブルメタデータの取得のために AWS Glue を呼び出すので、呼び出し回数が Glue の無料利用枠制限を超える場合に料金が発生する可能性があります。
- この理由は、EXPLAIN ANALYZE クエリが実行されデータをスキャンすると、Athena は、そのスキャンされたデータ量に対して課金をするためです。
- Lake Formation で定義されている行またはセルのフィルタリング情報、およびクエリの統計情報は、EXPLAIN および EXPLAIN ANALYZE の出力には表示されません。

EXPLAIN 構文

```
EXPLAIN [ ( option [, ...]) ] statement
```

option は以下のいずれかにすることができます。

```
FORMAT { TEXT | GRAPHVIZ | JSON }
TYPE { LOGICAL | DISTRIBUTED | VALIDATE | IO }
```

FORMAT オプションを指定しない場合、出力はデフォルトで TEXT 形式となります。IO 型は、クエリが読み込むテーブルとスキーマに関する情報を提供します。IO は Athena エンジンバージョン 2 のみでサポートされており、JSON 形式のみで返すことができます。

EXPLAIN ANALYZE 構文

EXPLAIN からの出力に加えて、EXPLAIN ANALYZE では、CPU 使用率、入力された行数、出力された行数など、指定したクエリの実行時に関する統計情報も出力します。

```
EXPLAIN ANALYZE [ ( option [, ...]) ] statement
```

option は以下のいずれかにすることができます。

```
FORMAT { TEXT | JSON }
```

FORMAT オプションを指定しない場合、出力はデフォルトで TEXT 形式となります。EXPLAIN ANALYZE によるすべてのクエリは DISTRIBUTED なので、EXPLAIN ANALYZE では TYPE オプションを使用することはできません。

*******は以下のいずれかになります。

```
SELECT
CREATE TABLE AS SELECT
INSERT
UNLOAD
```

EXPLAIN での例

以下の EXPLAIN に関する例では、まず端的なものを示し、その後に、複合型のものへと進みます。

EXPLAIN の例 1: テキスト形式でクエリプランを表示するために、EXPLAIN ステートメントを使用する

次の例の EXPLAIN では、Elastic Load Balancing ログでの SELECT クエリについて、その実行プランを表示します 使用される形式はデフォルトのテキスト出力です。

```
EXPLAIN
SELECT
   request_timestamp,
   elb_name,
   request_ip
FROM sampledb.elb_logs;
```

結果

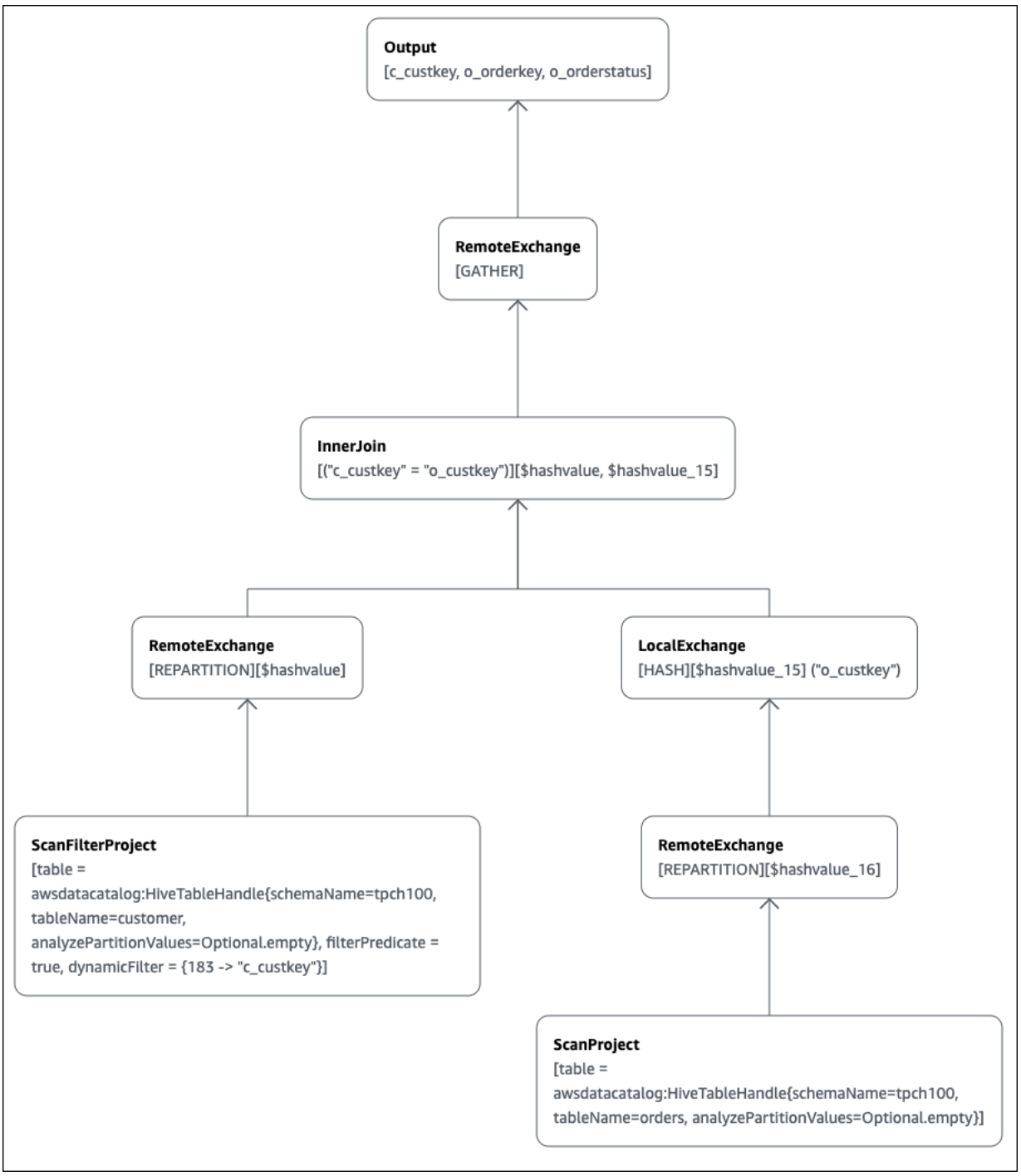
```
- Output[request_timestamp, elb_name, request_ip] => [[request_timestamp, elb_name,
 request_ip]]
    - RemoteExchange[GATHER] => [[request_timestamp, elb_name, request_ip]]
        - TableScan[awsdatacatalog:HiveTableHandle{schemaName=sampledb,
 tableName=elb_logs,
analyzePartitionValues=Optional.empty}] => [[request_timestamp, elb_name, request_ip]]
                LAYOUT: sampledb.elb_logs
                request_ip := request_ip:string:2:REGULAR
                request_timestamp := request_timestamp:string:0:REGULAR
                elb_name := elb_name:string:1:REGULAR
```

EXPLAIN 例 2: クエリプランをグラフ化する

Athena コンソールを使用してクエリプランをグラフ化できます。次のような SELECT ステートメントを Athena クエリエディタに入力し、[EXPLAIN] を選択します。

```
SELECT
      c.c_custkey,
      o.o_orderkey,
      o.o_orderstatus
   FROM tpch100.customer c
   JOIN tpch100.orders o
       ON c.c_custkey = o.o_custkey
```

Athena クエリエディタの [Explain] ページが開き、クエリの分散プランと論理プランが表示されます。次のグラフは、この例の論理プランを示します。



Important

現在、一部のパーティションフィルターは、Athena がクエリに適用しても、ネストされたオペレーターツリーグラフに表示されない場合があります。このようなフィルターの効果を検証するには、クエリで `EXPLAIN` または `EXPLAIN ANALYZE` を実行し、結果を表示します。

Athena コンソールでクエリプランのグラフ機能を使用する方法の詳細については、「SQL クエリの実行プランの表示」を参照してください。

EXPLAIN の例 3: EXPLAIN ステートメントを使用してパーティションプルーニングを検証する

パーティションされたキーにフィルタリング述語を使用してパーティションテーブルをクエリする場合、クエリエンジンはパーティションされたキーにこの述語を適用して、読み込むデータの量を減らします。

以下の例では、パーティションテーブルでの SELECT クエリについてパーティションプルーニングを検証するために EXPLAIN クエリを使用します。まず、CREATE TABLE ステートメントが tpch100.orders_partitioned テーブルを作成します。テーブルは、列 o_orderdate でパーティションされます。

```
CREATE TABLE `tpch100.orders_partitioned`(
  `o_orderkey` int,
  `o_custkey` int,
  `o_orderstatus` string,
  `o_totalprice` double,
  `o_orderpriority` string,
  `o_clerk` string,
  `o_shippriority` int,
  `o_comment` string)
PARTITIONED BY (
  `o_orderdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/<your_directory_path>/'
```

SHOW PARTITIONS コマンドによる表示にあるように、tpch100.orders_partitioned テーブルには o_orderdate にいくつかのパーティションがあります。

```
SHOW PARTITIONS tpch100.orders_partitioned;

o_orderdate=1994
o_orderdate=2015
o_orderdate=1998
```

```
o_orderdate=1995
o_orderdate=1993
o_orderdate=1997
o_orderdate=1992
o_orderdate=1996
```

以下の EXPLAIN クエリは、指定された SELECT ステートメントでパーティションプルーニングを検証します。

```
EXPLAIN
SELECT
   o_orderkey,
   o_custkey,
   o_orderdate
FROM tpch100.orders_partitioned
WHERE o_orderdate = '1995'
```

結果

```
Query Plan
- Output[o_orderkey, o_custkey, o_orderdate] => [[o_orderkey, o_custkey, o_orderdate]]
    - RemoteExchange[GATHER] => [[o_orderkey, o_custkey, o_orderdate]]
        - TableScan[awsdatacatalog:HiveTableHandle{schemaName=tpch100,
 tableName=orders_partitioned,
analyzePartitionValues=Optional.empty}] => [[o_orderkey, o_custkey, o_orderdate]]
                LAYOUT: tpch100.orders_partitioned
                o_orderdate := o_orderdate:string:-1:PARTITION_KEY
                    :: [[1995]]
                o_custkey := o_custkey:int:1:REGULAR
                o_orderkey := o_orderkey:int:0:REGULAR
```

結果にある太字のテキストは、PARTITION_KEY に述語 o_orderdate = '1995' が適用されたことを示しています。

EXPLAIN の例 4: EXPLAIN クエリを使用して結合順序と結合タイプをチェックする

以下の EXPLAIN クエリは、SELECT ステートメントの結合順序と結合タイプをチェックします。EXCEEDED_LOCAL_MEMORY_LIMIT エラーが発生する確率を減らすことができるように、このようなクエリを使用してクエリのメモリ使用量を調べます。

```
EXPLAIN (TYPE DISTRIBUTED)
```

```
   SELECT
      c.c_custkey,
      o.o_orderkey,
      o.o_orderstatus
   FROM tpch100.customer c
   JOIN tpch100.orders o
       ON c.c_custkey = o.o_custkey
   WHERE c.c_custkey = 123
```

結果

```
Query Plan
Fragment 0 [SINGLE]
    Output layout: [c_custkey, o_orderkey, o_orderstatus]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - Output[c_custkey, o_orderkey, o_orderstatus] => [[c_custkey, o_orderkey,
 o_orderstatus]]
        - RemoteSource[1] => [[c_custkey, o_orderstatus, o_orderkey]]

Fragment 1 [SOURCE]
    Output layout: [c_custkey, o_orderstatus, o_orderkey]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - CrossJoin => [[c_custkey, o_orderstatus, o_orderkey]]
            Distribution: REPLICATED
        - ScanFilter[table = awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=customer, analyzePartitionValues=Optional.empty}, grouped = false,
filterPredicate = ("c_custkey" = 123)] => [[c_custkey]]
                LAYOUT: tpch100.customer
                c_custkey := c_custkey:int:0:REGULAR
        - LocalExchange[SINGLE] () => [[o_orderstatus, o_orderkey]]
            - RemoteSource[2] => [[o_orderstatus, o_orderkey]]

Fragment 2 [SOURCE]
    Output layout: [o_orderstatus, o_orderkey]
    Output partitioning: BROADCAST []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - ScanFilterProject[table = awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=orders, analyzePartitionValues=Optional.empty}, grouped = false,
filterPredicate = ("o_custkey" = 123)] => [[o_orderstatus, o_orderkey]]
            LAYOUT: tpch100.orders
            o_orderstatus := o_orderstatus:string:2:REGULAR
```

```
            o_custkey := o_custkey:int:1:REGULAR
            o_orderkey := o_orderkey:int:0:REGULAR
```

このクエリ例は、パフォーマンスを向上させるためにクロス結合に最適化されています。結果は、tpch100.orders が BROADCAST 分散タイプとして分散されることを示しています。これは、結合オペレーションを実行するすべてのノードに tpch100.orders テーブルが分散されることを意味します。BROADCAST 分散タイプでは、tpch100.orders テーブルのフィルタリングされた結果のすべてが、結合操作を実行する各ノードのメモリに収まることが必要とされています。

ただし、tpch100.customer テーブルは tpch100.orders よりも小さくなります。tpch100.customer に必要なメモリは少ないため、クエリを tpch100.orders ではなく BROADCAST tpch100.customer に書き直すことができます。これにより、クエリが EXCEEDED_LOCAL_MEMORY_LIMIT エラーを受け取る確率が低くなります。この戦略では、以下の点を前提としています。

- tpch100.customer.c_custkey が tpch100.customer テーブルで一意である。
- tpch100.customer と tpch100.orders の間に 1 対多のマッピング関係がある。

以下の例は、書き直されたクエリを示しています。

```
SELECT
    c.c_custkey,
    o.o_orderkey,
    o.o_orderstatus
FROM tpch100.orders o
JOIN tpch100.customer c -- the filtered results of tpch100.customer are distributed to
 all nodes.
    ON c.c_custkey = o.o_custkey
WHERE c.c_custkey = 123
```

EXPLAIN の例 5: EXPLAIN クエリを使用して効果がない述語を削除する

EXPLAIN クエリを使用して、フィルタリング述語の有効性をチェックすることができます。以下の例にあるように、結果を使用して効果がない述語を削除できます。

```
EXPLAIN
   SELECT
      c.c_name
   FROM tpch100.customer c
   WHERE c.c_custkey = CAST(RANDOM() * 1000 AS INT)
```

```
   AND c.c_custkey BETWEEN 1000 AND 2000
   AND c.c_custkey = 1500
```

結果

```
Query Plan
- Output[c_name] => [[c_name]]
    - RemoteExchange[GATHER] => [[c_name]]
        - ScanFilterProject[table =
awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=customer, analyzePartitionValues=Optional.empty},
filterPredicate = (("c_custkey" = 1500) AND ("c_custkey" =
CAST(("random"() * 1E3) AS int)))] => [[c_name]]
                LAYOUT: tpch100.customer
                c_custkey := c_custkey:int:0:REGULAR
                c_name := c_name:string:1:REGULAR
```

結果にある filterPredicate は、オプティマイザが元の 3 つの述語を 2 つの述語にマージし、それらの適用順序を変更したことを示しています。

```
filterPredicate = (("c_custkey" = 1500) AND ("c_custkey" = CAST(("random"() * 1E3) AS
 int)))
```

結果が述語 AND c.c_custkey BETWEEN 1000 AND 2000 に効果がないことを示しているので、クエリ結果を変更することなくこの述語を削除できます。

EXPLAIN クエリの結果で使用された用語については、「Athena EXPLAIN ステートメントの結果について」を参照してください。

EXPLAIN ANALYZE での例

次に、EXPLAIN ANALYZE クエリとその出力に関する例を示します。

EXPLAIN ANALYZE の例 1: クエリプランとコンピューティングのコストをテキスト形式で表示するために、EXPLAIN ANALYZE を使用する

次の例の EXPLAIN ANALYZE は、CloudFront ログでの SELECT クエリについて、その実行プランとコンピューティングコストを表示します。使用される形式はデフォルトのテキスト出力です。

```
EXPLAIN ANALYZE SELECT FROM cloudfront_logs LIMIT 10
```

結果

```
 Fragment 1
     CPU: 24.60ms, Input: 10 rows (1.48kB); per task: std.dev.: 0.00, Output: 10 rows
 (1.48kB)
     Output layout: [date, time, location, bytes, requestip, method, host, uri, status,
 referrer,\
       os, browser, browserversion]
Limit[10] => [[date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
  browser, browserversion]]
             CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
             Input avg.: 10.00 rows, Input std.dev.: 0.00%
LocalExchange[SINGLE] () => [[date, time, location, bytes, requestip, method, host,
 uri, status, referrer, os,\
 browser, browserversion]]
                 CPU: 0.00ns (0.00%), Output: 10 rows (1.48kB)
                 Input avg.: 0.63 rows, Input std.dev.: 387.30%
RemoteSource[2] => [[date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
  browser, browserversion]]
                     CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
                     Input avg.: 0.63 rows, Input std.dev.: 387.30%

 Fragment 2
     CPU: 3.83s, Input: 998 rows (147.21kB); per task: std.dev.: 0.00, Output: 20 rows
 (2.95kB)
     Output layout: [date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
       browser, browserversion]
LimitPartial[10] => [[date, time, location, bytes, requestip, method, host, uri,
 status, referrer, os,\
  browser, browserversion]]
             CPU: 5.00ms (0.13%), Output: 20 rows (2.95kB)
             Input avg.: 166.33 rows, Input std.dev.: 141.42%
TableScan[awsdatacatalog:HiveTableHandle{schemaName=default, tableName=cloudfront_logs,
\
  analyzePartitionValues=Optional.empty},
grouped = false] => [[date, time, location, bytes, requestip, method, host, uri, st
                 CPU: 3.82s (99.82%), Output: 998 rows (147.21kB)
                 Input avg.: 166.33 rows, Input std.dev.: 141.42%
                 LAYOUT: default.cloudfront_logs
                 date := date:date:0:REGULAR
                 referrer := referrer:string:9:REGULAR
```

```
                os := os:string:10:REGULAR
                method := method:string:5:REGULAR
                bytes := bytes:int:3:REGULAR
                browser := browser:string:11:REGULAR
                host := host:string:6:REGULAR
                requestip := requestip:string:4:REGULAR
                location := location:string:2:REGULAR
                time := time:string:1:REGULAR
                uri := uri:string:7:REGULAR
                browserversion := browserversion:string:12:REGULAR
                status := status:int:8:REGULAR
```

EXPLAIN ANALYZE の例 2: クエリプランを JSON 形式で表示するために、EXPLAIN ANALYZE を使用する

次の例は、CloudFront ログでの SELECT クエリについて、その実行プランとコンピューティングコストを表示します。この例では、出力形式として JSON を指定しています。

```
EXPLAIN ANALYZE (FORMAT JSON) SELECT * FROM cloudfront_logs LIMIT 10
```

結果

```
{
    "fragments": [{
        "id": "1",

        "stageStats": {
            "totalCpuTime": "3.31ms",
            "inputRows": "10 rows",
            "inputDataSize": "1514B",
            "stdDevInputRows": "0.00",
            "outputRows": "10 rows",
            "outputDataSize": "1514B"
        },
        "outputLayout": "date, time, location, bytes, requestip, method, host,\
           uri, status, referrer, os, browser, browserversion",

        "logicalPlan": {
            "1": [{
                "name": "Limit",
                "identifier": "[10]",
                "outputs": ["date", "time", "location", "bytes", "requestip", "method",
 "host",\
```

```
                  "uri", "status", "referrer", "os", "browser", "browserversion"],
                "details": "",
                "distributedNodeStats": {
                    "nodeCpuTime": "0.00ns",
                    "nodeOutputRows": 10,
                    "nodeOutputDataSize": "1514B",
                    "operatorInputRowsStats": [{
                        "nodeInputRows": 10.0,
                        "nodeInputRowsStdDev": 0.0
                    }]
                },
                "children": [{
                    "name": "LocalExchange",
                    "identifier": "[SINGLE] ()",
                    "outputs": ["date", "time", "location", "bytes", "requestip",
"method", "host",\
                      "uri", "status", "referrer", "os", "browser", "browserversion"],
                    "details": "",
                    "distributedNodeStats": {
                        "nodeCpuTime": "0.00ns",
                        "nodeOutputRows": 10,
                        "nodeOutputDataSize": "1514B",
                        "operatorInputRowsStats": [{
                            "nodeInputRows": 0.625,
                            "nodeInputRowsStdDev": 387.2983346207417
                        }]
                    },
                    "children": [{
                        "name": "RemoteSource",
                        "identifier": "[2]",
                        "outputs": ["date", "time", "location", "bytes", "requestip",
"method", "host",\
                          "uri", "status", "referrer", "os", "browser",
"browserversion"],
                        "details": "",
                        "distributedNodeStats": {
                            "nodeCpuTime": "0.00ns",
                            "nodeOutputRows": 10,
                            "nodeOutputDataSize": "1514B",
                            "operatorInputRowsStats": [{
                                "nodeInputRows": 0.625,
                                "nodeInputRowsStdDev": 387.2983346207417
                            }]
                        },
```

```
                        "children": []
                    }]
                }]
            }]
        }
    }, {
        "id": "2",

        "stageStats": {
            "totalCpuTime": "1.62s",
            "inputRows": "500 rows",
            "inputDataSize": "75564B",
            "stdDevInputRows": "0.00",
            "outputRows": "10 rows",
            "outputDataSize": "1514B"
        },
        "outputLayout": "date, time, location, bytes, requestip, method, host, uri,
status,\
           referrer, os, browser, browserversion",

        "logicalPlan": {
            "1": [{
                "name": "LimitPartial",
                "identifier": "[10]",
                "outputs": ["date", "time", "location", "bytes", "requestip", "method",
"host", "uri",\
                  "status", "referrer", "os", "browser", "browserversion"],
                "details": "",
                "distributedNodeStats": {
                    "nodeCpuTime": "0.00ns",
                    "nodeOutputRows": 10,
                    "nodeOutputDataSize": "1514B",
                    "operatorInputRowsStats": [{
                        "nodeInputRows": 83.33333333333333,
                        "nodeInputRowsStdDev": 223.60679774997897
                    }]
                },
                "children": [{
                    "name": "TableScan",
                    "identifier": "[awsdatacatalog:HiveTableHandle{schemaName=default,\
                       tableName=cloudfront_logs,
analyzePartitionValues=Optional.empty},\
                       grouped = false]",
```

```
                    "outputs": ["date", "time", "location", "bytes", "requestip",
 "method", "host", "uri",\
                       "status", "referrer", "os", "browser", "browserversion"],
                    "details": "LAYOUT: default.cloudfront_logs\ndate :=
 date:date:0:REGULAR\nreferrer :=\
                       referrer: string:9:REGULAR\nos := os:string:10:REGULAR
\nmethod := method:string:5:\
                       REGULAR\nbytes := bytes:int:3:REGULAR\nbrowser :=
 browser:string:11:REGULAR\nhost :=\
                       host:string:6:REGULAR\nrequestip := requestip:string:4:REGULAR
\nlocation :=\
                       location:string:2:REGULAR\ntime := time:string:1: REGULAR
\nuri := uri:string:7:\
                       REGULAR\nbrowserversion := browserversion:string:12:REGULAR
\nstatus :=\
                       status:int:8:REGULAR\n",
                    "distributedNodeStats": {
                        "nodeCpuTime": "1.62s",
                        "nodeOutputRows": 500,
                        "nodeOutputDataSize": "75564B",
                        "operatorInputRowsStats": [{
                            "nodeInputRows": 83.33333333333333,
                            "nodeInputRowsStdDev": 223.60679774997897
                        }]
                    },
                    "children": []
                }]
            }]
        }
    }]
}
```

追加リソース

詳細については、次のリソースを参照してください。

- Athena EXPLAIN ステートメントの結果について
- SQL クエリの実行プランの表示
- 完了したクエリの統計と実行の詳細の表示
- Trino の「EXPLAIN」ドキュメント
- Trino の「EXPLAIN ANALYZE」ドキュメント

- AWS Big Data Blog の「Optimize Federated Query Performance using EXPLAIN and EXPLAIN ANALYZE in Amazon Athena」。

Athena EXPLAIN ステートメントの結果について

このトピックでは、Athena EXPLAIN ステートメントの結果で使用されるオペレーション用語の概要を説明します。

EXPLAIN ステートメントの出力タイプ

EXPLAIN ステートメントの出力は、以下の 2 つのタイプのいずれかになります。

- 論理プラン – SQL エンジンがステートメントの実行に使用する論理プランを表示します。このオプションの構文は EXPLAIN または EXPLAIN (TYPE LOGICAL) です。
- 分散プラン – 分散環境での実行プランを表示します。出力には、処理ステージであるフラグメントが表示されます。各プランフラグメントは、1 つ、または複数のノードによって処理されます。データは、フラグメントを処理するノード間で交換できます。このオプションの構文は EXPLAIN (TYPE DISTRIBUTED) です。

  分散プランの出力では、フラグメント (処理ステージ) は Fragment *number* [*fragment_type*] により表されます。この時、*number* はゼロから始まる整数であり、*fragment_type* はフラグメントがノードによってどのように実行されるかを指定します。以下の表では、データ交換のレイアウトに関する洞察を提供するフラグメントタイプが説明されています。

  分散プランのフラグメントタイプ

フラグメントタイプ	説明
SINGLE	フラグメントは単一のノードで実行されます。
HASH	フラグメントは固定数のノードで実行されます。入力データは、ハッシュ関数を使用して分散されます。
ROUND_ROBIN	フラグメントは固定数のノードで実行されます。入力データは、ラウンドロビン方式で分散されます。
BROADCAST	フラグメントは固定数のノードで実行されます。入力データは、すべてのノードにブロードキャストされます。

フラグメントタイプ	説明
SOURCE	フラグメントは入力分割がアクセスされるノードで実行されます。

Exchange

Exchange 関連の用語は、データがワーカーノード間でどのように交換されるかを表します。転送はローカルまたはリモートのいずれかです。

LocalExchange [***exchange_type***]

クエリのさまざまなステージで、データをワーカーノード内でローカルに転送します。*exchange_type* の値は、このセクションで後ほど説明する論理交換タイプまたは分散交換タイプのいずれかになります。

RemoteExchange [***exchange_type***]

クエリのさまざまなステージで、データをワーカーノード間で転送します。*exchange_type* の値は、このセクションで後ほど説明する論理交換タイプまたは分散交換タイプのいずれかになります。

論理交換タイプ

以下の交換タイプは、論理プランの交換フェーズで実行されるアクションを説明します。

- **GATHER** – 単一のワーカーノードが、他のすべてのワーカーノードからの出力を収集します。例えば、選択クエリの最後のステージでは、すべてのノードから結果が収集され、その結果が Amazon S3 に書き込まれます。
- **REPARTITION** – 次の演算子への適用に必要なパーティショニングスキームに基づいて、行データを特定のワーカーに送信します。
- **REPLICATE** – 行データをすべてのワーカーにコピーします。

分散交換タイプ

以下の交換タイプは、データが分散プラン内のノード間で交換されるときのデータのレイアウトを示します。

- **HASH** – この交換は、ハッシュ関数を使用して複数の宛先にデータを分散します。

- **SINGLE** – この交換は、データを単一の宛先に分散します。

スキャニング

以下の用語は、クエリ中にデータがどのようにスキャンされるかを説明します。

TableScan

Amazon S3 または Apache Hive コネクタからテーブルのソースデータをスキャンし、フィルタ述語から生成されたパーティションプルーニングを適用します。

ScanFilter

Amazon S3 または Hive コネクタからテーブルのソースデータをスキャンし、フィルタ述語から生成されたパーティションプルーニングと、パーティションプルーニング経由で適用されない追加のフィルタ述語から生成されたパーティションプルーニングを適用します。

ScanFilterProject

最初に、Amazon S3 または Hive コネクタからテーブルのソースデータをスキャンし、フィルタ述語から生成されたパーティションプルーニングと、パーティションプルーニング経由で適用されない追加のフィルタ述語から生成されたパーティションプルーニングを適用します。次に、出力データのメモリレイアウトを新しい射影に変更して、後続ステージのパフォーマンスを向上させます。

Join

2 つのテーブル間でデータを結合します。結合は、結合タイプおよび分散タイプ別に分類できます。

結合の種類

結合タイプは、結合オペレーションの実行方法を定義します。

CrossJoin – 結合された 2 つのテーブルのデカルト積を生成します。

InnerJoin – 両方のテーブルに一致する値を持つレコードを選択します。

LeftJoin – 左側のテーブルからのすべてのレコードと、右側のテーブルからの一致するレコードを選択します。一致するレコードがない場合、右側の結果は NULL になります。

RightJoin – 右側のテーブルからのすべてのレコードと、左側のテーブルからの一致するレコードを選択します。一致するレコードがない場合、左側の結果は NULL になります。

FullJoin – 左または右のテーブルのレコードに一致するものがあるすべてのレコードを選択します。結合されたテーブルには両方のテーブルからのすべてのレコードが含まれ、どちらかのテーブルで一致するレコードが欠落していれば NULL が入力されます。

> **Note**
>
> パフォーマンス上の理由から、クエリエンジンは、結合クエリを異なる結合タイプに書き直して、同じ結果を生成することができます。例えば、1 つのテーブルでの述語を使用した内部結合クエリは、CrossJoin に書き直すことができます。これは、スキャンされるデータを少なくするために、述語をテーブルのスキャンフェーズにプッシュダウンします。

結合の分散タイプ

分散タイプは、結合オペレーションの実行時に、データがワーカーノード間でどのように交換されるかを定義します。

パーティション – 左と右の両方のテーブルが、すべてのワーカーノード全体でハッシュパーティション分割されます。パーティション分散では、各ノードで消費されるメモリが少なくなります。パーティション分散は、レプリケート結合よりもはるかに遅くなる可能性があります。パーティション結合は、2 つの大きなテーブルを結合する場合に適しています。

レプリケート – 一方のテーブルがすべてのワーカーノード全体でハッシュパーティション分割され、もう一方のテーブルは、結合操作を実行するために、すべてのワーカーノードにレプリケートされます。レプリケート分散はパーティション結合よりもはるかに高速ですが、各ワーカーノードで消費されるメモリが多くなります。レプリケートされたテーブルが大きすぎる場合は、ワーカーノードでメモリ不足エラーが発生する可能性があります。レプリケート結合は、結合されたテーブルの 1 つが小さい場合に適しています。

PREPARE

後で実行するために、名前 statement_name で SQL ステートメントを作成します。ステートメントには、疑問符で表されるパラメータを含めることができます。パラメータの値を指定してプリペアドステートメントを実行するには、EXECUTE を使用します。

概要

```
PREPARE statement_name FROM statement
```

次の表は、パラメータの説明です。

パラメータ	説明
statement_name	準備されるステートメントの名前です。この名前は、ワークグループ内で一意である必要があります。
statement	SELECT、CTAS、または INSERT INTO クエリです。

> **Note**
>
> ワークグループ内の準備済みステートメントの最大数は 1,000 です。

例

次の例では、パラメータなしで選択クエリを準備します。

```
PREPARE my_select1 FROM
SELECT * FROM nation
```

次の例では、パラメータを含む選択クエリを準備します。productid および quantity の値は、EXECUTE ステートメントの USING 句によって指定されます。

```
PREPARE my_select2 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
```

次の例では、挿入クエリを準備します。

```
PREPARE my_insert FROM
INSERT INTO cities_usa (city, state)
SELECT city, state
FROM cities_world
WHERE country = ?
```

追加リソース

準備済みステートメントを使用したクエリ

EXECUTE

DEALLOCATE PREPARE

INSERT INTO

EXECUTE

statement_name という名前のプリペアドステートメントを実行します。プリペアドステートメントの疑問符のパラメータ値は、カンマ区切りのリストの USING 句で定義されています。プリペアドステートメントを作成するには、PREPARE を使用します。

概要

```
EXECUTE statement_name [ USING parameter1[, parameter2, ... ] ]
```

例

次の例は、パラメータがないクエリを準備し、実行します。

```
PREPARE my_select1 FROM
SELECT name FROM nation
EXECUTE my_select1
```

次の例は、単一のパラメータがあるクエリを準備し、実行します。

```
PREPARE my_select2 FROM
SELECT * FROM "my_database"."my_table" WHERE year = ?
EXECUTE my_select2 USING 2012
```

これは次に相当します:

```
SELECT * FROM "my_database"."my_table" WHERE year = 2012
```

次の例は、2 つのパラメータを持つクエリを準備し、実行します。

```
PREPARE my_select3 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
EXECUTE my_select3 USING 346078, 12
```

追加リソース

準備済みステートメントを使用したクエリ

PREPARE

INSERT INTO

DEALLOCATE PREPARE

現在のワークグループの準備済みステートメントから、指定された名前を持つ準備済みステートメントを削除します。

概要

```
DEALLOCATE PREPARE statement_name
```

例

以下の例では、現在のワークグループから my_select1 準備済みステートメントを削除しています。

```
DEALLOCATE PREPARE my_select1
```

追加リソース

準備済みステートメントを使用したクエリ

PREPARE

UNLOAD

SELECT ステートメントからのクエリ結果を指定したデータ形式に書き込みます。UNLOAD でサポートされる形式には Apache Parquet、ORC、Apache Avro、および JSON が含まれます。CSV は Athena SELECT コマンドがサポートする唯一の出力形式ですが、さまざまな出力形式をサポートする UNLOAD コマンドを使用して SELECT クエリを囲み、クエリの出力を UNLOAD がサポートする形式のいずれかに書き換えることができます。

CTAS ステートメントを使用して CSV 以外の形式でデータを出力できますが、これらのステートメントでも、Athena でテーブルを作成する必要があります。UNLOAD ステートメントは、SELECT クエリの結果を非 CSV 形式で出力したいが、関連付けられたテーブルを必要としない場合に便利で

す。たとえば、ダウンストリームアプリケーションでは、SELECT クエリの結果を JSON 形式にする必要があります。また、SELECT クエリの結果を追加の分析に使用する場合は、Parquet や ORC の方が CSV よりもパフォーマンス面での利点を得られる可能性があります。

考慮事項と制約事項

Athena で UNLOAD ステートメント使用するときは、次の点に注意してください。

- ファイルのグローバルな順序付けがありません - UNLOAD 結果が複数のファイルに並行して書き込まれます。UNLOAD ステートメントの SELECT クエリでソート順序が指定されている場合、各ファイルの内容はソート順序になりますが、ファイル同士は互いにソートされません。
- 孤立したデータが削除されません - 障害が発生した場合、Athena は孤立したデータの削除を試みません。この動作は、CTAS および INSERT INTO ステートメントの動作と同じです。
- 最大パーティション数 - UNLOAD で使用できるパーティションの最大数は 100 です。
- メタデータおよびマニフェストファイル - Athena が、UNLOAD クエリごとにメタデータファイルとデータマニフェストファイルを生成します。マニフェストは、クエリが書き込んだファイルを追跡します。両方のファイルが、Amazon S3 の Athena クエリ結果の場所に保存されます。詳細については、「クエリ出力ファイルの識別」を参照してください。
- 暗号化 - UNLOAD 出力ファイルが、Amazon S3 で使用される暗号化設定に従って暗号化されます。暗号化設定を設定して UNLOAD 結果を暗号化するには、EncryptionConfiguration API を使用します。
- 準備済みステートメント - UNLOAD は、準備済みステートメントで使用できます。Athena での準備済みステートメントの詳細については、「パラメータ化されたクエリの使用」を参照してください。
- サービスクォータ - UNLOAD は、DML クエリクォータを使用します。クオータの詳細については、「Service Quotas」を参照してください。
- 予想されるバケット所有者 — 予想されるバケット所有者の設定は、UNLOAD クエリで指定された送信先の Amazon S3 の場所には適用されません。予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。詳細については、「Athena コンソールを使用したクエリ結果の場所の指定」を参照してください。

構文

UNLOAD ステートメントでは、次の構文を使用します。

```
UNLOAD (SELECT col_name[, ...] FROM old_table)
```

```
TO 's3://DOC-EXAMPLE-BUCKET/my_folder/'
WITH ( property_name = 'expression' [, ...] )
```

パーティションに書き込む場合を除き、TO 書き込み先にはデータのない Amazon S3 ロケーションを指定する必要があります。UNLOAD クエリが、指定された場所に書き込む前に、バケットの場所が空であることを確認します。UNLOAD は、指定された場所にデータがすでに存在する場合、その場所にデータを書き込まないので、UNLOADが既存のデータを上書きすることはありません。バケットの場所を UNLOAD の送信先として再利用するには、バケットの場所にあるデータを削除して、クエリを再度実行します。

UNLOAD がパーティションにき込む場合、これとは異なる動作になることに注意してください。同じ SELECT ステートメント、同じ TO ロケーション、および同じパーティションを持つ同一の UNLOAD クエリを複数回実行する場合、各 UNLOAD クエリはその口ケーションにある Amazon S3 と指定されたパーティションにデータをアンロードします。

パラメータ

property_name に使用できる値は次のとおりです。

形式 =「***file_format***」

必須。出力のファイル形式を指定します。*file_format* に使用できる値は、ORC、PARQUET、AVRO、JSON、 またはTEXTFILE です。

圧縮 =「***compression_format***」

オプション。このオプションは、ORC および Parquet 形式に固有のものです。ORC の場合、デフォルトは zlib で、Parquet の場合、デフォルトは gzip です。サポートされている圧縮形式については、「Athena 圧縮サポート」を参照してください。

> **Note**
>
> このオプションは、AVRO 形式には適用されません。Athena が、JSON および TEXTFILE 形式に gzip を使用します。

compression_level = ***compression_level***

オプション。ZSTD 圧縮に使用する圧縮レベル。このプロパティは、ZSTD 圧縮にのみ適用されます。詳細については、「Athena での ZSTD 圧縮レベルの使用」を参照してください。

field_delimiter =「*delimiter*」

オプション。CSV、TSV、およびその他のテキスト形式のファイルのための単一文字のフィールド区切り文字を指定します。次に示す例では、カンマ区切り文字を指定しています。

```
WITH (field_delimiter = ',')
```

現在、複数文字のフィールド区切り文字はサポートされていません。フィールド区切り文字を指定しない場合、8 進文字 \001 (^A) が使用されます。

partitioned_by = ARRAY[*col_name*[,...]]

オプション。出力をパーティション化する列の配列リスト。

> **Note**
>
> SELECT ステートメントでは、パーティション化された列の名前は、列のリストの最後に記述されていることを確認してください。

例

次の例では、JSON 形式を使用して SELECT クエリの出力を Amazon S3 の場所 s3://DOC-EXAMPLE-BUCKET/unload_test_1/ に書き込みます。

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/unload_test_1/'
WITH (format = 'JSON')
```

次の例では、Snappy 圧縮を使用して SELECT クエリの出力を Parquet 形式で書き込みます。

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format = 'PARQUET',compression = 'SNAPPY')
```

次の例では、出力が最後の列でパーティション化された状態で、4 つの列をテキスト形式で書き込みます。

```
UNLOAD (SELECT name1, address1, comment1, key1 FROM table1)
TO 's3://DOC-EXAMPLE-BUCKET/ partitioned/'
```

```
WITH (format = 'TEXTFILE', partitioned_by = ARRAY['key1'])
```

次の例では、Parquet ファイル形式、ZSTD 圧縮、および ZSTD 圧縮レベル 4 を使用して、クエリ結果を指定された場所にアンロードします。

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format = 'PARQUET', compression = 'ZSTD', compression_level = 4)
```

追加リソース

- AWS Big Data Blog の「Simplify your ETL and ML pipelines using the Amazon Athena UNLOAD feature」。

Amazon Athena の関数

Athena エンジンのバージョンに関する詳細については、「Athena エンジンバージョンリファレンス」を参照してください。AT TIME ZONE 演算子で使用できるタイムゾーンのリストについては、「サポートされているタイムゾーン」を参照してください。

Athena エンジンバージョン 3

Athena エンジンバージョン 3 の関数は Trino に基づいています。Trino の関数、演算子、および式については、Trino ドキュメントの「Functions and operators」(関数と演算子) とそのサブセクションを参照してください。

- Aggregate
- 配列
- バイナリ
- ビット操作
- Color
- 比較
- 条件付き
- 変換
- 日付および時間
- 10 進数

- 地理空間
- HyperLogLog
- IP アドレス
- JSON
- Lambda
- 論理
- 機械学習
- マッピング
- 数学
- 分位点ダイジェスト
- 正規表現
- セッション
- セットダイジェスト
- 文字列
- テーブル
- Teradata
- T ダイジェスト
- URL
- UUID
- Window

Athena エンジンバージョン 2

Athena エンジンバージョン 2 の関数は Presto 0.217 に基づいています。Athena エンジンバージョン 2 の地理空間関数については、「Athena エンジンバージョン 2 の地理空間関数」を参照してください。

> **Note**
>
> Presto 0.217 の関数のバージョン固有のドキュメントは利用できなくなりました。現在の Presto 関数、演算子、および式については、「Presto functions and operators」(Presto の関数と演算子)、またはこのセクションにあるサブカテゴリのリンクを参照してください。

- 論理演算子
- 比較関数および演算子
- 条件式
- 変換関数
- 数学関数と演算子
- ビット単位関数
- 進数関数および演算子
- 文字列関数および演算子
- バイナリ関数
- 日付/時刻関数と演算子
- 正規表現関数
- JSON 関数および演算子
- URL 関数
- 集計関数
- Window 関数
- カラー関数
- 配列関数と演算子
- マッピング関数と演算子
- Lambda 式および関数
- Teradata 関数

サポートされているタイムゾーン

次の例にあるように、AT TIME ZONE 演算子を SELECT timestamp ステートメントで使用して、返されるタイムスタンプのタイムゾーンを指定します。

```
SELECT timestamp '2012-10-31 01:00 UTC' AT TIME ZONE 'America/Los_Angeles' AS la_time;
```

結果

```
la_time
```

```
2012-10-30 18:00:00.000 America/Los_Angeles
```

以下のリストには、Athena で AT TIME ZONE 演算子に使用できるタイムゾーンが記載されています。その他のタイムゾーン関連の関数および例については、「タイムゾーン関数と例」を参照してください。

```
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Ceuta
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/El_Aaiun
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Juba
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
```

```
Africa/Malabo
Africa/Maputo
Africa/Maseru
Africa/Mbabane
Africa/Mogadishu
Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli
Africa/Tunis
Africa/Windhoek
America/Adak
America/Anchorage
America/Anguilla
America/Antigua
America/Araguaina
America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/ComodRivadavia
America/Argentina/Cordoba
America/Argentina/Jujuy
America/Argentina/La_Rioja
America/Argentina/Mendoza
America/Argentina/Rio_Gallegos
America/Argentina/Salta
America/Argentina/San_Juan
America/Argentina/San_Luis
America/Argentina/Tucuman
America/Argentina/Ushuaia
America/Aruba
America/Asuncion
America/Atikokan
America/Atka
America/Bahia
America/Bahia_Banderas
America/Barbados
America/Belem
America/Belize
```

```
America/Blanc-Sablon
America/Boa_Vista
America/Bogota
America/Boise
America/Buenos_Aires
America/Cambridge_Bay
America/Campo_Grande
America/Cancun
America/Caracas
America/Catamarca
America/Cayenne
America/Cayman
America/Chicago
America/Chihuahua
America/Coral_Harbour
America/Cordoba
America/Costa_Rica
America/Creston
America/Cuiaba
America/Curacao
America/Danmarkshavn
America/Dawson
America/Dawson_Creek
America/Denver
America/Detroit
America/Dominica
America/Edmonton
America/Eirunepe
America/El_Salvador
America/Ensenada
America/Fort_Nelson
America/Fort_Wayne
America/Fortaleza
America/Glace_Bay
America/Godthab
America/Goose_Bay
America/Grand_Turk
America/Grenada
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax
America/Havana
```

```
America/Hermosillo
America/Indiana/Indianapolis
America/Indiana/Knox
America/Indiana/Marengo
America/Indiana/Petersburg
America/Indiana/Tell_City
America/Indiana/Vevay
America/Indiana/Vincennes
America/Indiana/Winamac
America/Indianapolis
America/Inuvik
America/Iqaluit
America/Jamaica
America/Jujuy
America/Juneau
America/Kentucky/Louisville
America/Kentucky/Monticello
America/Knox_IN
America/Kralendijk
America/La_Paz
America/Lima
America/Los_Angeles
America/Louisville
America/Lower_Princes
America/Maceio
America/Managua
America/Manaus
America/Marigot
America/Martinique
America/Matamoros
America/Mazatlan
America/Mendoza
America/Menominee
America/Merida
America/Metlakatla
America/Mexico_City
America/Miquelon
America/Moncton
America/Monterrey
America/Montevideo
America/Montreal
America/Montserrat
America/Nassau
America/New_York
```

```
America/Nipigon
America/Nome
America/Noronha
America/North_Dakota/Beulah
America/North_Dakota/Center
America/North_Dakota/New_Salem
America/Ojinaga
America/Panama
America/Pangnirtung
America/Paramaribo
America/Phoenix
America/Port-au-Prince
America/Port_of_Spain
America/Porto_Acre
America/Porto_Velho
America/Puerto_Rico
America/Punta_Arenas
America/Rainy_River
America/Rankin_Inlet
America/Recife
America/Regina
America/Resolute
America/Rio_Branco
America/Rosario
America/Santa_Isabel
America/Santarem
America/Santiago
America/Santo_Domingo
America/Sao_Paulo
America/Scoresbysund
America/Shiprock
America/Sitka
America/St_Barthelemy
America/St_Johns
America/St_Kitts
America/St_Lucia
America/St_Thomas
America/St_Vincent
America/Swift_Current
America/Tegucigalpa
America/Thule
America/Thunder_Bay
America/Tijuana
America/Toronto
```

```
America/Tortola
America/Vancouver
America/Virgin
America/Whitehorse
America/Winnipeg
America/Yakutat
America/Yellowknife
Antarctica/Casey
Antarctica/Davis
Antarctica/DumontDUrville
Antarctica/Macquarie
Antarctica/Mawson
Antarctica/McMurdo
Antarctica/Palmer
Antarctica/Rothera
Antarctica/South_Pole
Antarctica/Syowa
Antarctica/Troll
Antarctica/Vostok
Arctic/Longyearbyen
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat
Asia/Ashkhabad
Asia/Atyrau
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Barnaul
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
Asia/Chita
Asia/Choibalsan
Asia/Chongqing
Asia/Chungking
Asia/Colombo
Asia/Dacca
```

```
Asia/Damascus
Asia/Dhaka
Asia/Dili
Asia/Dubai
Asia/Dushanbe
Asia/Gaza
Asia/Harbin
Asia/Hebron
Asia/Ho_Chi_Minh
Asia/Hong_Kong
Asia/Hovd
Asia/Irkutsk
Asia/Istanbul
Asia/Jakarta
Asia/Jayapura
Asia/Jerusalem
Asia/Kabul
Asia/Kamchatka
Asia/Karachi
Asia/Kashgar
Asia/Kathmandu
Asia/Katmandu
Asia/Khandyga
Asia/Kolkata
Asia/Krasnoyarsk
Asia/Kuala_Lumpur
Asia/Kuching
Asia/Kuwait
Asia/Macao
Asia/Macau
Asia/Magadan
Asia/Makassar
Asia/Manila
Asia/Muscat
Asia/Nicosia
Asia/Novokuznetsk
Asia/Novosibirsk
Asia/Omsk
Asia/Oral
Asia/Phnom_Penh
Asia/Pontianak
Asia/Pyongyang
Asia/Qatar
Asia/Qyzylorda
```

```
Asia/Rangoon
Asia/Riyadh
Asia/Saigon
Asia/Sakhalin
Asia/Samarkand
Asia/Seoul
Asia/Shanghai
Asia/Singapore
Asia/Srednekolymsk
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Tel_Aviv
Asia/Thimbu
Asia/Thimphu
Asia/Tokyo
Asia/Tomsk
Asia/Ujung_Pandang
Asia/Ulaanbaatar
Asia/Ulan_Bator
Asia/Urumqi
Asia/Ust-Nera
Asia/Vientiane
Asia/Vladivostok
Asia/Yakutsk
Asia/Yangon
Asia/Yekaterinburg
Asia/Yerevan
Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faeroe
Atlantic/Faroe
Atlantic/Jan_Mayen
Atlantic/Madeira
Atlantic/Reykjavik
Atlantic/South_Georgia
Atlantic/St_Helena
Atlantic/Stanley
Australia/ACT
Australia/Adelaide
Australia/Brisbane
```

```
Australia/Broken_Hill
Australia/Canberra
Australia/Currie
Australia/Darwin
Australia/Eucla
Australia/Hobart
Australia/LHI
Australia/Lindeman
Australia/Lord_Howe
Australia/Melbourne
Australia/NSW
Australia/North
Australia/Perth
Australia/Queensland
Australia/South
Australia/Sydney
Australia/Tasmania
Australia/Victoria
Australia/West
Australia/Yancowinna
Brazil/Acre
Brazil/DeNoronha
Brazil/East
Brazil/West
CET
CST6CDT
Canada/Atlantic
Canada/Central
Canada/Eastern
Canada/Mountain
Canada/Newfoundland
Canada/Pacific
Canada/Saskatchewan
Canada/Yukon
Chile/Continental
Chile/EasterIsland
Cuba
EET
EST5EDT
Egypt
Eire
Europe/Amsterdam
Europe/Andorra
Europe/Astrakhan
```

```
Europe/Athens
Europe/Belfast
Europe/Belgrade
Europe/Berlin
Europe/Bratislava
Europe/Brussels
Europe/Bucharest
Europe/Budapest
Europe/Busingen
Europe/Chisinau
Europe/Copenhagen
Europe/Dublin
Europe/Gibraltar
Europe/Guernsey
Europe/Helsinki
Europe/Isle_of_Man
Europe/Istanbul
Europe/Jersey
Europe/Kaliningrad
Europe/Kiev
Europe/Kirov
Europe/Lisbon
Europe/Ljubljana
Europe/London
Europe/Luxembourg
Europe/Madrid
Europe/Malta
Europe/Mariehamn
Europe/Minsk
Europe/Monaco
Europe/Moscow
Europe/Nicosia
Europe/Oslo
Europe/Paris
Europe/Podgorica
Europe/Prague
Europe/Riga
Europe/Rome
Europe/Samara
Europe/San_Marino
Europe/Sarajevo
Europe/Simferopol
Europe/Skopje
Europe/Sofia
```

```
Europe/Stockholm
Europe/Tallinn
Europe/Tirane
Europe/Tiraspol
Europe/Ulyanovsk
Europe/Uzhgorod
Europe/Vaduz
Europe/Vatican
Europe/Vienna
Europe/Vilnius
Europe/Volgograd
Europe/Warsaw
Europe/Zagreb
Europe/Zaporozhye
Europe/Zurich
GB
GB-Eire
Hongkong
Iceland
Indian/Antananarivo
Indian/Chagos
Indian/Christmas
Indian/Cocos
Indian/Comoro
Indian/Kerguelen
Indian/Mahe
Indian/Maldives
Indian/Mauritius
Indian/Mayotte
Indian/Reunion
Iran
Israel
Jamaica
Japan
Kwajalein
Libya
MET
MST7MDT
Mexico/BajaNorte
Mexico/BajaSur
Mexico/General
NZ
NZ-CHAT
Navajo
```

```
PRC
PST8PDT
Pacific/Apia
Pacific/Auckland
Pacific/Bougainville
Pacific/Chatham
Pacific/Chuuk
Pacific/Easter
Pacific/Efate
Pacific/Enderbury
Pacific/Fakaofo
Pacific/Fiji
Pacific/Funafuti
Pacific/Galapagos
Pacific/Gambier
Pacific/Guadalcanal
Pacific/Guam
Pacific/Honolulu
Pacific/Johnston
Pacific/Kiritimati
Pacific/Kosrae
Pacific/Kwajalein
Pacific/Majuro
Pacific/Marquesas
Pacific/Midway
Pacific/Nauru
Pacific/Niue
Pacific/Norfolk
Pacific/Noumea
Pacific/Pago_Pago
Pacific/Palau
Pacific/Pitcairn
Pacific/Pohnpei
Pacific/Ponape
Pacific/Port_Moresby
Pacific/Rarotonga
Pacific/Saipan
Pacific/Samoa
Pacific/Tahiti
Pacific/Tarawa
Pacific/Tongatapu
Pacific/Truk
Pacific/Wake
Pacific/Wallis
```

```
Pacific/Yap
Poland
Portugal
ROK
Singapore
Turkey
US/Alaska
US/Aleutian
US/Arizona
US/Central
US/East-Indiana
US/Eastern
US/Hawaii
US/Indiana-Starke
US/Michigan
US/Mountain
US/Pacific
US/Pacific-New
US/Samoa
W-SU
WET
```

タイムゾーン関数と例

追加のタイムゾーン関連の関数と例を次に示します。

- at_timezone(***timestamp***、***zone***) – *zone* について、対応する現地時間の *timestamp* の値を返します。

  例

  ```
  SELECT at_timezone(timestamp '2021-08-22 00:00 UTC', 'Canada/Newfoundland')
  ```

  結果

  ```
  2021-08-21 21:30:00.000 Canada/Newfoundland
  ```

- timezone_hour(***timestamp***) – timestamp からオフセットされたタイムゾーンの時間を bigint として返します。

  例

```
SELECT timezone_hour(timestamp '2021-08-22 04:00 UTC' AT TIME ZONE 'Canada/
Newfoundland')
```

結果

```
-2
```

- timezone_minute(***timestamp***) – *timestamp* からオフセットされたタイムゾーンの分を bigint として返します。

  例

```
SELECT timezone_minute(timestamp '2021-08-22 04:00 UTC' AT TIME ZONE 'Canada/
Newfoundland')
```

  結果

```
-30
```

- with_timezone(***timestamp***, ***zone***) – 指定された *timestamp* と *zone* の値からタイムゾーン付きのタイムスタンプを返します。

  例

```
SELECT with_timezone(timestamp '2021-08-22 04:00', 'Canada/Newfoundland')
```

  結果

```
2021-08-22 04:00:00.000 Canada/Newfoundland
```

DDL ステートメント

以下の DDL ステートメントを Athena で直接使用します。

Athena クエリエンジンの一部は、HiveQL DDL に基づいています。

Athena はすべての DDL ステートメントをサポートしておらず、HiveQL DDL と Athena DDL にはいくつかの違いがあります。詳細については、このセクションのリファレンストピック、および「サポートされない DDL」を参照してください。

トピック

- サポートされない DDL
- ALTER DATABASE SET DBPROPERTIES
- ALTER TABLE ADD COLUMNS
- ALTER TABLE ADD PARTITION
- ALTER TABLE DROP PARTITION
- ALTER TABLE RENAME PARTITION
- ALTER TABLE REPLACE COLUMNS
- ALTER TABLE SET LOCATION
- ALTER TABLE SET TBLPROPERTIES
- CREATE DATABASE
- CREATE TABLE
- CREATE TABLE AS
- CREATE VIEW
- DESCRIBE
- DESCRIBE VIEW
- DROP DATABASE
- DROP TABLE
- DROP VIEW
- MSCK REPAIR TABLE
- SHOW COLUMNS
- SHOW CREATE TABLE
- SHOW CREATE VIEW
- SHOW DATABASES
- SHOW PARTITIONS
- SHOW TABLES

- SHOW TBLPROPERTIES
- SHOW VIEWS

サポートされない DDL

以下の DDL ステートメントは、Athena でサポートされません。

- ALTER INDEX
- ALTER TABLE *table_name* ARCHIVE PARTITION
- ALTER TABLE *table_name* CLUSTERED BY
- ALTER TABLE *table_name* EXCHANGE PARTITION
- ALTER TABLE *table_name* NOT CLUSTERED
- ALTER TABLE *table_name* NOT SKEWED
- ALTER TABLE *table_name* NOT SORTED
- ALTER TABLE *table_name* NOT STORED AS DIRECTORIES
- ALTER TABLE *table_name* partitionSpec CHANGE COLUMNS
- ALTER TABLE *table_name* partitionSpec COMPACT
- ALTER TABLE *table_name* partitionSpec CONCATENATE
- ALTER TABLE *table_name* partitionSpec SET FILEFORMAT
- ALTER TABLE *table_name* SET SERDEPROPERTIES
- ALTER TABLE *table_name* SET SKEWED LOCATION
- ALTER TABLE *table_name* SKEWED BY
- ALTER TABLE *table_name* TOUCH
- ALTER TABLE *table_name* UNARCHIVE PARTITION
- COMMIT
- CREATE INDEX
- CREATE ROLE
- CREATE TABLE *table_name* LIKE *existing_table_name*
- CREATE TEMPORARY MACRO
- DELETE FROM

- DESCRIBE DATABASE
- DFS
- DROP INDEX
- DROP ROLE
- DROP TEMPORARY MACRO
- EXPORT TABLE
- GRANT ROLE
- IMPORT TABLE
- LOCK DATABASE
- LOCK TABLE
- REVOKE ROLE
- ROLLBACK
- SHOW COMPACTIONS
- SHOW CURRENT ROLES
- SHOW GRANT
- SHOW INDEXES
- SHOW LOCKS
- SHOW PRINCIPALS
- SHOW ROLE GRANT
- SHOW ROLES
- SHOW STATS
- SHOW TRANSACTIONS
- START TRANSACTION
- UNLOCK DATABASE
- UNLOCK TABLE

ALTER DATABASE SET DBPROPERTIES

データベースのプロパティを 1 つ以上作成します。DATABASE と SCHEMA は、どちらを使用しても同じように機能します。

概要

```
ALTER {DATABASE|SCHEMA} database_name
  SET DBPROPERTIES ('property_name'='property_value' [, ...] )
```

パラメータ

SET DBPROPERTIES ('property_name'='property_value' [, ...]

データベースのプロパティを property_name として指定し、各プロパティの値を property_value として設定します。property_name が既に存在する場合、古い値は property_value で上書きされます。

例

```
ALTER DATABASE jd_datasets
  SET DBPROPERTIES ('creator'='John Doe', 'department'='applied mathematics');
```

```
ALTER SCHEMA jd_datasets
  SET DBPROPERTIES ('creator'='Jane Doe');
```

ALTER TABLE ADD COLUMNS

既存のテーブルに 1 つ以上の列を追加します。オプションの PARTITION 構文を使用すると、パーティションメタデータが更新されます。

概要

```
ALTER TABLE table_name
  [PARTITION
   (partition_col1_name = partition_col1_value
   [,partition_col2_name = partition_col2_value][,...])]
  ADD COLUMNS (col_name data_type)
```

パラメータ

PARTITION (partition_col_name = partition_col_value [,...])

ユーザー指定の列名/値の組み合わせでパーティションを作成します。列のデータ型が文字列である場合にのみ、partition_col_value を引用符で囲みます。

ADD COLUMNS (col_name data_type [,col_name data_type,...])

既存の列の後、パーティション列の前に列を追加します。

例

```
ALTER TABLE events ADD COLUMNS (eventowner string)
```

```
ALTER TABLE events PARTITION (awsregion='us-west-2') ADD COLUMNS (event string)
```

```
ALTER TABLE events PARTITION (awsregion='us-west-2') ADD COLUMNS (eventdescription
 string)
```

メモ

- ALTER TABLE ADD COLUMNS の実行後に Athena クエリエディタのナビゲーションペインで新しいテーブル列を表示するには、エディタのテーブルリストを手動で更新してから、テーブルをもう一度展開します。
- ALTER TABLE ADD COLUMNS は、date データ型の列では機能しません。この問題の回避策は、代わりに timestamp データ型を使用することです。

ALTER TABLE ADD PARTITION

テーブルのパーティション列を 1 つ以上作成します。各パーティションは、1 つ以上の異なる列名/値の組み合わせで構成されます。指定した組み合わせごとに別個のデータディレクトリが作成されます。これにより、クエリパフォーマンスが向上する場合があります。パーティション分割された列は、テーブルのデータ自体内には存在しないため、テーブル内の列自体と同じ名前を使用すると、エラーになります。詳細については、「Athena でのデータのパーティション化」を参照してください。

Athena では、テーブルとそのパーティションが同じデータ形式を使用する必要がありますが、スキーマは異なる場合があります。詳細については、「」を参照してくださいパーティションがあるテーブルを更新する

IAM ポリシーで必要とされるリソースレベルのアクセス許可 (glue:CreatePartition を含む) については、「AWS Glue API アクセス許可: アクションとリソースのリファレンス」および「AWS Glue Data Catalog のデータベースとテーブルへのきめ細かなアクセス」を参照してくださ

い。Athena を使用する際のアクセス許可に関するトラブルシューティングについては、「アクセス許可」トピックの「Athena のトラブルシューティング」セクションを参照してください。

概要

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION
  (partition_col1_name = partition_col1_value
  [,partition_col2_name = partition_col2_value]
  [,...])
  [LOCATION 'location1']
  [PARTITION
  (partition_colA_name = partition_colA_value
  [,partition_colB_name = partition_colB_value
  [,...])]
  [LOCATION 'location2']
  [,...]
```

パラメータ

パーティションを追加するときは、パーティションの 1 つ、または複数の列名/値ペアと、そのパーティションのデータファイルが格納されている Amazon S3 パスを指定します。

[IF NOT EXISTS]

同じ定義のパーティションが既に存在する場合、エラーを抑制します。

PARTITION (partition_col_name = partition_col_value [,...])

ユーザー指定の列名/値の組み合わせでパーティションを作成します。列のデータ型が文字列である場合に限り、`partition_col_value` を文字列の文字で囲みます。

[LOCATION 'location']

前のステートメントで定義したパーティションの保存先のディレクトリを指定します。データが Hive スタイルのパーティショニング (pk1=v1/pk2=v2/pk3=v3) を使用する場合、LOCATION 句はオプションです。Hive スタイルのパーティション化では、テーブルの場所、パーティションキー名、およびパーティションキー値からフル Amazon S3 URI が自動的に作成されます。詳細については、「Athena でのデータのパーティション化」を参照してください。

考慮事項

Amazon Athena は、単一の ALTER TABLE ADD PARTITION DDL ステートメントに追加できるパーティションの数に特定の制限を設定していません。ただし、大量のパーティションを追加する必要がある場合は、パフォーマンス問題が発生する可能性を避けるため、操作を小規模なバッチに分割することを検討してください。以下の例は、連続するコマンドを使用してパーティションを個別に追加し、IF NOT EXISTS を使用して重複するパーティションが追加されないようにしています。

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-01')
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-02')
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-03')
```

Athena でパーティションを使用するときは、以下の点に留意してください。

- Athena では、1,000 万のパーティションを持つ AWS Glue テーブルへのクエリがサポートされていますが、1 回のスキャンで読み取れるのは、100 万のパーティションまでです。
- クエリを最適化し、スキャンされるパーティションの数を減らすため、パーティションプルーニングやパーティションインデックスの使用といった戦略を検討してください。
- AWS Glue Data Catalog を使用していない場合、1 テーブルあたりのパーティションの数は最大 20,000 個になります。クォータは、引き上げをリクエストすることができます。

Athena でのパーティションの使用に関するその他の考慮事項については、「Athena でのデータのパーティション化」を参照してください。

例

次の例では、Hive スタイルのパーティション化されたデータ用のテーブルに、1 つのパーティションを追加します。

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-14', country = 'IN');
```

次の例では、Hive スタイルのパーティション化されたデータ用のテーブルに、複数のパーティションを追加します。

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-31', country = 'IN')
  PARTITION (dt = '2016-06-01', country = 'IN');
```

テーブルが Hive スタイルのパーティション化されたデータ向けのものではない場合は、LOCATION 句が必要です。この句は、パーティションのデータが含まれるプレフィックスのフル Amazon S3 URI でなければなりません。

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-31', country = 'IN') LOCATION 's3://DOC-EXAMPLE-BUCKET/path/
to/INDIA_31_May_2016/'
  PARTITION (dt = '2016-06-01', country = 'IN') LOCATION 's3://DOC-EXAMPLE-BUCKET/path/
to/INDIA_01_June_2016/';
```

パーティションがすでに存在する場合に工ラーを無視するには、次の例のように IF NOT EXISTS 句を使用します。

```
ALTER TABLE orders ADD IF NOT EXISTS
  PARTITION (dt = '2016-05-14', country = 'IN');
```

ゼロバイト _$folder$ ファイル

ALTER TABLE ADD PARTITION ステートメントを実行し、すでに存在するパーティションと正しくない Amazon S3 の場所を誤って指定すると、形式 *partition_value*_$folder$ のゼロバイトプレースホルダーが Amazon S3 に作成されます。これらのファイルは手動で削除する必要があります。

これを防ぐには、次の例のように、ALTER TABLE ADD PARTITION ステートメントで ADD IF NOT EXISTS 構文を使用します。

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION […]
```

ALTER TABLE DROP PARTITION

名前付きテーブルの指定したパーティションを削除します。

概要

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION (partition_spec) [, PARTITION
 (partition_spec)]
```

パラメータ

[IF EXISTS]

指定したパーティションが存在しない場合、エラーメッセージを抑制します。

PARTITION (partition_spec)

各 `partition_spec` は、列名/値の組み合わせを `partition_col_name = partition_col_value [,...]` という形式で指定します。

例

```
ALTER TABLE orders
DROP PARTITION (dt = '2014-05-14', country = 'IN');
```

```
ALTER TABLE orders
DROP PARTITION (dt = '2014-05-14', country = 'IN'), PARTITION (dt = '2014-05-15',
 country = 'IN');
```

メモ

`ALTER TABLE DROP PARTITION` ステートメントには、すべてのパーティションを一括に削除するための単一的な構文は存在しません。また、削除するパーティションの範囲を指定するための、フィルタリング条件もサポートされていません。

この点の回避策として、スクリプト内で、AWS Glue API の GetPartitions および BatchDeletePartition アクションを使用できます。`GetPartitions` アクションでは、SQL の `WHERE` 式のような、複雑なフィルター表現がサポートされます。`GetPartitions` を使用して、削除対象のパーティションについてフィルタリングされたリストを作成した後は、`BatchDeletePartition` アクションにより、25 個までのパーティションをまとめて削除できます。

> **Important**
>
> 既知の問題により、`ALTER TABLE DROP PARTITION` ステートメントに無効なパーティションを指定すると、テーブルのすべてのパーティションが AWS Glue で削除されます。例えば、次のステートメントは、指定されたパーティションが存在しない場合でも、テーブル *`my_table`* のすべてのパーティションを削除します。回避策として、`ALTER TABLE`

DROP PARTITION ステートメントを実行する前にパーティション情報を正しく入力してください。

```
ALTER TABLE my_table DROP IF EXISTS PARTITION(zzz='');
```

ALTER TABLE RENAME PARTITION

パーティション値の名前を変更します。

> **Note**
>
> ALTER TABLE RENAME PARTITION はパーティション列の名前を変更しません。パーティション列の名前を変更するには、AWS Glue コンソールを使用できます。詳細については、このドキュメントで後述する「AWS Glue でのパーティション列の名前の変更」を参照してください。

概要

table_name という名前のテーブルの場合、partition_spec が指定したパーティション値の名前を、new_partition_spec が指定した値に変更します。

```
ALTER TABLE table_name PARTITION (partition_spec) RENAME TO PARTITION
 (new_partition_spec)
```

パラメータ

PARTITION (partition_spec)

各 partition_spec は、列名/値の組み合わせを partition_col_name = partition_col_value [,...] という形式で指定します。

例

```
ALTER TABLE orders
PARTITION (dt = '2014-05-14', country = 'IN') RENAME TO PARTITION (dt = '2014-05-15',
 country = 'IN');
```

AWS Glue でのパーティション列の名前の変更

AWS Glue コンソールでパーティション列の名前を変更するには、次の手順を実行します。

AWS Glue コンソールでテーブルパーティション列の名前を変更する

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[Tables (テーブル)] を選択します。
3. [テーブル] ページにある [テーブルをフィルタリング] 検索ボックスを使用して、変更するテーブルを見つけます。
4. [名前] 列で、変更するテーブルのリンクを選択します。
5. テーブルの詳細ページにある [スキーマ] セクションで、次のいずれかを実行します。
   - JSON 形式で名前を変更するには、[JSON としてスキーマを編集] を選択します。
   - 名前を直接変更するには、[スキーマを編集] を選択します。この手順では、[スキーマを編集] を選択します。
6. 名前を変更するパーティション分割された列のチェックボックスにチェックを入れてから、[編集] を選択します。
7. [スキーマエントリの編集] ダイアログボックスにある [名前] に、パーティション列の新しい名前を入力します。
8. [新しいテーブルバージョンとして保存] を選択します。このアクションは、パーティション列の名前を更新し、データの個別の物理的コピーを作成することなくスキーマの進化履歴を維持します。
9. テーブルバージョンを比較するには、テーブルの詳細ページで [アクション] を選択してから、[バージョンを比較] を選択します。

追加リソース

パーティショニングについての詳細は、「Athena でのデータのパーティション化」を参照してください。

ALTER TABLE REPLACE COLUMNS

LazySimpleSerDe で作成したテーブルから既存の列をすべて削除して、それらを指定された列のセットに置き換えます。オプションの `PARTITION` 構文を使用すると、パーティションメタデータが

更新されます。ALTER TABLE REPLACE COLUMNS を使用し、保持したい列のみを指定することによって、列をドロップすることもできます。

概要

```
ALTER TABLE table_name
  [PARTITION
   (partition_col1_name = partition_col1_value
   [,partition_col2_name = partition_col2_value][,...])]
  REPLACE COLUMNS (col_name data_type [, col_name data_type, ...])
```

パラメータ

PARTITION (partition_col_name = partition_col_value [,...])

ユーザー指定の列名/値の組み合わせでパーティションを指定します。列のデータ型が文字列である場合にのみ、`partition_col_value` を引用符で囲みます。

REPLACE COLUMNS (col_name data_type [,col_name data_type,...])

既存の列を、指定された列名とデータ型に置き換えます。

メモ

- ALTER TABLE REPLACE COLUMNS を実行した後で Athena クエリエディタのナビゲーションペインにテーブル列での変更を表示する場合に、エディタ内でテーブルのリストを手動で更新してから、テーブルの展開が再度必要となる場合があります。
- ALTER TABLE REPLACE COLUMNS は、`date` データ型の列では機能しません。この問題の回避策は、テーブルで `timestamp` データ型を代わりに使用することです。
- 単一の列だけを置き換える場合でも、構文は ALTER TABLE *table-name* REPLACE COLUMNS (複数形で columns を含む形式) とする必要があることに注意してください。置き換えの対象である列だけでなく、維持しておく列も指定する必要があります。これを行わない場合、指定されていない列は削除されます。この構文と動作は Apache Hive DDL から派生したものです。リファレンスについては、Apache ドキュメントの「Add/Replace Columns」(列の追加/置換) を参照してください。

例

次の例での (LazySimpleSerDe により作成された) テーブル names_cities には、col1、col2、および col3 という名前の 3 つの列があります。すべての列の型は string です。テーブル内の列を表示するために、次のコマンドでは SHOW COLUMNS ステートメントを使用しています。

```
SHOW COLUMNS IN names_cities
```

クエリの結果:

```
col1
col2
col3
```

次の ALTER TABLE REPLACE COLUMNS コマンドでは、列の名前を first_name、last_name、および city に置き換えます。基盤となるソースデータは影響を受けません。

```
ALTER TABLE names_cities
REPLACE COLUMNS (first_name string, last_name string, city string)
```

結果をテストするために、再度 SHOW COLUMNS が実行されます。

```
SHOW COLUMNS IN names_cities
```

クエリの結果:

```
first_name
last_name
city
```

更新された列名を表示する別の方法として、Athena クエリエディタでテーブルをプレビューする、または独自のSELECTクエリを実行することもできます。

ALTER TABLE SET LOCATION

table_name という名前のテーブルの場所と、必要に応じて partition_spec のパーティションを変更します。

概要

```
ALTER TABLE table_name [ PARTITION (partition_spec) ] SET LOCATION 'new location'
```

パラメータ

PARTITION (partition_spec)

場所を変更するパーティションを partition_spec パラメータで指定します。partition_spec は、列名/値の組み合わせを partition_col_name = partition_col_value という形式で指定します。

SET LOCATION 'new location'

新しい場所を指定します。Amazon S3 の場所にする必要があります。構文については、「Amazon S3 のテーブルの場所」を参照してください。

例

```
ALTER TABLE customers PARTITION (zip='98040', state='WA') SET LOCATION 's3://DOC-
EXAMPLE-BUCKET/custdata/';
```

ALTER TABLE SET TBLPROPERTIES

カスタムまたは事前定義されたメタデータプロパティをテーブルに追加して、それらの割り当てられた値を設定します。テーブル内のプロパティを表示するには、SHOW TBLPROPERTIES コマンドを使用します。

Apache Hive のマネージドテーブルはサポートされていないため、'EXTERNAL'='FALSE' を設定しても効果はありません。

概要

```
ALTER TABLE table_name SET TBLPROPERTIES ('property_name' = 'property_value' [ , ... ])
```

パラメータ

SET TBLPROPERTIES ('property_name' = 'property_value' [, ...])

追加するメタデータプロパティを property_name として指定し、各プロパティの値を property value として指定します。property_name が既に存在する場合は、その値が新たに指定された property_value に設定されます。

以下の事前定義されたテーブルプロパティには、特別な用途があります。

事前定義プロパティ	説明
classification	AWS Glue のデータ型を示します。可能な値は csv、parquet、orc、avro、または json です。CloudTrail コンソールで Athena 用に作成されたテーブルは、classification プロパティの値として cloudtrail を追加します。詳細については、「CREATE TABLE」の「TBLPROPERITERS」セクションを参照してください。
has_encrypted_data	LOCATION によって指定されたデータセットが暗号化されているかどうかを示します。詳細については、「CREATE TABLE」の「TBLPROPERITERS」セクション、および「Amazon S3 内の暗号化されたデータセットに基づくテーブルの作成」を参照してください。
orc.compress	ORC 形式のデータに対する圧縮形式を指定します。詳細については、「」を参照してくださいORC SerDe
parquet.compression	Parquet 形式のデータに対する圧縮形式を指定します。詳細については、「Parquet SerDe」を参照してください。
write.compression	テキストファイル、または JSON 形式のデータに対する圧縮形式を指定します。Parquet 形式と ORC 形式の場合は、それぞれに応じて parquet.compression もしくは orc.compress プロパティを使用します。
compression_level	使用する圧縮レベルを指定します。このプロパティは、ZSTD 圧縮にのみ適用されます。有効な値は 1 から 22 です。デフォルト値は 3 です。

事前定義プロパティ	説明
	詳細については、「Athena での ZSTD 圧縮レベルの使用」を参照してください。
projection.*	パーティション射影で使用されるカスタムプロパティで、Athena がテーブルでクエリを実行するときに、どのパーティションパターンを期待すべきかを把握できるようにします。詳細については、「」を参照してくださいAmazon Athena でのパーティション射影
skip.header.line.count	テーブルを定義するときに、データのヘッダーを無視します。詳細については、「」を参照してくださいヘッダーの無視
storage.location.template	射影されたパーティションに対してカスタム Amazon S3 パステンプレートを指定します。詳細については、「パーティション射影のセットアップ」を参照してください。

例

次の使用例では、テーブルプロパティにコメントノートを追加します。

```
ALTER TABLE orders
SET TBLPROPERTIES ('notes'="Please don't drop this table.");
```

次の例では、テーブル existing_table を変更して、ZSTD 圧縮および ZSTD 圧縮レベル 4 の Parquet ファイル形式を使用します。

```
ALTER TABLE existing_table
SET TBLPROPERTIES ('parquet.compression' = 'ZSTD', 'compression_level' = 4)
```

CREATE DATABASE

データベースを作成します。DATABASE と SCHEMA はどちらでも使用できます。どちらも同じ意味です。

> **Note**
>
> データベースの作成、テーブルの作成、および Athena のテーブルに対する SELECT クエリの実行の例については、開始 を参照してください。

概要

```
CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] database_name
  [COMMENT 'database_comment']
  [LOCATION 'S3_loc']
  [WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]
```

パラメータ

[IF NOT EXISTS]

database_name という名前のデータベースが既に存在する場合、エラーメッセージを抑制します。

[COMMENT database_comment]

comment という組み込みメタデータプロパティにメタデータ値を設定し、database_comment に値を指定します。AWS Glue では、COMMENT の内容がデータベースプロパティの Description フィールドに書き込まれます。

[LOCATION S3_loc]

データベースファイルとメタストアがある場所を S3_loc として指定します。これは Amazon S3 の場所にする必要があります。

[WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]

データベース定義のカスタムメタデータプロパティを指定します。

例

```
CREATE DATABASE clickstreams;
```

```
CREATE DATABASE IF NOT EXISTS clickstreams
```

```
  COMMENT 'Site Foo clickstream data aggregates'
  LOCATION 's3://DOC-EXAMPLE-BUCKET/clickstreams/'
  WITH DBPROPERTIES ('creator'='Jane D.', 'Dept.'='Marketing analytics');
```

データベースプロパティの表示

CREATE DATABASE を使用して AWSDataCatalog で作成するデータベースのデータベースプロパティを表示するには、以下の例にあるように、AWS CLI コマンドの aws glue get-database を使用することができます。

```
aws glue get-database --name <your-database-name>
```

JSON 出力では、結果が以下のようになります。

```
{
    "Database": {
        "Name": "<your-database-name>",
        "Description": "<your-database-comment>",
        "LocationUri": "s3://DOC-EXAMPLE-BUCKET",
        "Parameters": {
            "<your-database-property-name>": "<your-database-property-value>"
        },
        "CreateTime": 1603383451.0,
        "CreateTableDefaultPermissions": [
            {
                "Principal": {
                    "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                },
                "Permissions": [
                    "ALL"
                ]
            }
        ]
    }
}
```

AWS CLI の詳細については、AWS Command Line Interface ユーザーガイドを参照してください。

CREATE TABLE

指定した名前とパラメータでテーブルを作成します。

> **Note**
>
> このページには、要約されたリファレンス情報が含まれています。Athena でのテーブルの作成と CREATE TABLE ステートメントの例に関する詳細については、「Athena でのテーブルの作成」を参照してください。データベースの作成、テーブルの作成、および Athena のテーブルに対する SELECT クエリの実行の例については、「開始」を参照してください。

概要

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
 [db_name.]table_name [(col_name data_type [COMMENT col_comment] [, ...] )]
 [COMMENT table_comment]
 [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
 [CLUSTERED BY (col_name, col_name, ...) INTO num_buckets BUCKETS]
 [ROW FORMAT row_format]
 [STORED AS file_format]
 [WITH SERDEPROPERTIES (...)]
 [LOCATION 's3://DOC-EXAMPLE-BUCKET/[folder]/']
 [TBLPROPERTIES ( ['has_encrypted_data'='true | false',]
 ['classification'='aws_glue_classification',] property_name=property_value [, ...] ) ]
```

パラメータ

EXTERNAL

ユーザー指定の LOCATION で、テーブルが Amazon S3 に存在する基盤データに基づいていることを指定します。Iceberg テーブルの作成時を除いて、常に EXTERNAL キーワードが使用されます。Iceberg 以外のテーブルで、EXTERNAL キーワードを指定せずに CREATE TABLE を使用すると、Athena でエラーが発生します。外部テーブルを作成する場合、参照先のデータはデフォルト形式に準拠しているか、ユーザーが ROW FORMAT、STORED AS、および WITH SERDEPROPERTIES で指定する形式に準拠している必要があります。

[IF NOT EXISTS]

このパラメータは、同じ名前のテーブルが既に存在するかどうかを確認します。その場合、パラメータは TRUE を返し、Amazon Athena は CREATE TABLE アクションをキャンセルします。キャンセルは Athena がデータカタログを呼び出す前に行われるため、AWS CloudTrail イベントは発生しません。

[db_name.]table_name

作成するテーブルの名前を指定します。オプションの db_name パラメータは、テーブルを作成する先のデータベースを指定します。この引数を省略すると、現在のデータベースが使用されます。テーブル名に数字を含める場合は、table_name を引用符で囲みます (例: "table123")。table_name をアンダースコアで始める場合は、バックティックを使用します (例: `_mytable`)。特殊文字 (アンダースコア以外) はサポートされていません。

Athena のテーブル名では大文字と小文字が区別されませんが、Apache Spark を使用する場合、Spark ではテーブル名を小文字にする必要があります。

[(col_name data_type [COMMENT col_comment] [, ...])]

作成する各列の名前とデータ型を指定します。列名に特殊文字 (アンダースコア (_) を除く) を使用することはできません。col_name をアンダースコアで始める場合は、列名をバックティックで囲みます (例: `_mycolumn`)。

data_type 値には次のいずれかを指定できます。

- boolean – 値は true もしくは false です。
- tinyint – 2 の補数形式の 8 ビット符号付き整数で、最小値は -2^7、最大値は 2^7-1 です。
- smallint – 2 の補数形式の 16 ビット符号付き整数で、最小値は -2^15、最大値は 2^15-1 です。
- int – CREATE TABLE のようなデータ定義言語 (DDL) のクエリでは、整数を表すために int キーワードを使用します。他のクエリでは、integer キーワードを使用します。ここで integer は 2 の補数形式の 32 ビット符号付き値で表され、最小値は -2^31、最大値は 2^31-1 です。JDBC ドライバーでは、ビジネス分析アプリケーションとの互換性を確保するために integer が返されます。
- bigint – 2 の補数形式の 64 ビット符号付き整数で、最小値は -2^63、最大値は 2^63-1 です。
- double – 64 ビットの符号付き倍精度浮動小数点数です。これは、4.94065645841246544e-324d から 1.79769313486231570e+308d の範囲の正または負の値です。double は、IEEE Standard for Floating-Point Arithmetic (IEEE 754) に準拠しています。
- float – 32 ビットの符号付き単精度浮動小数点数です。これは、1.40129846432481707e-45 から 3.40282346638528860e+38 の範囲の正または負の値です。floatは、IEEE Standard for Floating-Point Arithmetic (IEEE 754) に準拠しています。Presto の real に相当します。Athena では、CREATE TABLE のような DDL ステートメントで float、SELECT CAST

のような SQL 関数で real を使用します。AWS Glue クローラは float で値を返し、Athena は real 型と float 型を内部で変換します (2018 年 5 月 6 日 リリースノートを参照)。

- decimal [(*precision*, *scale*)]。ここで *precision* は桁の合計数であり、*scale* (オプション) は小数点以下の桁数です。デフォルトは 0 です。たとえば、decimal(11,5)、decimal(15) のタイプ定義を使用します。*precision* の最大値は 38 で、*scale* の最大値も 38 です。

  たとえばクエリ DDL 式で特定の小数値を含む行を選択する場合など、小数値をリテラルとして指定するには、decimal 型の定義を指定し、クエリ内で小数値をリテラル (一重引用符) としてリストします。例としては decimal_value = decimal '0.12' のようになります。
- char – 固定長の文字データです。char(10) のように、1 から 255 の長さを指定します。詳細については、「CHAR Hive データ型」を参照してください。
- varchar – 可変長の文字データです。varchar(10) のように、1 から 65535 の長さを指定します。詳細については、「VARCHAR Hive データ型」を参照してください。
- string – 一重引用符または二重引用符で囲まれた文字列リテラルです。

  > **Note**
  >
  > 文字列以外のデータ型を Athena の string にキャストすることはできません。その代わりに varchar にキャストします。

- binary – (Parquet のデータ用)
- date – *YYYY*-*MM*-*DD* などの ISO 形式の日付です。例えば、date '2008-09-15' と指定します。例外として、1970 年 1 月 1 日以降の経過日数を使用する OpenCSVSerDe があります。詳細については、「CSV を処理するための OpenCSVSerDe」を参照してください。
- timestamp – 分解能をミリ秒単位まで指定可能な、java.sql.Timestamp 互換形式での日付と時刻のインスタント (例、*yyyy*-*MM*-*dd* *HH*:*mm*:*ss*[.f...]) です。例えば、timestamp '2008-09-15 03:04:05.324' と指定します。例外として、UNIX の数値形式 (1579059880000 など) の TIMESTAMP データを使用する OpenCSVSerDe があります。詳細については、「CSV を処理するための OpenCSVSerDe」を参照してください。
- array < data_type >
- map < primitive_type, data_type >
- struct< col_name : data_type [comment col_comment] [, ...]>

[COMMENT table_comment]

comment テーブルプロパティを作成し、指定した table_comment を追加します。

[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]

`col_name`、`data_type` および `col_comment` が指定された 1 つ以上の列を持つ、パーティションされたテーブルを作成します。テーブルは、個別の列名と値の組み合わせで構成されるパーティションを 1 つ以上持つことができます。指定した組み合わせごとに別個のデータディレクトリが作成されます。これにより、クエリパフォーマンスが向上する場合があります。パーティション化された列はテーブルデータ内には存在しません。テーブル列と同じ `col_name` の値を使用する場合は、エラーになります。詳細については、「データのパーティション」を参照してください。

> **Note**
>
> パーティションされたテーブルを作成したら、MSCK REPAIR TABLE 句で構成されるクエリを続けて実行し、パーティションメタデータ (`MSCK REPAIR TABLE cloudfront_logs;` など) を更新します。Hive と互換性のないパーティションの場合、ALTER TABLE ADD PARTITION を使用してパーティションをロードすることで、データをクエリできるようにします。

[CLUSTERED BY (col_name, col_name, ...) INTO num_buckets BUCKETS]

パーティションの有無にかかわらず、指定された `col_name` 列のデータをバケットと呼ばれるデータサブセットに分割します。`num_buckets` パラメータは、作成するバケットの数を指定します。バケット化は、大規模なデータセットでの一部のクエリのパフォーマンスを向上させることができます。

[ROW FORMAT row_format]

テーブルの行形式と基になるソースデータ (該当する場合) を指定します。`row_format` として、`DELIMITED` 句で 1 つ以上の区切り記号を指定できます。または、以下に説明するように、`SERDE` 句を使用できます。`ROW FORMAT` を省略するか、`ROW FORMAT DELIMITED` を指定すると、ネイティブ SerDe が使用されます。

- [DELIMITED FIELDS TERMINATED BY char [ESCAPED BY char]]
- [DELIMITED COLLECTION ITEMS TERMINATED BY char]
- [MAP KEYS TERMINATED BY char]
- [LINES TERMINATED BY char]
- [NULL DEFINED AS char]

Hive 0.13 で、STORED AS ファイル形式が TEXTFILE である場合にのみ使用できます。

--または--

- SERDE 'serde_name' [WITH SERDEPROPERTIES ("property_name" = "property_value", "property_name" = "property_value" [, ...])]

  serde_name は、使用する SerDe を示します。WITH SERDEPROPERTIES 句を使用すると、SerDe で許可される 1 つ以上のカスタムプロパティを指定できます。

[STORED AS file_format]

テーブルデータのファイル形式を指定します。省略すると、デフォルトの TEXTFILE が使用されます。file_format のオプションは以下のとおりです。

- SEQUENCEFILE
- TEXTFILE
- RCFILE
- ORC
- PARQUET
- AVRO
- ION
- INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

[LOCATION 's3://DOC-EXAMPLE-BUCKET/[folder]/']

テーブルが作成される Amazon S3 内の基盤データの場所を指定します。ロケーションパスは、バケット名、またはバケット名と 1 つ以上のフォルダである必要があります。パーティションを使用する場合は、パーティションされたデータのルートを指定します。テーブルのロケーションの詳細については、「Amazon S3 のテーブルの場所」を参照してください。データ形式と許可については、「Athena のテーブルと Amazon S3 のデータに関する要件」を参照してください。

フォルダやバケットの後にはスラッシュを使用します。ファイル名や glob 文字を使用しないでください。

を使用します。

s3://DOC-EXAMPLE-BUCKET/

s3://DOC-EXAMPLE-BUCKET/*folder*/

s3://DOC-EXAMPLE-BUCKET/*folder*/*anotherfolder*/

使用不可:

```
s3://DOC-EXAMPLE-BUCKET
```

```
s3://DOC-EXAMPLE-BUCKET/*
```

```
s3://DOC-EXAMPLE-BUCKET/mydatafile.dat
```

[TBLPROPERTIES (['has_encrypted_data'='true | false',] ['classification'='classification_value',] property_name=property_value [, ...])]

定義済みのテーブルプロパティ ("comment" など) に加えて、テーブル定義のカスタムメタデータとしてキーと値のペアを指定します。

has_encrypted_data — Athena には、has_encrypted_data という組み込みプロパティがあります。このプロパティを true に設定し、LOCATION で指定する基になるデータを暗号化することを指定します。省略するか、そのワークグループの設定がクライアント側の設定を上書きしない場合、false が設定されます。省略するか、false に設定して基になるデータを暗号化すると、クエリはエラーになります。詳細については、「保管中の暗号化」を参照してください。

分類 — CloudTrail コンソールで Athena 用に作成されたテーブルは、classification プロパティの値として cloudtrail を追加されます。ETL ジョブを実行する場合、AWS Glue では、テーブルの作成時に classification プロパティを設定し、AWS Glue のデータ型を csv、parquet、orc、avro、json のいずれかに指定する必要があります。例えば、'classification'='csv' と指定します。このプロパティを指定しないと、ETL ジョブは失敗します。AWS Glue コンソール、API、または CLI を使用して後で指定できます。詳細については、「Athena における ETL 用の AWS Glue ジョブの使用」と、「AWS Glue デベロッパーガイド」の「AWS Glue でジョブを作成する」を参照してください。

compression_level — compression_level プロパティは使用する圧縮レベルを指定します。このプロパティは、ZSTD 圧縮にのみ適用されます。有効な値は 1 から 22 です。デフォルト値は 3 です。詳細については、「Athena での ZSTD 圧縮レベルの使用」を参照してください。

その他のテーブルプロパティの詳細については、「ALTER TABLE SET TBLPROPERTIES」を参照してください。

例

次のサンプル CREATE TABLE ステートメントは、Amazon S3 に保存されているタブ区切りの惑星データに基づいてテーブルを作成します。

```
CREATE EXTERNAL TABLE planet_data (
  planet_name string,
  order_from_sun int,
  au_to_sun float,
  mass float,
  gravity_earth float,
  orbit_years float,
  day_length float
  )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/tsv/'
```

以下の点に注意してください。

- ROW FORMAT DELIMITED 句は、データが特定の文字で区切られていることを示します。
- FIELDS TERMINATED BY '\t' 句は、TSV データ内のフィールドがタブ文字 ('\t') で区切られることを指定します。
- STORED AS TEXTFILE 句は、データがプレーンテキストファイルとして Amazon S3 に保存されることを示します。

データをクエリするには、次のようなシンプルな SELECT ステートメントを使用できます。

```
SELECT * FROM planet_data
```

この例を使用して Athena で独自の TSV テーブルを作成するには、テーブル名と列名を独自のテーブルと列の名前とデータ型に置き換えてから、TSV ファイルが保存されている Amazon S3 パスをポイントするように LOCATION 句を更新します。

テーブル作成の詳細については、「Athena でのテーブルの作成」を参照してください。

CREATE TABLE AS

SELECT クエリの結果が入力された新しいテーブルを作成します。空のテーブルを作成するには、CREATE TABLE を使用します。CREATE TABLE AS は CREATE TABLE DDL ステートメントと SELECT DML ステートメントを組み合わせているため、技術的には DDL と DML の両方を含んでいます。ここでは CREATE TABLE AS が他の DDL ステートメントとまとめられている

が、Athena の CTAS クエリは、Service Quotas の目的で DML として扱われることに注意してください。Athena の Service Quotasの詳細については、「Service Quotas」を参照してください。

> **Note**
>
> CTAS ステートメントの場合、予想されるバケット所有者の設定は、Amazon S3 内の送信先テーブルの場所には適用されません。予期されるバケット所有者の設定は、Athena クエリの結果の出力先として指定した Amazon S3 内の場所にのみ適用されます。詳細については、「Athena コンソールを使用したクエリ結果の場所の指定」を参照してください。

このリファレンストピックの対象外である CREATE TABLE AS に関する追加情報については、「クエリ結果からのテーブルの作成 (CTAS)」を参照してください。

トピック

概要

```
CREATE TABLE table_name
[ WITH ( property_name = expression [, ...] ) ]
AS query
[ WITH [ NO ] DATA ]
```

コードの説明は以下のとおりです。

WITH (property_name = expression [, ...])

オプションの CTAS テーブルのプロパティのリスト。一部のプロパティはデータストレージ形式に固有です。「CTAS テーブルのプロパティ」を参照してください。

query

新しいテーブルの作成に使用される SELECT クエリです。

> **Important**
>
> パーティションがあるクエリを作成する予定がある場合は、SELECT ステートメントで列のリストの最後に、パーティションされた列の名前を指定します。

[WITH [NO] DATA]

`WITH NO DATA` が指定されている場合、元のテーブルと同じスキーマを持つ新しい空のテーブルが作成されます。

> **Note**
>
> クエリの結果出力に列ヘッダーを含めるには、CTAS クエリの代わりに単純な SELECT クエリを実行します。クエリ結果の場所から結果を取得するか、Athena コンソールを使用して結果を直接ダウンロードすることができます。詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」を参照してください。

CTAS テーブルのプロパティ

Athena の各 CTAS テーブルには、`WITH (property_name = expression [, ...] )` を使用して指定するオプションの CTAS テーブルプロパティのリストがあります。これらのパラメータの使用方法については、「CTAS クエリの例」を参照してください。

`WITH (property_name = expression [, ...], )`

`table_type = ['HIVE', 'ICEBERG']`

オプション。デフォルト: `HIVE`。生成されるテーブルのテーブルタイプを指定します

例：

```
WITH (table_type ='ICEBERG')
```

external_location = [location]

> **Note**
>
> Iceberg テーブルは外部ではないため、このプロパティは Iceberg テーブルには適用されません。CTAS ステートメントで Iceberg テーブルのルートロケーションを定義するには、このセクションで後述する `location` プロパティを使用します。

オプション。Athena が CTAS クエリを Amazon S3 に保存する場所です。

例：

```
WITH (external_location ='s3://DOC-EXAMPLE-BUCKET/tables/parquet_table/')
```

Athena は、クエリ結果に同じパスを 2 回使用しません。場所を手動で指定する場合は、指定する Amazon S3 の場所にデータがないことを確認してください。Athena がデータの削除を試みることはありません。再度同じ場所を使用する場合は、手動でデータを削除します。そうしないと、CTAS クエリは失敗します。

クエリ結果の場所を強制するワークグループで `external_location` を指定する CTAS クエリを実行すると、クエリはエラーメッセージを伴って敗します。ワークグループに対して指定されたクエリ結果の場所を確認するには、ワークグループの詳細を参照してください。

ワークグループがクエリ結果の場所に関するクライアント側の設定を上書きする場合、Athena は以下の場所にテーブルを作成します。

```
s3://DOC-EXAMPLE-BUCKET/tables/query-id/
```

`external_location` プロパティを使用して場所を指定せず、ワークグループがクライアント側の設定を上書きしない場合、Athena はクエリ結果の場所にクライアント側の設定を使用して、以下の場所にテーブルを作成します。

```
s3://DOC-EXAMPLE-BUCKET/Unsaved-or-query-name/year/month/date/tables/query-id/
```

is_external = [boolean]

オプション。テーブルが外部テーブルかどうかを示します。デフォルトは true です。Iceberg テーブルの場合、これを false に設定する必要があります。

例：

```
WITH (is_external = false)
```

location = [location]

Iceberg テーブルには必須です。クエリ結果から作成される Iceberg テーブルのルート位置を指定します。

例：

```
WITH (location ='s3://DOC-EXAMPLE-BUCKET/tables/iceberg_table/')
```

field_delimiter = [delimiter]

オプションであり、テキストベースのデータストレージ形式に固有のパラメータ。CSV と TSV のファイル、およびテキストファイルのための単一文字のフィールド区切り文字で、例えば、 。WITH (field_delimiter = ',')現在、CTAS クエリは複数文字のフィールド区切り文字をサポートしてません。フィールド区切り文字を指定しなかった場合、デフォルトで \001 が使用されます。

format = [storage_format]

CTAS クエリ結果のストレージ形式 (例: ORC、PARQUET、AVRO、JSON、ION、または TEXTFILE)。Iceberg テーブルで使用できる形式は ORC、PARQUET、および AVRO です。省略した場合は、デフォルトで PARQUET が使用されます。このパラメータの名前である format は小文字で記述する必要があります。そうしないと、CTAS クエリは失敗します。

例：

```
WITH (format = 'PARQUET')
```

bucketed_by = ARRAY[column_name[,…], bucket_count = [int]]

> **Note**
>
> このプロパティは Iceberg テーブルには適用されません。Iceberg テーブルの場合は、バケット変換によるパーティショニングを使用してください。

データの配置先となるバケットの配列リスト。省略された場合、Athena はこのクエリでデータをバケット化しません。

bucket_count = [int]

> **Note**
>
> このプロパティは Iceberg テーブルには適用されません。Iceberg テーブルの場合は、バケット変換によるパーティショニングを使用してください。

データの配置先となるバケットの数。省略された場合、Athena はデータをバケット化しません。例：

```
CREATE TABLE bucketed_table WITH (
  bucketed_by = ARRAY[column_name],
  bucket_count = 30, format = 'PARQUET',
  external_location ='s3://DOC-EXAMPLE-BUCKET/tables/parquet_table/'
) AS
SELECT
  *
FROM
  table_name
```

partitioned_by = ARRAY[col_name[,…]]

> **Note**
>
> このプロパティは Iceberg テーブルには適用されません。Iceberg テーブルにパーティション変換を使用するには、このセクションで後述する `partitioning` プロパティを使用してください。

オプション。CTAS テーブルをパーティションする列の配列リスト。パーティションされた列の名前は、SELECT ステートメントで列のリストの最後に記述します。

partitioning = ARRAY[partition_transform, ...]

オプション。作成する Iceberg テーブルのパーティショニングを指定します。Iceberg は、さまざまなパーティション変換およびパーティションの進化に対応しています。パーティション変換を次の表に要約します。

変換	説明
year(ts)	各年度のパーティションを作成します。パーティション値は、ts から 1970 年 1 月 1 日までの年数の差を整数で表したものです。
month(ts)	各年度の各月にパーティションを作成します。パーティション値は、ts から 1970 年 1 月 1 日までの月数の差を整数で表したものです。
day(ts)	各年度の各日にパーティションを作成します。パーティション値は、ts から 1970 年 1 月 1 日までの日数の差を整数で表したものです。
hour(ts)	各日の各時間帯にパーティションを作成します。パーティション値は、分と秒をゼロに設定したタイムスタンプです。
bucket(x, nbuckets)	データを指定された数のバケットにハッシュします。パーティション値は、0 から nbuckets - 1 までの値を持つ、x の整数ハッシュです。
truncate(s, nchars)	パーティション値を s の最初の nchars 文字にします。

例：

```
WITH (partitioning = ARRAY['month(order_date)',
                           'bucket(account_number, 10)',
                           'country']))
```

optimize_rewrite_min_data_file_size_bytes = [long]

オプション。データ最適化固有の設定。指定した値より小さいファイルは、最適化のために含まれます。デフォルトは write_target_data_file_size_bytes の 0.75 倍の値です。このプロパティは Iceberg テーブルにのみ適用されます。詳細については、「Iceberg テーブルの最適化」を参照してください。

例：

```
WITH (optimize_rewrite_min_data_file_size_bytes = 402653184)
```

optimize_rewrite_max_data_file_size_bytes = [long]

オプション。データ最適化固有の設定。指定した値より大きいファイルは、最適化のために含まれます。デフォルトは write_target_data_file_size_bytes の 1.8 倍の値です。このプロパティは Iceberg テーブルにのみ適用されます。詳細については、「Iceberg テーブルの最適化」を参照してください。

例：

```
WITH (optimize_rewrite_max_data_file_size_bytes = 966367641)
```

optimize_rewrite_data_file_threshold = [int]

オプション。データ最適化固有の設定。最適化が必要なデータファイルの数が指定されたしきい値より少ない場合、データファイルは書き換えられません。これにより、蓄積するデータファイルの数を増やしてターゲットサイズに近いファイルを生成し、不要な計算をスキップしてコストを削減できます。デフォルトは 5 です。このプロパティは Iceberg テーブルにのみ適用されます。詳細については、「Iceberg テーブルの最適化」を参照してください。

例：

```
WITH (optimize_rewrite_data_file_threshold = 5)
```

optimize_rewrite_delete_file_threshold = [int]

オプション。データ最適化固有の設定。データファイルに関連付けられた削除ファイルの数がしきい値より少ない場合、データファイルは書き換えられません。これにより、データファイルごとに蓄積する削除ファイルの数を増やしてコストを削減できます。デフォルトは 2 です。このプロパティは Iceberg テーブルにのみ適用されます。詳細については、「Iceberg テーブルの最適化」を参照してください。

例：

```
WITH (optimize_rewrite_delete_file_threshold = 2)
```

vacuum_min_snapshots_to_keep = [int]

オプション。バキューム固有の設定。保持する最新のスナップショットの最小数。デフォルト は 1 です。このプロパティは Iceberg テーブルにのみ適用されます。詳細については、「VACUUM」を参照してください。

> **Note**
>
> vacuum_min_snapshots_to_keep プロパティには、Athena エンジンバージョン 3 が必要です。

例：

```
WITH (vacuum_min_snapshots_to_keep = 1)
```

vacuum_max_snapshot_age_seconds = [long]

オプション。バキューム固有の設定。保持するスナップショットの経過時間を表す秒単位の期間。デフォルトは 432,000 (5 日) です。このプロパティは Iceberg テーブルにのみ適用されます。詳細については、「VACUUM」を参照してください。

> **Note**
>
> vacuum_max_snapshot_age_seconds プロパティには、Athena エンジンバージョン 3 が必要です。

例：

```
WITH (vacuum_max_snapshot_age_seconds = 432000)
```

write_compression = [compression_format]

圧縮を指定できるストレージ形式に使用するための圧縮タイプ。compression_format は、テーブルへのデータの書き込み時に使用される圧縮を指定します。TEXTFILE、JSON、PARQUET、および ORC のファイル形式に対して圧縮を指定できます。

例えば、format プロパティにおいてストレージ形式として PARQUET が指定されている場合、write_compression の値で Parquet での圧縮形式を指定します。この場合、write_compression の値を指定することは、parquet_compression の値を指定することと同等です。

同様に、format プロパティにおいてストレージ形式として ORC が指定されている場合には、write_compression の値で ORC の圧縮形式を指定します。この場合、write_compression の値を指定することは、orc_compression の値を指定することと同等です。

同じ CTAS クエリで、テーブルプロパティに複数の圧縮形式を指定することはできません。例えば、同一のクエリ内で、write_compression と parquet_compression の両方を指定することはできません。同様なことが write_compression と orc_compression にも当てはまります。各ファイル形式でサポートされている圧縮タイプの詳細については、「Athena での圧縮のサポート」を参照してください。

orc_compression = [compression_format]

テーブルへの ORC データ書き込み時に、ORC ファイル形式に対して使用される圧縮タイプ 例えば、WITH (orc_compression = 'ZLIB') と指定します。ORC ファイル (ORC Postscript を除く) 内のチャンクは、指定された圧縮を使用して圧縮されます。この指定を省略した場合、ORC に対してはデフォルトで ZLIB 圧縮が使用されます。

> **Note**
>
> 整合性を保つために、orc_compression の代わりに write_compression プロパティを使用することをお勧めします。format プロパティを使用して、ストレージ形式を ORC に指定した上で、write_compression プロパティにより、圧縮形式を ORC が使用するものに指定します。

parquet_compression = [compression_format]

Parquet データのテーブルへの書き込み時に、Parquet ファイル形式に対して使用される圧縮タイプ。例えば、WITH (parquet_compression = 'SNAPPY') と指定します。この圧縮は Parquet ファイル内の列チャンクに適用されます。この指定を省略した場合、Parquet に対してはデフォルトで GZIP 圧縮が使用されます。

> **Note**
>
> 整合性を保つために、parquet_compression の代わりに write_compression プロパティを使用することをお勧めします。format プロパティを使用して、ストレージ形式を PARQUET に指定した上で、write_compression プロパティにより、圧縮形式を PARQUET が使用するものに指定します。

compression_level = [compression_level]

使用する圧縮レベル。このプロパティは、ZSTD 圧縮にのみ適用されます。有効な値は 1 から 22 です。デフォルト値は 3 です。詳細については、「Athena での ZSTD 圧縮レベルの使用」を参照してください。

例

CTAS クエリの例については、次のリソースを参照してください。

- CTAS クエリの例
- ETL およびデータ分析での CTAS および INSERT INTO の使用
- コストを削減してパフォーマンスを向上させるために Amazon Athena で CTAS ステートメントを使用する
- CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する

CREATE VIEW

指定する SELECT クエリから新しいビューを作成します。このビューは、この先のクエリで参照できる論理的なテーブルです。ビューにはデータは一切含まれず、データを書き込むこともできません。代わりに、ビューで指定するクエリは、別のクエリでそのビューを参照するたびに実行されます。

> **Note**
>
> このトピックでは、参照用に概要情報を提供します。Athena でのビューの使用の詳細については、「ビューの使用」を参照してください。ビューの制限については、「ビューの制限」を参照してください。

概要

```
CREATE [ OR REPLACE ] VIEW view_name AS query
```

オプションの OR REPLACE 句は、既存のビューを更新して置き換えます。詳細については、「ビューの作成」を参照してください。

例

テーブル orders から test ビューを作成するには、次のようなクエリを使用します。

```
CREATE VIEW test AS
SELECT
orderkey,
orderstatus,
totalprice / 2 AS half
FROM orders;
```

テーブル orders から orders_by_date ビューを作成するには、次のクエリを使用します。

```
CREATE VIEW orders_by_date AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate;
```

既存のビューを更新するには、次のような例を使用します。

```
CREATE OR REPLACE VIEW test AS
SELECT orderkey, orderstatus, totalprice / 4 AS quarter
FROM orders;
```

「SHOW COLUMNS」、「SHOW CREATE VIEW」、「DESCRIBE VIEW」、「DROP VIEW」も参照してください。

DESCRIBE

指定したテーブルの 1 つ以上の列 (パーティション列を含む) を表示します。このコマンドは、複雑な列の属性を調べるのに便利です。

概要

```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]table_name [PARTITION partition_spec]
 [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

> **Important**
>
> このステートメントの構文は DESCRIBE *table_name* であり、DESCRIBE TABLE *table_name* ではありません。後者の構文を使用すると、エラーメッセージ FAILED: SemanticException [Error 10001]: Table not found table が表示されます。

パラメータ

[EXTENDED | FORMATTED]

出力の形式を判断します。これらのパラメータを省略すると、列名と対応するデータ型 (パーティション列を含む) が表形式で表示されます。FORMATTED を指定すると、列名とデータ型が表形式で表示されるだけでなく、詳細なテーブルおよびストレージ情報も表示されます。EXTENDED は、列およびデータ型の情報を表形式で表示し、Thrift シリアル化形式でテーブルの詳細なメタデータを示します。この形式は読みにくいですが、主としてデバッグの際に役立ちます。

[PARTITION partition_spec]

含まれている場合は、partition_spec で指定されたパーティションのメタデータが一覧表示されます。partition_spec は (partition_column = partition_col_value, partition_column = partition_col_value, ...) の形式です。

[col_name ([.field_name] | [.'$elem$'] | [.'key'] | [.'$value$'])*]

確認する列と属性を指定します。構造体の要素には .field_name、配列要素には '$elem$'、マップキーには 'key'、およびマップ値には '$value$' を指定できます。これを再帰的に指定することで、複雑な列を詳しく調査できます。

例

```
DESCRIBE orders
```

```
DESCRIBE FORMATTED mydatabase.mytable PARTITION (part_col = 100) columnA;
```

次のクエリおよび出力では、Amazon EMR サンプルデータに基づいた impressions テーブルの列およびデータ型の情報が表示されています。

```
DESCRIBE impressions
```

```
requestbegintime          string                                              from
 deserializer
adid                      string                                              from
 deserializer
impressionid              string                                              from
 deserializer
referrer                  string                                              from
 deserializer
useragent                 string                                              from
 deserializer
usercookie                string                                              from
 deserializer
ip                        string                                              from
 deserializer
number                    string                                              from
 deserializer
processid                 string                                              from
 deserializer
browsercokie              string                                              from
 deserializer
requestendtime            string                                              from
 deserializer
timers                    struct<modellookup:string,requesttime:string>  from
 deserializer
threadid                  string                                              from
 deserializer
hostname                  string                                              from
 deserializer
sessionid                 string                                              from
 deserializer
dt                        string

# Partition Information
# col_name                data_type               comment

dt                        string
```

次のクエリおよび出力の例では、FORMATTED オプションが使用された場合の同じテーブルの結果が表示されています。

```
DESCRIBE FORMATTED impressions
```

```
requestbegintime          string                                              from
 deserializer
adid                      string                                              from
 deserializer
impressionid              string                                              from
 deserializer
referrer                  string                                              from
 deserializer
useragent                 string                                              from
 deserializer
usercookie                string                                              from
 deserializer
ip                        string                                              from
 deserializer
number                    string                                              from
 deserializer
processid                 string                                              from
 deserializer
browsercokie              string                                              from
 deserializer
requestendtime            string                                              from
 deserializer
timers                    struct<modellookup:string,requesttime:string>  from
 deserializer
threadid                  string                                              from
 deserializer
hostname                  string                                              from
 deserializer
sessionid                 string                                              from
 deserializer
dt                        string

# Partition Information
# col_name                data_type                comment

dt                        string

# Detailed Table Information
```

```
Database:                   sampledb
Owner:                      hadoop
CreateTime:                 Thu Apr 23 02:55:21 UTC 2020
LastAccessTime:             UNKNOWN
Protect Mode:               None
Retention:                  0
Location:                   s3://us-east-1.elasticmapreduce/samples/hive-ads/tables/
impressions
Table Type:                 EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                    TRUE
        transient_lastDdlTime       1587610521

# Storage Information
SerDe Library:                          org.openx.data.jsonserde.JsonSerDe
InputFormat:                            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:
 org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat
Compressed:                             No
Num Buckets:                            -1
Bucket Columns:                         []
Sort Columns:                           []
Storage Desc Params:
        paths                                   requestbegintime, adid, impressionid,
 referrer, useragent, usercookie, ip
        serialization.format                    1
```

次のクエリおよび出力の例では、EXTENDED オプションが使用された場合の同じテーブルの結果が表示されています。詳細なテーブル情報は 1 行で出力されますが、ここでは読みやすくするためにフォーマットされています。

```
DESCRIBE EXTENDED impressions
```

```
requestbegintime          string                                              from
 deserializer
adid                      string                                              from
 deserializer
impressionid              string                                              from
 deserializer
referrer                  string                                              from
 deserializer
```

```
useragent                 string                                              from
 deserializer
usercookie                string                                              from
 deserializer
ip                        string                                              from
 deserializer
number                    string                                              from
 deserializer
processid                 string                                              from
 deserializer
browsercokie              string                                              from
 deserializer
requestendtime            string                                              from
 deserializer
timers                    struct<modellookup:string,requesttime:string>  from
 deserializer
threadid                  string                                              from
 deserializer
hostname                  string                                              from
 deserializer
sessionid                 string                                              from
 deserializer
dt                        string

# Partition Information
# col_name                data_type               comment

dt                        string

Detailed Table Information       Table(tableName:impressions, dbName:sampledb,
 owner:hadoop, createTime:1587610521,
lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:
[FieldSchema(name:requestbegintime, type:string, comment:null),
FieldSchema(name:adid, type:string, comment:null), FieldSchema(name:impressionid,
 type:string, comment:null),
FieldSchema(name:referrer, type:string, comment:null), FieldSchema(name:useragent,
 type:string, comment:null),
FieldSchema(name:usercookie, type:string, comment:null), FieldSchema(name:ip,
 type:string, comment:null),
FieldSchema(name:number, type:string, comment:null), FieldSchema(name:processid,
 type:string, comment:null),
FieldSchema(name:browsercokie, type:string, comment:null),
 FieldSchema(name:requestendtime, type:string, comment:null),
```

```
FieldSchema(name:timers, type:struct<modellookup:string,requesttime:string>,
 comment:null), FieldSchema(name:threadid,
type:string, comment:null), FieldSchema(name:hostname, type:string, comment:null),
 FieldSchema(name:sessionid,
type:string, comment:null)], location:s3://us-east-1.elasticmapreduce/samples/hive-ads/
tables/impressions,
inputFormat:org.apache.hadoop.mapred.TextInputFormat,
outputFormat:org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat, compressed:false,
 numBuckets:-1,
serdeInfo:SerDeInfo(name:null, serializationLib:org.openx.data.jsonserde.JsonSerDe,
 parameters:{serialization.format=1,
paths=requestbegintime, adid, impressionid, referrer, useragent, usercookie, ip}),
 bucketCols:[], sortCols:[], parameters:{},
skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[],
 skewedColValueLocationMaps:{}),
storedAsSubDirectories:false), partitionKeys:[FieldSchema(name:dt, type:string,
 comment:null)],
parameters:{EXTERNAL=TRUE, transient_lastDdlTime=1587610521}, viewOriginalText:null,
 viewExpandedText:null,
tableType:EXTERNAL_TABLE)
```

DESCRIBE VIEW

名前が付けられたビューの列のリストを表示します。これにより、複合型のビューの属性を確認できます。

概要

```
DESCRIBE [db_name.]view_name
```

例

```
DESCRIBE orders;
```

「SHOW COLUMNS」、「SHOW CREATE VIEW」、「SHOW VIEWS」、「DROP VIEW」も参照してください。

DROP DATABASE

指定したデータベースをカタログから削除します。データベースにテーブルが含まれている場合は、DROP DATABASE を実行する前にそれらのテーブルを削除するか、CASCADE 句を使用する必要があります。DATABASE と SCHEMA はどちらでも使用できます。どちらも同じ意味です。

概要

```
DROP {DATABASE | SCHEMA} [IF EXISTS] database_name [RESTRICT | CASCADE]
```

パラメータ

[IF EXISTS]

database_name が存在しない場合に、エラーを抑制します。

[RESTRICT|CASCADE]

database_name 内のテーブルを DROP オペレーションで処理する方法を決定します。RESTRICT を指定すると、テーブルが含まれているデータベースは削除されません。これがデフォルトの動作です。CASCADE を指定すると、データベースとそのすべてのテーブルが削除されます。

例

```
DROP DATABASE clickstreams;
```

```
DROP SCHEMA IF EXISTS clickstreams CASCADE;
```

> **Note**
>
> 名前に特殊文字 (例: my-database) を含むデータベースを削除しようとすると、エラーメッセージが表示されることがあります。この問題を解決するには、データベース名をバックォート文字 (`) で囲んでみてください。Athena でのデータベースの名前の付け方については、「テーブル、データベース、および列の名前」を参照してください。

DROP TABLE

table_name という名前のテーブルからメタデータのテーブル定義を削除します。外部テーブルをドロップすると、基礎となるデータはそのままです。

概要

```
DROP TABLE [IF EXISTS] table_name
```

パラメータ

[IF EXISTS]

table_name が存在しない場合に、エラーを抑制します。

例

```
DROP TABLE fulfilled_orders
```

```
DROP TABLE IF EXISTS fulfilled_orders
```

Athena コンソールのクエリエディタを使用してアンダースコア (_) 以外の特殊文字を含むテーブルをドロップする場合は、以下の例にあるように、バックティックを使用します。

```
DROP TABLE `my-athena-database-01.my-athena-table`
```

JDBC コネクタを使用して特殊文字を含むテーブルを削除する場合は、バックティック文字は必要ありません。

```
DROP TABLE my-athena-database-01.my-athena-table
```

DROP VIEW

既存のビューをドロップ (削除) します。オプションの IF EXISTS 句は、ビューが存在しない場合に、エラーが制御される原因となります。

詳細については、「ビューの使用」を参照してください。

概要

```
DROP VIEW [ IF EXISTS ] view_name
```

例

```
DROP VIEW orders_by_date
```

```
DROP VIEW IF EXISTS orders_by_date
```

CREATE VIEW、SHOW COLUMNS、SHOW CREATE VIEW、SHOW VIEWS および DESCRIBE VIEW も参照してください。

MSCK REPAIR TABLE

Hive 互換パーティションを追加した後でカタログ内のメタデータを更新するには、MSCK REPAIR TABLE コマンドを使用します。

MSCK REPAIR TABLE コマンドは、テーブルの作成後にファイルシステムに追加された Hive 互換パーティションで Simple Storage Service (Amazon S3) などのファイルシステムをスキャンします。MSCK REPAIR TABLE は、テーブルメタデータのパーティションと S3 のパーティションを比較します。テーブルの作成時に指定した S3 の場所に新しいパーティションが存在する場合は、これらのパーティションがメタデータと Athena のテーブルに追加されます。

物理パーティションを追加すると、カタログ内のメタデータがファイルシステム内のデータのレイアウトと整合しなくなるので、新しいパーティションに関する情報をカタログに追加する必要があります。メタデータを更新するには、Athena から新しいパーティションのデータをクエリできるように MSCK REPAIR TABLE を実行します。

> **Note**
>
> MSCK REPAIR TABLE は、パーティションをメタデータに追加するだけであり、パーティションを削除しません。Amazon S3 でパーティションが手動で削除された後でメタデータからそれらのパーティションを削除するには、コマンド ALTER TABLE *table-name* DROP PARTITION を実行します。詳細については、「ALTER TABLE DROP PARTITION」を参照してください。

考慮事項と制約事項

MSCK REPAIR TABLE を使用する際は、次のポイントに注意が必要です。

- すべてのパーティションを追加するには時間がかかる場合があります。このオペレーションがタイムアウトになると、未完了状態となり、一部のパーティションのみがカタログに追加されます。すべてのパーティションが追加されるまで、同じテーブルで MSCK REPAIR TABLE を実行してください。詳細については、「Athena でのデータのパーティション化」を参照してください。

- Hive と互換性のないパーティションの場合、ALTER TABLE ADD PARTITION を使用してパーティションをロードすることで、データをクエリできるようにします。
- Athena で使用されるパーティションの場所は、s3 プロトコル (s3://DOC-EXAMPLE-BUCKET/*folder*/ など) を使用する必要があります。Athena では、他のプロトコル (s3a://*bucket*/*folder*/ など) を使用する場所は、そこにあるテーブルに対して MSCK REPAIR TABLE クエリを実行する場合にクエリが失敗する原因になります。
- MSCK REPAIR TABLE がフォルダとそのサブフォルダの両方をスキャンして一致するパーティションスキームを検索するため、別個のテーブルのデータは別個のフォルダ階層に保存するようにしてください。例えば、テーブル 1 のデータが s3://DOC-EXAMPLE-BUCKET1 にあり、テーブル 2 のデータが s3://DOC-EXAMPLE-BUCKET1/table-2-data にあるとします。両方のテーブルが文字列でパーティション分割されている場合、MSCK REPAIR TABLE はテーブル 2 のパーティションをテーブル 1 に追加します。これを回避するには、この代わりに s3://DOC-EXAMPLE-BUCKET2 や s3://DOC-EXAMPLE-BUCKET1 といった別個のフォルダ構造を使用します。この動作は、Amazon EMR および Apache Hive と同じであることに注意してください。
- 既知の問題により、パーティション値にコロン (:) 記号が含まれている場合 (例えば、パーティション値がタイムスタンプの場合)、MSCK REPAIR TABLE は警告なしに失敗します。回避方法として、ALTER TABLE ADD PARTITION を使用します。
- MSCK REPAIR TABLE では、アンダースコア (_) で始まるパーティション列名は追加されません。この制限を回避するには、ALTER TABLE ADD PARTITION を使用します。

概要

```
MSCK REPAIR TABLE table_name
```

例

```
MSCK REPAIR TABLE orders;
```

トラブルシューティング

MSCK REPAIR TABLE の実行後、AWS Glue Data Catalog のテーブルにパーティションが追加されない場合は、以下をチェックしてください。

- AWS Glue アクセス – AWS Identity and Access Management (IAM) ロールに、glue:BatchCreatePartition アクションを許可するポリシーがあることを確認します。詳細については、このドキュメントで後述する「IAM ポリシーで glue:BatchCreatePartition を許可する」を参照してください。

- Amazon S3 アクセス – ロールに `s3:DescribeJob` アクションを含む Amazon S3 にアクセスするために、十分なアクセス許可を持つポリシーがあることを確認します。許可する Simple Storage Service (Amazon S3) アクションの例については、「Athena での Amazon S3 バケットへのクロスアカウントアクセス」のバケットポリシー例を参照してください。
- Amazon S3 オブジェクトキーの大文字と小文字の区別 – Amazon S3 パスがキャメルケースではなく小文字になっている (例: `userId` ではなく `userid`) ことを確認するか、`ALTER TABLE ADD PARTITION` を使用してオブジェクトキー名を指定します。詳細については、このドキュメントで後述する「Amazon S3 パスを変更するか再度定義する」を参照してください。
- クエリのタイムアウト – `MSCK REPAIR TABLE` は、初めてテーブルを作成する場合、またはデータとパーティションメタデータ間のパリティについて不確実性がある場合の使用に最適です。`MSCK REPAIR TABLE` を使用して新しいパーティションを頻繁に追加し (例えば、日常的に追加するなど)、クエリのタイムアウトが発生している場合は、ALTER TABLE ADD PARTITION の使用を検討してください。
- ファイルシステムからパーティションが欠落している – Amazon S3 でパーティションを手動で削除してから `MSCK REPAIR TABLE` を実行すると、「Partitions missing from filesystem」というエラーメッセージが表示される場合があります。これは、`MSCK REPAIR TABLE` がテーブルメタデータから古いパーティションを削除しないために発生します。削除済みのパーティションをテーブルメタデータから除外するには、代わりに ALTER TABLE DROP PARTITION を実行します。SHOW PARTITIONS も同様に、ファイルシステムのパーティションではなく、メタデータのパーティションだけをリストすることに注意してください。
- 「NullPointerException 名は null です」というエラー

  AWS Glue CreateTable API 操作や AWS CloudFormation `AWS::Glue::Table` テンプレートを使用して、`TableType` プロパティを指定せずに Athena で使用するテーブルを作成し、`SHOW CREATE TABLE` または `MSCK REPAIR TABLE` などの DDL クエリを実行すると、「失敗: NullPointerException 名は null です」というエラーメッセージを受け取る場合があります。

  このエラーを解決するには、TableInput `TableType` 属性の値を AWS Glue `CreateTable` API コール、または AWS CloudFormation テンプレートの一部として指定します。`TableType` に使用できる値には、`EXTERNAL_TABLE` や `VIRTUAL_VIEW` が含まれます。

  この要件は、AWS Glue `CreateTable` API 操作や `AWS::Glue::Table` テンプレートを使用してテーブルを作成する場合にだけ適用されます。DDL ステートメントや AWS Glue クローラを使用して Athena のテーブルを作成すると、`TableType` プロパティが自動的に定義されます。

以下のセクションで詳細を説明します。

IAM ポリシーで glue:BatchCreatePartition を許可する

MSCK REPAIR TABLE の実行に使用しているロールにアタッチされている IAM ポリシーを見直します。Athena で AWS Glue Data Catalog を使用する場合は、IAM ポリシーで glue:BatchCreatePartition アクションが許可されている必要があります。glue:BatchCreatePartition アクションを許可する IAM ポリシーの例については、「AWS 管理ポリシー: AmazonAthenaFullAccess」を参照してください。

Amazon S3 パスを変更するか再度定義する

Amazon S3 パスの 1 つ以上のオブジェクトキーが小文字ではなくキャメルケースになっている場合、MSCK REPAIR TABLE がパーティションを AWS Glue Data Catalog に追加しない場合があります。例えば、Amazon S3 パスにオブジェクトキー名 userId が含まれている場合、次のパーティションが AWS Glue Data Catalog に追加されない可能性があります。

```
s3://DOC-EXAMPLE-BUCKET/path/userId=1/

s3://DOC-EXAMPLE-BUCKET/path/userId=2/

s3://DOC-EXAMPLE-BUCKET/path/userId=3/
```

この問題を解決するには、次のいずれかを実行します。

- Amazon S3 オブジェクトキーを作成するときには、キャメルケースではなく小文字を使用してください。

  ```
  s3://DOC-EXAMPLE-BUCKET/path/userid=1/

  s3://DOC-EXAMPLE-BUCKET/path/userid=2/

  s3://DOC-EXAMPLE-BUCKET/path/userid=3/
  ```

- 次の例に示されているように、ALTER TABLE ADD PARTITION を使用して場所を再度定義します。

  ```
  ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION (userId=1)
  LOCATION 's3://DOC-EXAMPLE-BUCKET/path/userId=1/'
  PARTITION (userId=2)
  LOCATION 's3://DOC-EXAMPLE-BUCKET/path/userId=2/'
  PARTITION (userId=3)
  ```

```
LOCATION 's3://DOC-EXAMPLE-BUCKET/path/userId=3/'
```

Amazon S3 オブジェクトキー名には大文字を使用できますが、Amazon S3 バケットの名称は常に小文字にする必要があります。詳細については、「Amazon S3 ユーザーガイド」の「オブジェクトキー命名ガイドライン」と「バケット命名規則」を参照してください。

SHOW COLUMNS

単一の指定したテーブルまたはビューの列の名前のみを表示します。より詳細な情報を取得するには、代わりに AWS Glue Data Catalog をクエリします。詳細と例については、AWS Glue Data Catalog のクエリ トピックの次のセクションを参照してください。

- 列メタデータ (データ型など) を表示するには、「指定したテーブルまたはビューの列のリスト化または検索」を参照してください。
- AwsDataCatalog の特定のデータベース内の全テーブルですべての列を一覧表示するには、「指定したテーブルまたはビューの列のリスト化または検索」を参照してください。
- AwsDataCatalog の全データベースにある全テーブルの全列を表示するには、「全テーブルでのすべての列の一覧表示」を参照してください。
- データベース内の特定のテーブルに共通する列を表示するには、「特定のテーブルに共通する列を一覧表示する」 を参照してください。

概要

```
SHOW COLUMNS {FROM|IN} database_name.table_name
```

```
SHOW COLUMNS {FROM|IN} table_name [{FROM|IN} database_name]
```

FROM および IN キーワードは、同じように使用できます。*table_name* または *database_name* にハイフンのような特殊文字が含まれている場合は、名前をバッククォートで囲みます (`my-database`.`my-table` など)。*table_name*または*database_name*をシングルクォートまたはダブルクォートで囲んではいけません。現在、LIKE およびパターンマッチングの式の使用はサポートされていません。

例

次の例はどれも同等で、customers データベース内の orders テーブルの列を表示します。最初の 2 つの例では、customers が現在のデータベースとします。

```
SHOW COLUMNS FROM orders
```

```
SHOW COLUMNS IN orders
```

```
SHOW COLUMNS FROM customers.orders
```

```
SHOW COLUMNS IN customers.orders
```

```
SHOW COLUMNS FROM orders FROM customers
```

```
SHOW COLUMNS IN orders IN customers
```

SHOW CREATE TABLE

table_name という既存のテーブルを分析し、作成元のクエリを生成します。

概要

```
SHOW CREATE TABLE [db_name.]table_name
```

パラメータ

TABLE [db_name.]table_name

db_name パラメータはオプションです。省略すると、コンテキストはデフォルトで現在のデータベースになります。

> **Note**
>
> テーブル名は必須です。

例

```
SHOW CREATE TABLE orderclickstoday;
```

```
SHOW CREATE TABLE `salesdata.orderclickstoday`;
```

トラブルシューティング

AWS Glue CreateTable API 操作や AWS CloudFormation AWS::Glue::Table テンプレートを使用して、TableType プロパティを指定せずに Athena で使用するテーブルを作成し、SHOW CREATE TABLE または MSCK REPAIR TABLE などの DDL クエリを実行すると、「失敗: NullPointerException 名は null です」というエラーメッセージを受け取る場合があります。

このエラーを解決するには、TableInput TableType 属性の値を AWS Glue CreateTable API コール、または AWS CloudFormation テンプレートの一部として指定します。TableType に使用できる値には、EXTERNAL_TABLE や VIRTUAL_VIEW が含まれます。

この要件は、AWS Glue CreateTable API 操作や AWS::Glue::Table テンプレートを使用してテーブルを作成する場合にだけ適用されます。DDL ステートメントや AWS Glue クローラを使用して Athena のテーブルを作成すると、TableType プロパティが自動的に定義されます。

SHOW CREATE VIEW

指定するビューを作成する SQL ステートメントを表示します。

概要

```
SHOW CREATE VIEW view_name
```

例

```
SHOW CREATE VIEW orders_by_date
```

「CREATE VIEW」および「DROP VIEW」も参照してください。

SHOW DATABASES

メタストアに定義されているすべてのデータベースを一覧表示します。DATABASES または SCHEMAS を使用できます。どちらも同じ意味です。

SHOW DATABASES と同等の動作をプログラム的に行うのは、ListDatabases Athena API アクションです。AWS SDK for Python (Boto3) での同等のメソッドは、list_databases です。

概要

```
SHOW {DATABASES | SCHEMAS} [LIKE 'regular_expression']
```

パラメータ

[LIKE '*regular_expression*']

データベースのリストをフィルタ処理して、指定した *regular_expression* と一致するものに絞り込みます。ワイルドカード文字の照合では、組み合わせ .* を使用できます。この組み合わせは、任意の文字に無制限に一致します。

例

```
SHOW SCHEMAS;
```

```
SHOW DATABASES LIKE '.*analytics';
```

SHOW PARTITIONS

Athena テーブルのすべてのパーティションをソートせずに表示します。

概要

```
SHOW PARTITIONS table_name
```

- テーブル内のパーティションを表示し、特定の順序で一覧表示するには、AWS Glue Data Catalog のクエリ ページの「特定のテーブルのパーティションのリスト化」セクションを参照してください。
- パーティションの内容を表示するには、Athena でのデータのパーティション化 ページの「データのクエリ」セクションを参照してください。
- SHOW PARTITIONS は、Athena によって射影されていても、AWS Glue カタログに登録されていないパーティションをリストしません。パーティション射影の詳細については、「Amazon Athena でのパーティション射影」を参照してください。
- SHOW PARTITIONS はメタデータのパーティションをリストしますが、実際のファイルシステムにあるパーティションはリストしません。Amazon S3 でパーティションを手動で削除してからメタデータを更新するには、ALTER TABLE DROP PARTITION を実行してください。

例

次のクエリの例では、flight_delays_csv テーブルのパーティションが表示されます。このテーブルには、米国運輸省のフライトテーブルデータが表示されます。この flight_delays_csv テーブルの例の詳細については、「CSV、TSV、およびカスタム区切りファイルの LazySimpleSerDe」を参照してください。テーブルは、年ごとにパーティション化されます。

```
SHOW PARTITIONS flight_delays_csv
```

結果

```
year=2007
year=2015
year=1999
year=1993
year=1991
year=2003
year=1996
year=2014
year=2004
year=2011
...
```

次のクエリの例では、impressions テーブルのパーティションが表示されます。このテーブルには、ウェブ閲覧データのサンプルが含まれています。この impressions テーブルの例の詳細については、「Athena でのデータのパーティション化」を参照してください。テーブルは、dt (datetime) 列によってパーティション化されています。

```
SHOW PARTITIONS impressions
```

結果

```
dt=2009-04-12-16-00
dt=2009-04-13-18-15
dt=2009-04-14-00-20
dt=2009-04-12-13-00
dt=2009-04-13-02-15
dt=2009-04-14-12-05
dt=2009-04-14-06-15
dt=2009-04-12-21-15
```

```
dt=2009-04-13-22-15
...
```

パーティションをソートしてリストする

結果リストでパーティションを整列させるには、SHOW PARTITIONS の代わりに次の SELECT 構文を使用します。

```
SELECT * FROM database_name."table_name$partitions" ORDER BY column_name
```

次のクエリでは、flight_delays_csv の例のパーティションの一覧を表示しますが、ソートされます。

```
SELECT * FROM "flight_delays_csv$partitions" ORDER BY year
```

結果

```
year
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
...
```

詳細については、AWS Glue Data Catalog のクエリ ページの「特定のテーブルのパーティションのリスト化」セクションを参照してください。

SHOW TABLES

データベース内のすべての基本テーブルとビューを一覧表示します。

概要

```
SHOW TABLES [IN database_name] ['regular_expression']
```

パラメータ

[IN database_name]

テーブルを一覧表示する database_name を指定します。省略すると、現在のコンテキストのデータベースが使用されます。

> **Note**
>
> database_name がハイフンなどのサポートされていない文字を使用すると、SHOW TABLES は失敗する可能性があります。回避策として、データベース名をバッククォートで囲んでみてください。

['regular_expression']

テーブルのリストをフィルタして、指定した regular_expression と一致するものに絞り込みます。AWSDataCatalog テーブルの任意の文字を示すには、* または .* ワイルドカード式を使用できます。Apache Hive データベースの場合は、.* ワイルドカード式を使用します。文字間の選択を示すには、| 文字を使用します。

例

Example – データベース **sampledb** 内のすべてのテーブルを表示します

```
SHOW TABLES IN sampledb
```

Results

```
alb_logs
cloudfront_logs
elb_logs
flights_2016
flights_parquet
view_2016_flights_dfw
```

Example – **sampledb** 内にある「flights」という単語を含むすべてのテーブルの名前を表示します

```
SHOW TABLES IN sampledb '*flights*'
```

Results

```
flights_2016
flights_parquet
view_2016_flights_dfw
```

Example – **sampledb** 内にある「logs」という単語で終わるすべてのテーブルの名前を表示します

```
SHOW TABLES IN sampledb '*logs'
```

Results

```
alb_logs
cloudfront_logs
elb_logs
```

SHOW TBLPROPERTIES

指定されたテーブルのテーブルプロパティを一覧表示します。

概要

```
SHOW TBLPROPERTIES table_name [('property_name')]
```

パラメータ

[('property_name')]

このパラメータを含めると、property_name というプロパティの値のみが表示されます。

例

```
SHOW TBLPROPERTIES orders;
```

```
SHOW TBLPROPERTIES orders('comment');
```

SHOW VIEWS

指定するデータベース、あるいは、データベース名を省略する場合の現在のデータベースのビューのリスト。オプションの LIKE 句を正規表現で使用して、ビュー名のリストを制限します。

Athena は、各値がビューの名前である STRING 型の値のリストを返します。

概要

```
SHOW VIEWS [IN database_name] [LIKE 'regular_expression']
```

パラメータ

[IN database_name]

ビューを一覧表示する database_name を指定します。省略すると、現在のコンテキストのデータベースが使用されます。

[LIKE 'regular_expression']

ビューのリストをフィルタして、指定した regular_expression と一致するものに絞り込みます。使用できるのは、任意の文字を示すワイルドカードの *、または複数の文字からの選択を示す | のみです。

例

```
SHOW VIEWS;
```

```
SHOW VIEWS IN marketing_analytics LIKE 'orders*'
```

「SHOW COLUMNS」、「SHOW CREATE VIEW」、「DESCRIBE VIEW」、「DROP VIEW」も参照してください。

Amazon Athena での SQL クエリに関する考慮事項と制約事項

Athena でクエリを実行するときは、以下の考慮事項と制限事項に留意してください。

- ストアドプロシージャ – ストアドプロシージャはサポートされていません。

- パーティションの最大数 – CREATE TABLE AS SELECT (CTAS) ステートメントで作成できるパーティションの最大数は 100 です。詳細については、「CREATE TABLE AS」を参照してください。回避策については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。
- サポートされていないステートメント – 以下のステートメントはサポートされていません。
  - CREATE TABLE LIKE はサポートされていません。
  - DESCRIBE INPUT および DESCRIBE OUTPUT はサポートされていません。
  - MERGE ステートメントは、トランザクションテーブル形式でのみサポートされています。詳細については、「MERGE INTO」を参照してください。
  - UPDATE ステートメントはサポートされません。
- Trino および Presto コネクタ – Trino コネクタも Presto コネクタもサポートされていません。データソースの接続には Amazon Athena の横串検索を使用します。詳細については、「Amazon Athena 横串検索の使用」を参照してください。
- 多数のパーティションがあるテーブルでのタイムアウト – Athena は、何千ものパーティションがあるテーブルをクエリするときにタイムアウトする場合があります。これは、テーブルに string 型ではないパーティションが多数ある場合に発生する可能性があります。string 型を使用すると、Athena はメタストアレベルでパーティションをプルーニングします ただし、他のデータ型を使用する場合、Athena はサーバー側のパーティションをプルーニングします。パーティションの数が多いほど、このプロセスにかかる時間が長くなり、クエリがタイムアウトする可能性が高くなります。この問題を解決するには、パーティションの型を string に設定して、Athena がメタストアレベルでパーティションをプルーニングするようにします。これにより、オーバーヘッドが軽減され、クエリがタイムアウトするのを防ぐことができます。
- S3 Glacier のサポート — 復元された Amazon S3 Glacier オブジェクトをクエリするための詳細については、「復元された Amazon S3 Glacier オブジェクトへのクエリ」を参照してください。
- 非表示として扱われるファイル – Athena は、アンダースコア (_) またはドット (.) で始まるソースファイルを非表示として扱います。この制限を回避するには、ファイルの名前を変更します。
- 行または列サイズの制限 – 単一の行、またはその列を 32 MB を超えるサイズにすることはできません。この制限は、例えば CSV ファイルまたは JSON ファイルの行に 300 MB の単一の列が含まれる場合に超過する可能性があります。この制限を超えると、「Line too long in text file」というエラーメッセージが表示されることもあります。この制限を回避するには、行内の列の合計データサイズが 32 MB 未満であることを確認してください。
- LIMIT 句の最大値 — LIMIT 句に指定できる最大行数は

  9223372036854775807 です。ORDER BY を使用するとき、LIMIT 句でサポートされる行の最大数は 2147483647 です。この制限を超えると、NOT_SUPPORTED: ORDER BY LIMIT >

2147483647 is not supported (NOT_SUPPORTED: ORDER BY LIMIT > 2147483647 はサポートされていません) というエラーメッセージが表示されます。

- information_schema – `information_schema` のクエリのパフォーマンスは、AWS Glue メタデータの量が少ない、または中程度の場合に最も高くなります。大量のメタデータがある場合は、エラーが発生する可能性があります。AWS Glue メタデータのための `information_schema` データベースのクエリについては、「AWS Glue Data Catalog のクエリ」を参照してください。
- 配列初期化— Java の制限により、254 個を超える引数を持つ配列を Athena で初期化することはできません。
- 非表示メタデータ列 — Hive または Iceberg の非表示メタデータ列 `$bucket`、`$file_modified_time`、`$file_size`、および `$partition` は、ビューでサポートされていません。Athena の `$path`メタデータ列の使用方法については「Amazon S3 内にあるソースデータのファイルの場所の取得 」を参照してください。

クエリ文字列の最大長、クエリタイムアウトのクォータ、アクティブな DML クエリ数のクォータについては、「Service Quotas」を参照してください。

Athena のトラブルシューティング

Athena チームは、お客様の問題から以下のトラブルシューティング情報を収集しました。これは包括的なものではありませんが、一般的なパフォーマンス、タイムアウト、およびメモリ不足の問題に関するアドバイスが含まれています。

トピック

- クエリタイムアウトの問題
- スロットリングの問題
- ビュー
- ワークグループ
- 追加リソース
- Athena エラーカタログ

CREATE TABLE AS SELECT (CTAS)

同時 CTAS ステートメントで重複したデータが発生する

Athena は、CTAS の同時検証を維持しません。同じ場所に対して重複した CTAS ステートメントが同時に存在しないことを確認するようにしてください。CTAS または INSERT INTO ステートメントが失敗した場合でも、ステートメントで指定されたデータの場所に孤立したデータが残される可能性があります。

HIVE_TOO_MANY_OPEN_PARTITIONS

CTAS ステートメントを使用して 100 を超えるパーティションを持つテーブルを作成すると、HIVE_TOO_MANY_OPEN_PARTITIONS: Exceeded limit of 100 open writers for partitions/buckets (HIVE_TOO_MANY_OPEN_PARTITIONS: パーティション/バケットのオープンライターの制限である 100 を超えました) エラーメッセージが表示されることがあります。これらの制限を回避するには、CTAS ステートメントと、それぞれ最大 100 個のパーティションを作成または挿入する一連の `INSERT INTO` ステートメントを使用できます。詳細については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。

データファイルの問題

Athena が隠しファイルを読み取れない

Athena は、アンダースコア (_) またはドット (.) で始まるソースファイルを隠しファイルとして扱います。この制限を回避するには、ファイルの名前を変更します。

AWS Glue クローラから除外したファイルを Athena が読み取ってしまう

Athena は、AWS Glue クローラに指定した除外パターンを認識しません。例えば、`.csv` と `.json` ファイルの両方が含まれる Amazon S3 バケットがある場合、`.json` ファイルをクローラから除外し

ても、Athena は両方のファイルのグループをクエリします。これを回避するには、除外するファイルを別の場所に配置します。

HIVE_BAD_DATA: Error parsing field value

このエラーは、以下のシナリオで発生する可能性があります。

- テーブルで定義されているデータ型がソースデータと一致しない、または単一のフィールドに異なるデータ型が含まれている。解決案については、「AWS ナレッジセンター」の「My Amazon Athena query fails with the error "HIVE_BAD_DATA: Error parsing field value for field x: For input string: "12312845691""」(Amazon Athena のクエリが失敗してエラー「HIVE_BAD_DATA: Error parsing field value for field x: For input string: "12312845691"」が表示される) を参照してください。
- 整数フィールドに NULL 値が存在する。回避策の 1 つは、null 値を `string` とした列を作成してから、`CAST` を使用してクエリのフィールドを変換して、null にデフォルト値の `0` を提供することです。詳細については、「AWS ナレッジセンター」の「When I query CSV data in Athena, I get the error "HIVE_BAD_DATA: Error parsing field value " for field x: For input string: ""」(Athena で CSV データをクエリすると、エラー「HIVE_BAD_DATA: Error parsing field value " for field x: For input string: ""」が表示される) を参照してください。

HIVE_CANNOT_OPEN_SPLIT: Error opening Hive split s3://DOC-EXAMPLE-BUCKET

このエラーは、多数のオブジェクトを持つ Amazon S3 バケットプレフィックスをクエリするときに発生する可能性があります。詳細については、AWS ナレッジセンターで「Athena の『HIVE_CANNOT_OPEN_SPLIT: Error opening Hive split s3://awsdoc-example-bucket/: Slow Down (Service: Amazon S3; Status Code: 503; Error Code: 503 Slow Down;』というエラーの解決方法を教えてください」を参照してください。

HIVE_CURSOR_ERROR: com.amazonaws.services.s3.model.AmazonS3Exception: The specified key does not exist

このエラーは通常、クエリの実行中にファイルが削除されたときに発生します。クエリを再度実行する、またはワークフローをチェックして、クエリの実行中に別のジョブまたはプロセスがファイルを変更していないかどうかを確認してください。

HIVE_CURSOR_ERROR: Unexpected end of input stream

このメッセージは、ファイルが破損しているか、空であることを示します。ファイルの整合性を確認し、クエリを再度実行してください。

HIVE_FILESYSTEM_ERROR: Incorrect fileSize *1234567* for file

このメッセージは、クエリプランとクエリ実行の間でファイルが変更された場合に発生する可能性があります。通常、Amazon S3 上のファイルがインプレイスで置き換えられた場合に発生します (例えば、オブジェクトが既に存在するキーに対して PUT が実行された場合など)。Athena では、クエリ実行時のファイル内容の削除や置換はサポートされていません。このエラーを回避するには、クエリが実行されない場合にファイルを上書きまたは削除するジョブ、もしくは新しいファイルまたはパーティションにのみデータを書き込むジョブをスケジュールします。

HIVE_UNKNOWN_ERROR: Unable to create input format

このエラーは、以下のような問題が原因になっている可能性があります。

- AWS Glue クローラがデータ形式を分類できなかった
- 特定の AWS Glue テーブル定義プロパティが空になっている
- Athena が Amazon S3 にあるファイルのデータ形式をサポートしていない

詳細については、「AWS ナレッジセンター」の「Athena の『unable to create input format (入力形式を作成できません)』というエラーを解決する方法を教えてください」を参照、またはナレッジセンターの動画をご覧ください。

クエリ結果を保存するために提供された S3 の場所が無効になっている

クエリ結果のために有効な S3 の場所を指定していることを確認します。詳細については、「クエリ結果、最近のクエリ、および出力ファイルの使用」トピックの「クエリ結果の場所の指定」を参照してください。

Linux Foundation Delta Lake テーブル

Delta Lake テーブルのスキーマが同期されていません

AWS Glue に古いスキーマを含む Delta Lake テーブルをクエリするとき、次のエラーメッセージが表示されることがあります。

```
INVALID_GLUE_SCHEMA: Delta Lake table schema in Glue does not match the most recent
 schema of the
Delta Lake transaction log. Please ensure that you have the correct schema defined in
 Glue.
```

Athena に追加した後に AWS Glue でスキーマを変更した場合、最新の情報とならない可能性があります。スキーマを更新するには、次のいずれかのステップを実行します。

- AWS Glue で AWS Glue クローラーを実行します。
- Athena でテーブルをドロップして、もう一度作成します。
- Athena の ALTER TABLE ADD COLUMNS ステートメントを使用するか、AWS Glue でテーブルスキーマを編集し、欠落している列を手動で追加します。

横串検索

ListTableMetadata の呼び出し中にタイムアウトが発生した

ListTableMetadata API の呼び出しは、データソースに多くのテーブルが存在する場合、データソースが遅い場合、またはネットワークが遅い場合にタイムアウトすることがあります。この問題のトラブルシューティングを行うには、次のステップを使用します。

- テーブル数を確認 — テーブルが 1,000 を超える場合、テーブル数を減らしてみてください。最速の `ListTableMetadata` 応答を得るには、カタログあたりのテーブル数を 1000 未満にすることをお勧めします。
- Lambda 設定を確認する — Lambda 関数の動作をモニタリングすることは非常に重要です。フェデレーレーションカタログを使用するときは、Lambda 関数の実行ログを確認してください。結果に基づいて、メモリとタイムアウトの値を調整してください。タイムアウトに関する潜在的な問題を特定するには、Lambda 設定を再確認してください。詳細については、「AWS Lambda デベロッパーガイド」の「関数タイムアウトの設定 (コンソール)」を参照してください。
- フェデレーションデータソースのログを確認する — フェデレーションデータソースからのログとエラーメッセージを調べて、問題やエラーがないかどうかを確認します。ログからはタイムアウトの原因に関する貴重な情報を得ることができます。
- **`StartQueryExecution`** をメタデータの取得に使用 — テーブルが 1,000 を超える場合、フェデレーションコネクタを使用してメタデータを取得するのに予想以上に時間がかかることがあります。StartQueryExecution の非同期性により、Athena がクエリを最適な方法で実行できるため、`ListTableMetadata` の代替として `StartQueryExecution` を使用することを検討してください。次の AWS CLI の例は、`ListTableMetadata` の代わりに `StartQueryExecution` を使用してデータカタログにあるテーブルのメタデータをすべて取得する方法を示しています。

  まず、次の例のように、すべてのテーブルを取得するクエリを実行します。

```
aws athena start-query-execution --region us-east-1 \
```

```
--query-string "SELECT table_name FROM information_schema.tables LIMIT 50" \
--work-group "your-work-group-name"
```

その後、次の例のように、個別のテーブルのメタデータを取得します。

```
aws athena start-query-execution --region us-east-1 \
--query-string "SELECT * FROM information_schema.columns \
WHERE table_name = 'your-table-name' AND \
table_catalog = 'your-catalog-name'" \
--work-group "your-work-group-name"
```

結果を取得するまでにかかる時間は、カタログ内のテーブル数によって異なります。

フェデレーションクエリのトラブルシューティングについては、GitHub の「awslabs/aws-athena-query-federation」セクションで「Common_Problems」または個々の「Athena データソースコネクタ」のドキュメントを参照してください。

JSON 関連のエラー

JSON データを読み込もうとしたときの NULL または不正なデータエラー

JSON データを読み込もうとしたときの NULL または不正なデータエラーには、多くの原因が考えられます。OpenX SerDe の使用時にエラーを発生させている行を特定するには、`ignore.malformed.json` を `true` に設定します。不正な形式のレコードが NULL として返されます。詳細については、AWS ナレッジセンターの「Amazon Athena で JSON データを読み込もうとするとエラーが発生します」を参照、またはナレッジセンターの動画をご覧ください。

HIVE_BAD_DATA: Error parsing field value for field 0: java.lang.String cannot be cast to org.openx.data.jsonserde.json.JSONObject

OpenX JSON SerDe は、Athena クエリの列の解析に失敗した場合にこのエラーをスローします。これは、列を `map` または `struct` として定義したが、基盤となるデータが実際には `string`、`int`、またはその他のプリミティブ型であるという場合に発生する可能性があります。

HIVE_CURSOR_ERROR: Row is not a valid JSON object - JSONException: Duplicate key

このエラーは、Athena を使用して、大文字と小文字が異なった同じ名前のタグが複数ある AWS Config リソースをクエリする場合に発生します。解決策は、`WITH SERDEPROPERTIES`

'case.insensitive'='false' を使用して CREATE TABLE を実行し、名前をマッピングすることです。case.insensitive およびマッピングについては、「JSON SerDe ライブラリ」を参照してください。詳細については、「AWS ナレッジセンター」の「Athena で AWS Config からファイルを読み込むときの「HIVE_CURSOR_ERROR: Row is not a valid JSON Object - JSONException: Duplicate key」を解決する方法を教えてください。」を参照してください。

プリティプリントされた JSON に伴う HIVE_CURSOR_ERROR メッセージ

Hive JSON SerDe および OpenX JSON SerDe ライブラリでは、各 JSON ドキュメントが、レコード内のフィールドを区切る行終端文字がない 1 行のテキストに存在することが想定されます。JSON テキストがプリティプリント形式の場合、テーブルを作成した後にクエリを実行しようとすると、以下のようなエラーメッセージが表示される場合があります。「HIVE_CURSOR_ERROR: Row is not a valid JSON Object」、または「HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT」。詳細については、GitHub の OpenX SerDe のドキュメントで「JSON Data Files」(JSON データファイル) を参照してください。

複数の JSON レコードが 1 の SELECT COUNT を返す

OpenX JSON SerDe を使用している場合は、レコードが改行文字で区切られていることを確認してください。詳細については、「AWS ナレッジセンター」の「入力 JSON ファイルに複数のレコードが含まれているにもかかわらず、Amazon Athena の SELECT COUNT クエリが 1 つのレコードのみを返す」を参照してください。

カスタム JSON 分類子を使用する AWS Glue クローラによって作成されたテーブルをクエリできない

Athena エンジンはカスタム JSON 分類子をサポートしません。この問題を回避するには、カスタム分類子なしで、新しいテーブルを作成します。JSON を変換するには、CTAS を使用するか、ビューを作成することができます。例えば、配列を使用している場合は、UNNEST オプションを使用して JSON をフラット化できます。もう 1 つのオプションは、カスタム分類子をサポートする AWS Glue ETL ジョブを使用し、Amazon S3 でデータを Parquet に変換してから、それを Athena でクエリすることです。

MSCK REPAIR TABLE

MSCK REPAIR TABLE 関連の問題については、「MSCK REPAIR TABLE」ページの「考慮事項と制約事項」および「トラブルシューティング」セクションを参照してください。

出力の問題

Unable to verify/create output bucket

このエラーは、指定されたクエリ結果の場所が存在しない、または適切な許可が存在しない場合に発生します。詳細については、「AWS ナレッジセンター」の「How do I resolve the "unable to verify/create output bucket" error in Amazon Athena?」(Amazon Athena の「unable to verify/create output bucket」エラーの解決方法) を参照してください。

TIMESTAMP 結果が空になる

Athena には Java TIMESTAMP 形式が必要です。詳細については、「AWS ナレッジセンター」の「Amazon Athena のテーブルにクエリを実行すると、TIMESTAMP の結果が空になる」を参照してください。

Athena クエリ出力を CSV 以外の形式で保存する

デフォルトでは、Athena は CSV 形式でのみファイルを出力します。異なる形式で SELECT クエリの結果を出力するには、UNLOAD ステートメントを使用できます。詳細については、「UNLOAD」を参照してください。CTAS クエリで `format` テーブルプロパティを使用して、出力形式を設定することもできます。UNLOAD とは異なり、CTAS ではテーブルの作成が必要です。詳細については、「AWS ナレッジセンター」の「Athena クエリ出力を、CSV 以外の形式 (圧縮形式など) で保存する方法を教えてください」を参照してください。

The S3 location provided to save your query results is invalid

このエラーメッセージは、出力バケットの場所が、クエリを実行したリージョンと同じリージョンにない場合に表示される可能性があります。これを回避するには、クエリを実行するリージョンにあるクエリ結果の場所を指定します。この手順については、「クエリ結果の場所の指定」を参照してください。

Parquet の問題

org.apache.parquet.io.GroupColumnIO cannot be cast to org.apache.parquet.io.PrimitiveColumnIO

このエラーは、Parquet スキーマの不一致が原因で発生します。AWS Glue で、非プリミティブ型 (`array` など) を持つ列がプリミティブ型 (`string` など) として宣言されています。この問題をトラ

ブルシューティングするには、ファイルのデータスキーマをチェックして、それを AWS Glue で宣言されたスキーマと比較してください。

Parquet 統計の問題

Parquet データを読み取ると、次のようなエラーメッセージが表示されることがあります。

```
HIVE_CANNOT_OPEN_SPLIT: Index x out of bounds for length y
HIVE_CURSOR_ERROR: Failed to read x bytes
HIVE_CURSOR_ERROR: FailureException at Malformed input: offset=x
HIVE_CURSOR_ERROR: FailureException at java.io.IOException:
can not read class org.apache.parquet.format.PageHeader: Socket is closed by peer.
```

この問題を回避するには、次の例のように CREATE TABLE ステートメントまたは ALTER TABLE SET TBLPROPERTIES ステートメントを使用して Parquet SerDe `parquet.ignore.statistics` プロパティを `true` に設定します。

CREATE TABLE の例

```
...
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES ('parquet.ignore.statistics'='true')
STORED AS PARQUET
...
```

ALTER TABLE の例

```
ALTER TABLE ... SET TBLPROPERTIES ('parquet.ignore.statistics'='true')
```

Parquet Hive については、「Parquet SerDe」を参照してください。

パーティションの問題

MSCK REPAIR TABLE が古いパーティションを削除しない

Amazon S3 でパーティションを手動で削除してから MSCK REPAIR TABLE を実行すると、「Partitions missing from filesystem」というエラーメッセージが表示される場合があります。これは、MSCK REPAIR TABLE がテーブルメタデータから古いパーティションを削除しないために発生

します。ALTER TABLE DROP PARTITION を使用して、古いパーティションを手動で削除してください。詳細については、MSCK REPAIR TABLE トピックの「トラブルシューティング」セクションを参照してください。

MSCK REPAIR TABLE の失敗

大量のパーティション (100,000 を超えるなど) が特定のテーブルに関連付けられている場合、MSCK REPAIR TABLE がメモリ制限のために失敗する場合があります。この制限を回避するには、その代わりに ALTER TABLE ADD PARTITION を使用します。

MSCK REPAIR TABLE がパーティションを検出しても、それらを AWS Glue に追加しない

この問題は、Amazon S3 パスが小文字ではなくキャメルケースになっている、または IAM ポリシーが glue:BatchCreatePartition アクションを許可しない場合に発生する可能性があります。詳細については、「AWS ナレッジセンター」の「MSCK REPAIR TABLE は Athena のパーティションを検出しますが、それらを AWS Glue Data Catalog に追加しません」を参照してください。

dd-MM-yyyy-HH-mm-ss または yyyy-MM-dd の日付形式でのパーティション射影範囲が機能しない

正しく動作させるには、日付形式を yyyy-MM-dd HH:00:00 にする必要があります。詳細については、Stack Overflow 記事の「Athena Partition Projection Not Working As Expected」を参照してください。

PARTITION BY が BIGINT 型をサポートしない

データ型を string に変換してから再試行してください。

意味のあるパーティションがない

このエラーメッセージは通常、パーティション設定が破損していることを意味します。この問題を解決するには、テーブルをドロップして、新しいパーティションでテーブルを作成します。

パーティション射影は、範囲パーティションと連動しません。

時間範囲単位の projection.*<columnName>*.interval.unit が、パーティションの区切り文字と一致することを確認してください。例えば、パーティションが日で区切られている場合、時間の範囲単位は機能しません。

範囲をハイフンで指定するとパーティション投影エラーが発生します

`range` テーブルプロパティをカンマの代わりにハイフンで指定すると、「INVALID_TABLE_PROPERTY: For input string: "*number-number*"」のようなエラーが表示されます。範囲の値は、ハイフンではなくカンマで区切ってください。詳細については、「整数型」を参照してください。

HIVE_UNKNOWN_ERROR: Unable to create input format

各 Glue パーティションが独自の固有な入力形式を単独で持っているため、1 つまたは複数の Glue パーティションが異なる形式で宣言されています。パーティションが AWS Glue でどのように定義されているか確認してください。

HIVE_PARTITION_SCHEMA_MISMATCH

パーティションのスキーマがテーブルのスキーマと異なる場合、クエリが「HIVE_PARTITION_SCHEMA_MISMATCH」というエラーメッセージで失敗する場合があります。詳細については、「"HIVE_PARTITION_SCHEMA_MISMATCH" を避けるためのパーティションスキーマの同期」を参照してください。

SemanticException table is not partitioned but partition spec exists

このエラーは、`CREATE TABLE` ステートメントにパーティションが定義されていない場合に発生する可能性があります。詳細については、「AWS ナレッジセンター」の「Athenaで『FAILED: SemanticException table is not partitioned but partition spec exists (SemanticException テーブルにはパーティション仕様が存在しますが、パーティションされていません)』というエラーのトラブルシューティング方法を教えてください」を参照してください。

ゼロバイト **_$folder$** ファイル

`ALTER TABLE ADD PARTITION` ステートメントを実行し、すでに存在するパーティションと正しくない Amazon S3 の場所を誤って指定すると、形式 *partition_value*`_$folder$` のゼロバイトプレースホルダーが Amazon S3 に作成されます。これらのファイルは手動で削除する必要があります。

これが起こらないようにするには、次のように、`ALTER TABLE ADD PARTITION` ステートメントで `ADD IF NOT EXISTS` 構文を使用します。

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITIION […]
```

パーティションされたデータからレコードが返されない

この問題はさまざまな理由で発生する可能性があります。考えられる原因と解決策については、AWS ナレッジセンターの「パーティションを定義してAmazon Athenaにテーブルを作成しましたが、このテーブルに対してクエリを実行するときにゼロ個のレコードが返されるのはなぜですか?」を参照してください。

「HIVE_TOO_MANY_OPEN_PARTITIONS」も参照してください。

アクセス許可

Amazon S3 のクエリ時におけるアクセス拒否エラー

これは、バケット内のデータを読み込む許可がない、結果バケットに書き込む許可がない、または Amazon S3 パスに `us-east-1.amazonaws.com` のようなリージョンエンドポイントが含まれる場合に発生する可能性があります。詳細については、「AWS ナレッジセンター」の「When I run an Athena query, I get an "access denied" error」(Athena クエリを実行すると「access denied」エラーになる) を参照してください。

Amazon S3 にある暗号化されたデータでの DDL クエリの実行時に、ステータスコード: 403 でアクセスが拒否される

「Access Denied (Service: Amazon S3; Status Code: 403; Error Code: AccessDenied; Request ID: *<request_id>*)」というエラーメッセージは、以下の状況が当てはまる場合に表示されることがあります。

1. `ALTER TABLE ADD PARTITION` または `MSCK REPAIR TABLE` のような DDL クエリを実行した。
2. デフォルト暗号化が `SSE-S3` を使用するように設定されているバケットがある。
3. また、そのバケットに、`PutObject` リクエストが PUT ヘッダー `"s3:x-amz-server-side-encryption": "true"` と `"s3:x-amz-server-side-encryption": "AES256"` を指定することを強制する、以下のようなバケットポリシーがある。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
```

```
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<resource-name>/*",
            "Condition": {
                "Null": {
                    "s3:x-amz-server-side-encryption": "true"
                }
            }
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<resource-name>/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "AES256"
                }
            }
        }
    ]
}
```

このような場合に推奨される解決策は、バケットのデフォルト暗号化がすでに存在することを前提として、上記のようなバケットポリシーを削除することです。

別のアカウントにある Amazon S3 バケットのクエリ時に、ステータスコード: 403 でアクセスが拒否される

このエラーは、別の AWS のサービス によって書き込まれたログをクエリしようとしており、2 番目のアカウントがバケット所有者であるがバケット内のオブジェクトを所有していない場合に発生する可能性があります。詳細については、「AWS ナレッジセンター」の「I get the Amazon S3 exception "access denied with status code: 403" in Amazon Athena when I query a bucket in another account」(別のアカウントでバケットをクエリすると、Amazon Athena で Amazon S3 例外「access denied with status code: 403」が発生する) を参照、またはナレッジセンターの動画をご覧ください。

IAM ロールの認証情報を使用して Athena JDBC ドライバーに接続する

ロールの一時的な認証情報を取得して、Athena への JDBC 接続を認証できます。一時的な認証情報の存続期間は最大 12 時間です。詳細については、「AWS ナレッジセンター」の「JDBC ドラ

イバーを使用して Athena に接続するときに、独自の IAM ロール認証情報を使用する、または別の IAM ロールに切り替える方法を教えてください」を参照してください。

クエリ構文の問題

失敗: NullPointerException 名は null です

AWS Glue CreateTable API 操作や AWS CloudFormation `AWS::Glue::Table` テンプレートを使用して、`TableType` プロパティを指定せずに Athena で使用するテーブルを作成し、`SHOW CREATE TABLE` または `MSCK REPAIR TABLE` などの DDL クエリを実行すると、「失敗: NullPointerException 名は null です」というエラーメッセージを受け取る場合があります。

このエラーを解決するには、TableInput `TableType` 属性の値を AWS Glue `CreateTable` API コール、または AWS CloudFormation テンプレートの一部として指定します。`TableType` に使用できる値には、`EXTERNAL_TABLE` や `VIRTUAL_VIEW` が含まれます。

この要件は、AWS Glue `CreateTable` API 操作や `AWS::Glue::Table` テンプレートを使用してテーブルを作成する場合にだけ適用されます。DDL ステートメントや AWS Glue クローラを使用して Athena のテーブルを作成すると、`TableType` プロパティが自動的に定義されます。

Function not Registered

このエラーは、Athena がサポートしていない関数を使用しようとするときに発生します。Athena がサポートする関数のリストについては、「Amazon Athena の関数」を参照する、またはクエリエディタで `SHOW FUNCTIONS` ステートメントを実行してください。独自のユーザー定義関数 (UDF) を作成することもできます。詳細については、「AWS ナレッジセンター」の「Athena の『function not registered (関数が登録されていません)』 構文エラーを解決する方法を教えてください」を参照してください。

GENERIC_INTERNAL_ERROR の例外

`GENERIC_INTERNAL_ERROR` の例外には、次のようなさまざまな原因が考えられます。

- GENERIC_INTERNAL_ERROR: Null — この例外は、次のいずれかの条件で表示される場合があります。
  - テーブルで定義された列のデータ型と実際のデータセットで使用されるデータ型との間に、スキーマの不一致がある。
  - 不正確な構文で `CREATE TABLE AS SELECT` (CTAS) クエリを実行している。

- GENERIC_INTERNAL_ERROR: parent builder is null — この例外は、データ型 `array` の列を持つテーブルをクエリし、OpenCSVSerDe ライブラリを使用している場合に表示されることがあります。OpenCSVSerde 形式では、`array` データ型はサポートされません。
- GENERIC_INTERNAL_ERROR: Value exceeds MAX_INT — この例外は、ソースデータ列がデータ型 `INT` で定義されており、値が 2,147,483,647 より大きい場合に表示される場合があります。
- GENERIC_INTERNAL_ERROR: Value exceeds MAX_BYTE — この例外は、ソースデータ列の値がデータ型 `BYTE` の許容サイズを超える場合に表示される場合があります。データ型 `BYTE` は `TINYINT` に相当します。`TINYINT` は、2 の補数形式の 8 ビット符号付き整数で、最小値は -128、最大値は 127 です。
- GENERIC_INTERNAL_ERROR: Number of partition values does not match number of filters — Amazon Simple Storage Service (Amazon S3) データのパーティションに矛盾がある場合、この例外が表示される場合があります。次のいずれかの条件の下では、パーティションに矛盾がある可能性があります。
  - Amazon S3 のパーティションが変更された (例: 新しいパーティションが追加された)。
  - テーブル内のパーティション列の数が、パーティションメタデータの列の数と一致しない。

これらの各エラーの詳細については、「AWS ナレッジセンター」の「Amazon Athenaのテーブルをクエリする際に発生する、『GENERIC_INTERNAL_ERROR』エラーを解決する方法を教えてください。」を参照してください。

Number of matching groups doesn't match the number of columns

このエラーは、CREATE TABLE ステートメントで Regex SerDe を使用しており、regex マッチンググループの数が、テーブルに指定した列の数と一致しない場合に発生します。詳細については、「AWS ナレッジセンター」の「How do I resolve the RegexSerDe error "number of matching groups doesn't match the number of columns" in amazon Athena?」(Amazon Athena の RegexSerDe エラー「number of matching groups doesn't match the number of columns」の解決方法) を参照してください。

queryString failed to satisfy constraint: Member must have length less than or equal to 262144

Athena でのクエリ文字列の最大長 (262,144 バイト) は、調整可能なクォータではありません。AWS Support はクォータを増やすことはできませんが、長いクエリを小さなクエリに分割することで問題を回避できます。詳細については、「AWS ナレッジセンター」の「Athena でクエリ文字列の最大長を増やすにはどうすればよいですか?」を参照してください。

SYNTAX_ERROR: Column cannot be resolved

バイトオーダーマーク (BOM) を持つ UTF-8 でエンコードされた CSV ファイルから AWS Glue クローラによって作成されたテーブルをクエリすると、このエラーが発生することがあります。AWS Glue は BOM を認識せず、疑問符に変更します。疑問符は Amazon Athena では認識されません。解決策は、Athena または AWS Glue で疑問符を削除することです。

関数呼び出しの引数が多すぎます

Athena エンジンバージョン 3 では、関数に含まれる引数は 127 個を超えて取ることができません。この制限は、デザインによるものです。127 個を超えるパラメーターを持つ関数を使用すると、次のようなエラーメッセージが表示されます。

TOO_MANY_ARGUMENTS: 行 *nnn*:*nn*: 関数呼び出し *function_name*() の引数が多すぎます。

この問題を解決するには、関数呼び出しごとに使用するパラメータを少なくします。

クエリタイムアウトの問題

Athena クエリでタイムアウトエラーが発生した場合、CloudTrail ログを確認してください。AWS Glue または Lake Formation API のスロットリングが原因で、クエリがタイムアウトすることがあります。このようなエラーが発生すると、対応するエラーメッセージは、スロットリングの問題ではなく、クエリのタイムアウトを示している可能性があります。問題のトラブルシューティングを行うには、AWS Support に連絡する前に CloudTrail のログを確認してください。詳細については、AWS CloudTrail ログのクエリおよびAWS CloudTrail を使用した Amazon Athena API コールのログ記録を参照してください。

`ListTableMetadata` API を呼び出す際のフェデレーションクエリのタイムアウトの問題については、「ListTableMetadata の呼び出し中にタイムアウトが発生した」を参照してください。

スロットリングの問題

クエリが Amazon S3、AWS KMS、AWS Glue、AWS Lambda など、依存しているサービスの制限を超える場合は、以下のメッセージが発生すると予想できます。これらの問題を解決するには、同じアカウントから発信される同時コールの数を減らします。

サービス	エラーメッセージ
AWS Glue	AWSGlueException: Rate exceeded.

サービス	エラーメッセージ
AWS KMS	You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
AWS Lambda	Rate exceeded TooManyRequestsException
Amazon S3	AmazonS3Exception: Please reduce your request rate.

Athena を使用するときに Amazon S3 のスロットリングを防ぐ方法については、「Amazon S3 スロットリングの防止」を参照してください。

ビュー

Apache Hive シェルで作成されたビューが Athena で機能しない

これらの実装は根本的に異なるため、Apache Hive シェルで作成されたビューには Athena との互換性がありません。この問題を解決するには、Athena でビューを再度作成します。

View is stale; it must be re-created

このエラーは、ビューの基盤であるテーブルが変更またはドロップされた場合に表示される可能性があります。解決策は、ビューを再度作成することです。詳細については、「AWS ナレッジセンター」の「How can I resolve the "view is stale; it must be re-created" error in Athena?」(Athena の「view is stale; it must be re-created」エラーの解決方法) を参照してください。

ワークグループ

ワークグループの問題のトラブルシューティングについては、「ワークグループのトラブルシューティング」(ワークグループのトラブルシューティング) を参照してください。

追加リソース

以下のページには、Amazon Athena での問題のトラブルシューティングに関する追加の情報が記載されています。

- Athena エラーカタログ
- Service Quotas
- Amazon Athena での SQL クエリに関する考慮事項と制約事項
- サポートされない DDL
- テーブル、データベース、および列の名前
- Amazon Athena のデータ型
- サポートされる SerDes とデータ形式
- Athena での圧縮のサポート
- 予約済みキーワード
- ワークグループのトラブルシューティング

以下の AWS リソースも役に立ちます。

- AWS ナレッジセンターの Athena に関するトピック
- AWS re:Post の Amazon Athena に関する質問
- AWS ビッグデータブログ の Athena に関する記事

トラブルシューティングには、エキスパート、またはヘルパーコミュニティによる反復的な照会と検出が必要になることがよくあります。このページにある提案事項を試しても問題が引き続き発生する場合は、AWS Support に問い合わせるか (AWS Management Console で [Support] (サポート)、[Support Center] (サポートセンター) の順に選択します)、[Amazon Athena] タグを使用して AWS re:Post で質問してください。

Athena エラーカタログ

Athena では、失敗したクエリを理解し、クエリエラーが発生した後の手順を実行するために役立つ標準化されたエラー情報を提供しています。`AthenaError` 機能には `ErrorCategory` フィールドと `ErrorType` フィールドが含まれます。`ErrorCategory` は失敗したクエリの原因がシステムエラー、ユーザーエラー、またはその他のエラーのいずれによるものかを記述しています。`ErrorType` は、障害の原因に関する詳細な情報を提供します。2 つのフィールドを組み合わせることで、発生した特定のエラーの状況と原因をよりよく理解することができます。

エラーカテゴリ

Athena エラーのカテゴリの値とその意味を次の表に示します。

エラーカテゴリ	ソース
1	SYSTEM
2	USER
3	OTHER

エラータイプ参照

Athena エラータイプの値とその意味を次の表に示します。

エラータイプ	説明
0	このスケール係数でクエリリソースが枯渇しました
1	このスケール係数でクエリリソースが枯渇しました
2	このスケール係数でクエリリソースが枯渇しました
3	このスケール係数でクエリリソースが枯渇しました
4	このスケール係数でクエリリソースが枯渇しました
5	このスケール係数でクエリリソースが枯渇しました
6	このスケール係数でクエリリソースが枯渇しました
7	このスケール係数でクエリリソースが枯渇しました
8	このスケール係数でクエリリソースが枯渇しました
100	内部サービスエラー
200	クエリエンジンで内部エラーが発生しました
201	クエリエンジンで内部エラーが発生しました
202	クエリエンジンで内部エラーが発生しました

エラータイプ	説明
203	ドライバーエラー
204	メタストアにエラーがありました
205	クエリエンジンで内部エラーが発生しました
2.0.6	クエリがタイムアウトしました
207	クエリエンジンで内部エラーが発生しました
208	クエリエンジンで内部エラーが発生しました
209	クエリのキャンセルに失敗しました
210	クエリがタイムアウトしました
211	クエリエンジンで内部エラーが発生しました
212	クエリエンジンで内部エラーが発生しました
213	クエリエンジンで内部エラーが発生しました
214	クエリエンジンで内部エラーが発生しました
215	クエリエンジンで内部エラーが発生しました
216	クエリエンジンで内部エラーが発生しました
217	クエリエンジンで内部エラーが発生しました
218	クエリエンジンで内部エラーが発生しました
219	クエリエンジンで内部エラーが発生しました
220	クエリエンジンで内部エラーが発生しました
221	クエリエンジンで内部エラーが発生しました
222	クエリエンジンで内部エラーが発生しました

エラータイプ	説明
223	クエリエンジンで内部エラーが発生しました
224	クエリエンジンで内部エラーが発生しました
225	クエリエンジンで内部エラーが発生しました
226	クエリエンジンで内部エラーが発生しました
227	クエリエンジンで内部エラーが発生しました
228	クエリエンジンで内部エラーが発生しました
229	クエリエンジンで内部エラーが発生しました
230	クエリエンジンで内部エラーが発生しました
231	クエリエンジンで内部エラーが発生しました
232	クエリエンジンで内部エラーが発生しました
233	Iceberg エラー
234	Lake Formation エラー
235	クエリエンジンで内部エラーが発生しました
236	クエリエンジンで内部エラーが発生しました
237	シリアル化エラー
238	Simple Storage Service (Amazon S3) へのメタデータのアップロードに失敗しました
239	一般的な永続エラー
240	クエリの送信に失敗しました
300	内部サービスエラー
301	内部サービスエラー

エラータイプ	説明
302	内部サービスエラー
303	内部サービスエラー
400	内部サービスエラー
401	Simple Storage Service (Amazon S3) へのクエリ結果の書き込みに失敗しました
402	Simple Storage Service (Amazon S3) へのクエリ結果の書き込みに失敗しました
1000	ユーザーエラー
1001	データエラー
1002	データエラー
1003	DDL タスクが失敗しました
1004	スキーマエラー
1005	シリアル化エラー
1006	構文エラー
1007	データエラー
1008	クエリが拒否されました
1009	クエリに失敗しました
1010	内部サービスエラー
1011	ユーザーによってキャンセルされたクエリ
1012	クエリエンジンで内部エラーが発生しました
1013	クエリエンジンで内部エラーが発生しました
1014	ユーザーによってキャンセルされたクエリ

エラータイプ	説明
1100	無効な引数が指定されました
1101	無効なプロパティが指定されました
1102	クエリエンジンで内部エラーが発生しました
1103	無効なテーブルプロパティが指定されました
1104	クエリエンジンで内部エラーが発生しました
1105	クエリエンジンで内部エラーが発生しました
1106	無効な関数引数が指定されました
1107	無効なビュー
1108	関数の登録に失敗しました
1109	提供された Simple Storage Service (Amazon S3) パスが見つかりません
1110	指定されたテーブルまたはビューは存在しません
1200	クエリはサポートされていません
1201	提供されているデコーダはサポートされていません
1202	クエリタイプがサポートされていません
1300	一般的な見つからないエラー
1301	一般的なエンティティが見つかりません
1302	ファイルが見つかりません
1303	提供された関数または関数の実装が見つかりません
1304	クエリエンジンで内部エラーが発生しました
1305	クエリエンジンで内部エラーが発生しました

エラータイプ	説明
1306	Simple Storage Service (Amazon S3) バケットが見つかりません
1307	選択したエンジンが見つかりません
1308	クエリエンジンで内部エラーが発生しました
1400	スロットリングエラー
1401	AWS Glue スロットリングによりクエリが失敗しました
1402	AWS Glue のテーブルバージョンが多すぎるためクエリが失敗しました
1403	Simple Storage Service (Amazon S3) スロットリングによりクエリが失敗しました
1404	Amazon Athena スロットリングによりクエリが失敗しました
1405	Amazon Athena スロットリングによりクエリが失敗しました
1406	Amazon Athena スロットリングによりクエリが失敗しました
1500	アクセス許可エラー
1501	Simple Storage Service (Amazon S3) アクセス許可エラー
1602	リザーブドキャパシティの制限を超えました。このクエリを実行するには容量が不足しています。
1700	Lake Formation の内部例外が原因でクエリが失敗しました
1701	AWS Glue の内部例外が原因でクエリが失敗しました
9999	内部サービスエラー

コードサンプル

このトピックの例は、Athena アプリケーションを記述するための開始点として SDK for Java 2.x を使用しています。

> **Note**
>
> 言語固有の AWS SDK を使用した Athena のプログラミングについては、次の関連リソースを参照してください。
>
> - AWS Command Line Interface (`athena`)
> - AWS SDK for .NET (`Amazon.Athena.Model`)
> - AWS SDK for C++ (`Aws::Athena::AthenaClient`)
> - AWS SDK for Go (`athena`)
> - AWS SDK for JavaScript v3 (`AthenaClient`)
> - AWS SDK for PHP 3.x (`Aws\Athena`)
> - AWS SDK for Python (Boto3) (`Athena.Client`)
> - AWS SDK for Ruby v3 (`Aws::Athena::Client`)

このセクションの Java コード例の実行についての詳細は、GitHub の AWS Code Examples Repository で「Amazon Athena Java Readme」を参照してください。Athena の Java プログラミングリファレンスについては、「AWS SDK for Java 2.x」の「AthenaClient」を参照してください。

- Java コードの例
  - 定数
  - Athena にアクセスするためのクライアントの作成
  - クエリ実行の使用
    - クエリ実行の開始
    - クエリ実行の停止
    - クエリ実行の表示
  - 名前付きクエリの使用
    - 名前付きクエリの作成
    - 名前付きクエリの削除
    - クエリ実行の表示

> **Note**
>
> これらのサンプルで使用する定数 (ATHENA_SAMPLE_QUERY など) は、ExampleConstants.java クラス宣言に定義されている文字列用です。これらの定数は、独自の文字列または定義済み定数を置き換えてください。

定数

ExampleConstants.java クラスは、Athena で「開始」チュートリアルに従って作成されたテーブルをクエリする方法を示しています。

```
package aws.example.athena;

public class ExampleConstants {

    public static final int CLIENT_EXECUTION_TIMEOUT = 100000;
    public static final String ATHENA_OUTPUT_BUCKET = "s3://bucketscott2"; // change
 the Amazon S3 bucket name to match
                                                                          // your
 environment
    // Demonstrates how to query a table with a comma-separated value (CSV) table.
    // For information, see
    // https://docs.aws.amazon.com/athena/latest/ug/work-with-data.html
    public static final String ATHENA_SAMPLE_QUERY = "SELECT * FROM scott2;"; // change
 the Query statement to match
                                                                            // your
 environment
    public static final long SLEEP_AMOUNT_IN_MS = 1000;
    public static final String ATHENA_DEFAULT_DATABASE = "mydatabase"; // change the
 database to match your database

}
```

Athena にアクセスするためのクライアントの作成

AthenaClientFactory.java クラスは、Amazon Athena クライアントを作成して設定する方法を示しています。

```
package aws.example.athena;
```

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.AthenaClientBuilder;

public class AthenaClientFactory {
    private final AthenaClientBuilder builder = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create());

    public AthenaClient createClient() {
        return builder.build();
    }
}
```

クエリ実行の開始

StartQueryExample は、Athena にクエリを送信し、結果が利用可能になるまで待機してから、結果を処理する方法を示しています。

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.QueryExecutionContext;
import software.amazon.awssdk.services.athena.model.ResultConfiguration;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.QueryExecutionState;
import software.amazon.awssdk.services.athena.model.GetQueryResultsRequest;
import software.amazon.awssdk.services.athena.model.GetQueryResultsResponse;
import software.amazon.awssdk.services.athena.model.ColumnInfo;
import software.amazon.awssdk.services.athena.model.Row;
import software.amazon.awssdk.services.athena.model.Datum;
import software.amazon.awssdk.services.athena.paginators.GetQueryResultsIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class StartQueryExample {

    public static void main(String[] args) throws InterruptedException {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String queryExecutionId = submitAthenaQuery(athenaClient);
        waitForQueryToComplete(athenaClient, queryExecutionId);
        processResultRows(athenaClient, queryExecutionId);
        athenaClient.close();
    }

    // Submits a sample query to Amazon Athena and returns the execution ID of the
    // query.
    public static String submitAthenaQuery(AthenaClient athenaClient) {
        try {
            // The QueryExecutionContext allows us to set the database.
            QueryExecutionContext queryExecutionContext =
 QueryExecutionContext.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .build();

            // The result configuration specifies where the results of the query should
 go.
            ResultConfiguration resultConfiguration = ResultConfiguration.builder()
                    .outputLocation(ExampleConstants.ATHENA_OUTPUT_BUCKET)
                    .build();

            StartQueryExecutionRequest startQueryExecutionRequest =
 StartQueryExecutionRequest.builder()
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .queryExecutionContext(queryExecutionContext)
                    .resultConfiguration(resultConfiguration)
                    .build();

            StartQueryExecutionResponse startQueryExecutionResponse = athenaClient
                    .startQueryExecution(startQueryExecutionRequest);
            return startQueryExecutionResponse.queryExecutionId();
```

```
        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return "";
    }

    // Wait for an Amazon Athena query to complete, fail or to be cancelled.
    public static void waitForQueryToComplete(AthenaClient athenaClient, String
queryExecutionId)
            throws InterruptedException {
        GetQueryExecutionRequest getQueryExecutionRequest =
GetQueryExecutionRequest.builder()
                .queryExecutionId(queryExecutionId)
                .build();

        GetQueryExecutionResponse getQueryExecutionResponse;
        boolean isQueryStillRunning = true;
        while (isQueryStillRunning) {
            getQueryExecutionResponse =
athenaClient.getQueryExecution(getQueryExecutionRequest);
            String queryState =
getQueryExecutionResponse.queryExecution().status().state().toString();
            if (queryState.equals(QueryExecutionState.FAILED.toString())) {
                throw new RuntimeException(
                        "The Amazon Athena query failed to run with error message: " +
getQueryExecutionResponse
                                .queryExecution().status().stateChangeReason());
            } else if (queryState.equals(QueryExecutionState.CANCELLED.toString())) {
                throw new RuntimeException("The Amazon Athena query was cancelled.");
            } else if (queryState.equals(QueryExecutionState.SUCCEEDED.toString())) {
                isQueryStillRunning = false;
            } else {
                // Sleep an amount of time before retrying again.
                Thread.sleep(ExampleConstants.SLEEP_AMOUNT_IN_MS);
            }
            System.out.println("The current status is: " + queryState);
        }
    }

    // This code retrieves the results of a query
    public static void processResultRows(AthenaClient athenaClient, String
queryExecutionId) {
```

```
        try {
            // Max Results can be set but if its not set,
            // it will choose the maximum page size.
            GetQueryResultsRequest getQueryResultsRequest =
 GetQueryResultsRequest.builder()
                    .queryExecutionId(queryExecutionId)
                    .build();

            GetQueryResultsIterable getQueryResultsResults = athenaClient
                    .getQueryResultsPaginator(getQueryResultsRequest);
            for (GetQueryResultsResponse result : getQueryResultsResults) {
                List<ColumnInfo> columnInfoList =
 result.resultSet().resultSetMetadata().columnInfo();
                List<Row> results = result.resultSet().rows();
                processRow(results, columnInfoList);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static void processRow(List<Row> row, List<ColumnInfo> columnInfoList) {
        for (Row myRow : row) {
            List<Datum> allData = myRow.data();
            for (Datum data : allData) {
                System.out.println("The value of the column is " +
 data.varCharValue());
            }
        }
    }
}
```

クエリ実行の停止

StopQueryExecutionExample はサンプルクエリを実行し、すぐにクエリを停止してから、クエリのステータスをチェックしてクエリがキャンセルされたことを確認します。

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
```

```
import software.amazon.awssdk.services.athena.model.StopQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.QueryExecutionState;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.QueryExecutionContext;
import software.amazon.awssdk.services.athena.model.ResultConfiguration;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class StopQueryExecutionExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String sampleQueryExecutionId = submitAthenaQuery(athenaClient);
        stopAthenaQuery(athenaClient, sampleQueryExecutionId);
        athenaClient.close();
    }

    public static void stopAthenaQuery(AthenaClient athenaClient, String
 sampleQueryExecutionId) {
        try {
            StopQueryExecutionRequest stopQueryExecutionRequest =
 StopQueryExecutionRequest.builder()
                    .queryExecutionId(sampleQueryExecutionId)
                    .build();

            athenaClient.stopQueryExecution(stopQueryExecutionRequest);
            GetQueryExecutionRequest getQueryExecutionRequest =
 GetQueryExecutionRequest.builder()
                    .queryExecutionId(sampleQueryExecutionId)
                    .build();

            GetQueryExecutionResponse getQueryExecutionResponse = athenaClient
```

```
                .getQueryExecution(getQueryExecutionRequest);
        if (getQueryExecutionResponse.queryExecution()
                .status()
                .state()
                .equals(QueryExecutionState.CANCELLED)) {

            System.out.println("The Amazon Athena query has been cancelled!");
        }

    } catch (AthenaException e) {
        e.printStackTrace();
        System.exit(1);
    }
}

// Submits an example query and returns a query execution Id value
public static String submitAthenaQuery(AthenaClient athenaClient) {
    try {
        QueryExecutionContext queryExecutionContext =
QueryExecutionContext.builder()
                .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                .build();

        ResultConfiguration resultConfiguration = ResultConfiguration.builder()
                .outputLocation(ExampleConstants.ATHENA_OUTPUT_BUCKET)
                .build();

        StartQueryExecutionRequest startQueryExecutionRequest =
StartQueryExecutionRequest.builder()
                .queryExecutionContext(queryExecutionContext)
                .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                .resultConfiguration(resultConfiguration).build();

        StartQueryExecutionResponse startQueryExecutionResponse = athenaClient
                .startQueryExecution(startQueryExecutionRequest);
        return startQueryExecutionResponse.queryExecutionId();

    } catch (AthenaException e) {
        e.printStackTrace();
        System.exit(1);
    }
    return null;

}
```

```
}
```

クエリ実行の表示

ListQueryExecutionsExample は、クエリ実行 ID のリストを取得する方法を示します。

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.ListQueryExecutionsRequest;
import software.amazon.awssdk.services.athena.model.ListQueryExecutionsResponse;
import software.amazon.awssdk.services.athena.paginators.ListQueryExecutionsIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListQueryExecutionsExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueryIds(athenaClient);
        athenaClient.close();
    }

    public static void listQueryIds(AthenaClient athenaClient) {
        try {
            ListQueryExecutionsRequest listQueryExecutionsRequest =
 ListQueryExecutionsRequest.builder().build();
            ListQueryExecutionsIterable listQueryExecutionResponses = athenaClient
                    .listQueryExecutionsPaginator(listQueryExecutionsRequest);
            for (ListQueryExecutionsResponse listQueryExecutionResponse :
 listQueryExecutionResponses) {
```

```
                List<String> queryExecutionIds =
 listQueryExecutionResponse.queryExecutionIds();
                System.out.println("\n" + queryExecutionIds);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

名前付きクエリの作成

CreateNamedQueryExample は、名前付きクエリの作成方法を示します。

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class CreateNamedQueryExample {
    public static void main(String[] args) {
        final String USAGE = """

                Usage:
                    <name>

                Where:
                    name - the name of the Amazon Athena query.\s
                """;
```

```
        if (args.length != 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String name = args[0];
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        createNamedQuery(athenaClient, name);
        athenaClient.close();
    }

    public static void createNamedQuery(AthenaClient athenaClient, String name) {
        try {
            // Create the named query request.
            CreateNamedQueryRequest createNamedQueryRequest =
 CreateNamedQueryRequest.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .description("Sample Description")
                    .name(name)
                    .build();

            athenaClient.createNamedQuery(createNamedQueryRequest);
            System.out.println("Done");

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

名前付きクエリの削除

DeleteNamedQueryExample は、名前付きクエリ ID を使用して名前付きクエリを削除する方法を示します。

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.DeleteNamedQueryRequest;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryRequest;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class DeleteNamedQueryExample {
    public static void main(String[] args) {
        final String USAGE = """

                Usage:
                    <name>

                Where:
                    name - the name of the Amazon Athena query.\s
                """;

        if (args.length != 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String name = args[0];
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String sampleNamedQueryId = getNamedQueryId(athenaClient, name);
        deleteQueryName(athenaClient, sampleNamedQueryId);
        athenaClient.close();
    }

    public static void deleteQueryName(AthenaClient athenaClient, String
 sampleNamedQueryId) {
        try {
```

```
            DeleteNamedQueryRequest deleteNamedQueryRequest =
 DeleteNamedQueryRequest.builder()
                    .namedQueryId(sampleNamedQueryId)
                    .build();

            athenaClient.deleteNamedQuery(deleteNamedQueryRequest);

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static String getNamedQueryId(AthenaClient athenaClient, String name) {
        try {
            CreateNamedQueryRequest createNamedQueryRequest =
 CreateNamedQueryRequest.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .name(name)
                    .description("Sample description")
                    .build();

            CreateNamedQueryResponse createNamedQueryResponse =
 athenaClient.createNamedQuery(createNamedQueryRequest);
            return createNamedQueryResponse.namedQueryId();

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return null;
    }
}
```

名前付きクエリの表示

ListNamedQueryExample は、名前付きクエリ ID のリストを取得する方法を示します。

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
```

```
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.ListNamedQueriesRequest;
import software.amazon.awssdk.services.athena.model.ListNamedQueriesResponse;
import software.amazon.awssdk.services.athena.paginators.ListNamedQueriesIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListNamedQueryExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listNamedQueries(athenaClient);
        athenaClient.close();
    }

    public static void listNamedQueries(AthenaClient athenaClient) {
        try {
            ListNamedQueriesRequest listNamedQueriesRequest =
 ListNamedQueriesRequest.builder()
                    .build();

            ListNamedQueriesIterable listNamedQueriesResponses = athenaClient
                    .listNamedQueriesPaginator(listNamedQueriesRequest);
            for (ListNamedQueriesResponse listNamedQueriesResponse :
 listNamedQueriesResponses) {
                List<String> namedQueryIds = listNamedQueriesResponse.namedQueryIds();
                System.out.println(namedQueryIds);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
```

```
}
```

Amazon Athena での Apache Spark の使用

Amazon Athena では、リソースの計画、設定、管理を必要とせずに、Apache Spark を使用してインタラクティブに簡単にデータ分析と探索を実行できます。Athena で Apache Spark アプリケーションを実行するということは、処理用 Spark コードを送信し、追加の設定をしなくても結果を直接受け取ることを意味します。Amazon Athena コンソールのシンプルなノートブックエクスペリエンスを使用すると、Python または Athena ノートブック API を使用して Apache Spark アプリケーションを開発できます。Amazon Athena の Apache Spark はサーバーレスであり、オンデマンドで自動的にスケーリングされるため、データ量や処理要件の変化に合わせて瞬時に処理できます。

Amazon Athena には次の特徴があります。

- コンソールの使用 - Amazon Athena コンソールから Spark アプリケーションを送信します。
- スクリプティング - Python で Apache Spark アプリケーションを迅速かつインタラクティブにビルドおよびデバッグできます。
- 動的スケーリング - Amazon Athena は、ジョブの実行に必要なコンピューティングリソースとメモリリソースを自動的に決定し、それに応じてそれらのリソースを指定された最大値まで継続的にスケーリングします。この動的スケーリングは、速度に影響を与えずにコストを削減します。
- ノートブックエクスペリエンス - Athena ノートブックエディタを使用すると、使い慣れたインターフェイスを使用して計算を作成、編集、実行できます。Athena ノートブックは Jupyter Notebook と互換性があり、計算として順番に実行されるセルのリストが含まれています。セルの内容には、コード、テキスト、マークダウン、数学、プロット、リッチメディアなどを含めることができます。

追加情報については、AWS Big Data Blog の「Explore your data lake using Amazon Athena for Apache Spark」を参照してください。

考慮事項と制約事項

- 現在、Amazon Athena for Apache Spark は以下の AWS リージョン でご利用いただけます。
  - アジアパシフィック (ムンバイ)
  - 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)
  - アジアパシフィック (東京)

  - 欧州 (フランクフルト)
  - 欧州 (アイルランド)
  - 米国東部 (バージニア北部)
  - 米国東部 (オハイオ)
  - 米国西部 (オレゴン)
- AWS Lake Formation はサポートされていません。
- パーティション射影を使用するテーブルはサポートされていません。
- Apache Spark 対応のワークグループでは Athena ノートブックエディターを使用できますが、Athena クエリエディターを使用することはできません。Athena SQL ワークグループのみが、Athena クエリエディターを使用できます。
- クロスエンジンビューのクエリはサポートされていません。Athena SQL で作成されたビューでは、Athena for Spark がクエリを実行することはできません。2 つのエンジンのビューは実装方法が異なるため、エンジン間の使用に互換性がありません。
- MLlib (Apache Spark 機械学習ライブラリ) および `pyspark.ml` パッケージはサポートされていません。サポートされている Python ライブラリのリストについては、「プリインストールされている Python ライブラリのリスト」を参照してください。
- 現在、Athena for Spark セッションで `pip install` はサポートされていません。
- 1 つのノートブックで許可されるアクティブなセッションは 1 つのみです。
- 複数のユーザーがコンソールを使用してワークグループ内の既存のセッションを開く場合、同じノートブックにアクセスします。混同を避けるため、自分で作成したセッションのみを開いてください。
- Amazon Athena で使用する Apache Spark アプリケーションのホスティングドメイン (例:`analytics-gateway.us-east-1.amazonaws.com`) は、インターネットの「パブリックサフィックスリスト (PSL)」に登録されています。ドメインに機密性の高いクッキーを設定する必要がある場合、ドメインをクロスサイトリクエストフォージェリ (CSRF) 攻撃から保護できるように、`__Host-` プレフィックスを付けたクッキーを使用することをお勧めします。詳細については、Mozilla.org のデベロッパー向けドキュメントの「クッキーの設定」ページを参照してください。
- Athena での Spark ノートブック、セッション、ワークグループのトラブルシューティングについては、「Athena for Spark のトラブルシューティング」を参照してください。

Amazon Athena で Apache Spark を開始

Amazon Athena で Apache Spark の使用を開始するには、最初に Spark が有効になっているワークグループを作成する必要があります。ワークグループに切り替えた後、ノートブックを作成するか、既存のノートブックを開くことができます。Athena でノートブックを開くと、そのノートブックに対して新しいセッションが自動的に開始され、Athena ノートブックエディタで直接操作できます。

> **Note**
>
> ノートブックを作成する前に、必ず Spark 対応のワークグループを作成してください。

Athena での Spark 対応ワークグループの作成

Athena のワークグループを使用して、ユーザー、チーム、アプリケーション、またはワークロードをグループ化し、コストを追跡できます。Amazon Athena で Apache Spark を使用するには、Spark エンジンを使用する Amazon Athena ワークグループを作成します。

> **Note**
>
> Apache Spark 対応のワークグループでは Athena ノートブックエディターを使用できますが、Athena クエリエディターを使用することはできません。Athena SQL ワークグループのみが、Athena クエリエディターを使用できます。

Athena で Spark 対応のワークグループを作成するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. ナビゲーションペインで、[Global networks] (グローバルネットワーク) を選択します。
4. [Workgroups] (ワークグループ) ページで、[Create workgroup] (ワークグループを作成する) をクリックします。
5. [Workgroup name] (ワークグループ名) には、Apache Spark ワークグループの名前を入力します。
6. (オプション) [Description] (説明) で、ワークグループの説明を入力します。
7. [Analytics engine] (アナリティクスエンジン) には、[Apache Spark] を選択してください。

> **Note**
>
> ワークグループを作成した後は、ワークグループの分析エンジンのタイプを変更することはできません。たとえば、Athena エンジンのバージョン 3 のワークグループを PySpark エンジンのバージョン 3 のワークグループに変更することはできません。

8. このチュートリアルでは、[Turn on example notebook] (ノートブックのサンプルを有効にする) を選択します。このオプション機能では、`example-notebook-random_string` という名前のノートブックのサンプルをワークグループに追加し、ノートブックがアカウント内の特定のデータベースやテーブルを作成、表示、削除するために使用する AWS Glue に関連するアクセス許可と、データセットのサンプルの Amazon S3 での読み込みアクセス許可を追加します。追加されたアクセス許可を確認するには、[View additional permissions details] (追加のアクセス許可の詳細を表示) を選択します。

> **Note**
>
> ノートブックのサンプルを実行すると、追加コストが発生する可能性があります。

9. [Additional configurations] (追加設定) では、以下の操作を行います。

   - [Use defaults] (デフォルトを使用) 設定を使用します。このオプションはデフォルトであり、Spark 対応のワークグループを使い始めるのに役立ちます。このオプションを使用すると、Athena は Amazon S3 に IAM ロールと計算結果のロケーションをユーザーに代わって作成します。IAM ロールの名前と作成する S3 バケットの場所は、[Additional configurations] (追加設定) の見出しの下にあるボックスに表示されます。
   - [Use defaults] (デフォルトを使用) 設定を無効にしてから、独自のワークグループ設定の指定 セクションの手順に進んでワークグループを手動で設定してください。

10. (オプション) [Tags] (タグ) - このオプションを使用して、ワークグループにタグを追加します。詳細については、「Athena リソースへのタグ付け」を参照してください。

11. [Create workgroup] (ワークグループの作成) を選択します。ワークグループが正常に作成されたことを知らせるメッセージが表示され、そのワークグループはワークグループのリストに表示されます。

独自のワークグループ設定の指定

ノートブック独自の IAM ロールと計算結果の場所を指定する場合は、このセクションの手順に従います。[Additional configurations] (その他の設定) オプションに [Use defaults] (デフォルトを使用) を選択した場合は、このセクションを飛ばして直接 ノートブックエクスプローラーを開いてワークグループを切り替える に進んでください。

以下の手順は、前のセクションの [To create a Spark enabled workgroup in Athena] (Athena で Spark 対応のワークグループを作成するには) の手順 1 ~ 9 を完了していることを前提としています。

独自のワークグループ設定を指定するには

1. 独自の IAM ロールを作成または使用したり、ノートブックの暗号化を設定したりする場合は、[IAM role configuration] (IAM ロール設定) を拡張してください。

   - [Service role] (サービスロール) で、以下のいずれかのオプションを選択します。

- サービスロールの作成 - このオプションを選択すると、Athena にサービスロールが自動的に作成されます。ロールを付与するアクセス許可を確認するには、[View permission details] (アクセス許可の詳細を表示) を選択します。
- [Choose an existing service role] (既存のサービスロールを選択) - ドロップダウンリストから既存のロールを選択します。選択するロールには、最初のオプションでアクセス許可を含める必要があります。ノートブック対応ワークグループのアクセス許可の詳細については、「Spark 対応ワークグループのトラブルシューティング」を参照してください。

- [Notebook and calculation code encryption key management] (ノートブックと計算コードの暗号化キーを管理) には、次のいずれかのオプションを選択します。
  - Amazon Athena による所有 - AWS KMS キーは Amazon Athena が所有および管理しています。このキーは追加料金なしで使用されます。
  - アカウントに保存され、ユーザーが所有および管理する対称キー - このオプションでは、次のいずれかを実行します。
    - 既存のキーを使用するには、検索ボックスを使用して AWS KMS を選択するか、キーの ARN を入力します。
    - AWS KMS コンソールでキーを作成するには、[AWS KMS キーを作成する] を選択します。実行ロールには、作成したキーを使用するアクセス許可が必要です。

> **Important**
>
> ワークグループの AWS KMS key を変更しても、更新前に管理されていたノートブックは引き続き古い KMS キーを参照します。更新後に管理されるノートブックでは、新しい KMS キーを使用します。以前のノートブックを新しい KMS キーを参照するように更新するには、以前のノートブックをそれぞれエクスポートしてからインポートします。以前のノートブックの参照を新しい KMS キーに更新する前に以前の KMS キーを削除すると、古いノートブックは復号化できなくなり、復元できなくなります。
>
> この動作は、KMS キーのわかりやすい名前であるエイリアスの更新にも当てはまります。KMS キーエイリアスを新しい KMS キーを指すように更新すると、エイリアスの更新前に管理されていたノートブックでは引き続き古い KMS キーを参照し、エイリアスの更新後に管理されたノートブックでは新しい KMS キーを使用します。KMS キーまたはエイリアスを更新する前に、これらの点を考慮してください。

2. 独自の計算結果の設定を指定する場合は、[Calculation result settings] (計算結果の設定) を展開し、次のオプションから選択します。

- 新しい S3 バケットを作成 - このオプションでは、計算の結果用 Amazon S3 バケットがアカウントに作成されます。バケット名は形式 *account_id*-*region*-athena-results-bucket-*alphanumeric_id* で、ACL を無効にする、パブリックアクセスをブロックする、バージョニングを無効にする、バケット所有者を強制するなどの設定を使用しています。
- 既存の S3 の場所を選択 - このオプションでは、次の操作を行います。
  - 検索ボックスに既存の場所への S3 パスを入力するか、[Browse S3] (S3 の参照) を選択してリストからバケットを選択します。

    > **Note**
    >
    > Amazon S3 の既存の場所を選択するときは、その場所にスラッシュ (/) を追加しないでください。これを行うと、計算の詳細ページの計算結果の場所へのリンクが間違ったディレクトリを指すようになります。このような場合、ワークグループの結果の場所を編集して末尾のスラッシュを削除してください。

  - (オプション) [View] (表示) を選択して Amazon S3 コンソールの [Buckets] (バケット) ページを開き、選択した既存のバケットに関する詳細情報を表示できます。
  - (オプション) [Expected bucket owner] (想定されるバケット所有者) に、クエリ結果の出力場所バケットの所有者になると想定されている AWS の ID を入力します。可能な限り、追加のセキュリティ対策としてこのオプションを選択することをお勧めします。バケット所有者のアカウント ID が指定した ID と一致しない場合、バケットに出力できません。詳細については、「Amazon S3 ユーザーガイド」の「バケット所有者条件によるバケット所有者の確認」を参照してください。
  - (オプション) 計算結果の場所が別のアカウントによって所有されており、クエリ結果に対する完全な制御を他のアカウントに許可したい場合、[Assign bucket owner full control over query results] (クエリ結果を完全に制御できるバケット所有者を割り当てる) を選択します。

3. (オプション) [Encrypt calculation results] (計算結果を暗号化) を選択し、次のいずれかを選択します。

   - SSE_S3 - これは SS3 で管理されるサーバー側の暗号化キーです。
   - SSE_KMS - ユーザーが提供するキー。[AWS KMS キーを選択] では、次のいずれかを選択できます。
     - 所有している AWS キーを使用 - ユーザーに代わって AWS が所有し管理するキーを使用します。

- 異なる AWS KMS キーを選択 (高度) - キーを選択または作成します。
  - 既存のキーを使用するには、検索ボックスを使用して AWS KMS を選択するか、キーの ARN を入力します。
  - KMS コンソールでキーを作成するには、[AWS KMS キーを作成する] を選択します。KMS コンソールでキーを作成し終えたら、Athena コンソールの [ワークグループを作成] ページに戻り、[AWS KMS キーを選択するか、ARN を入力] 検索ボックスを使用して、今作成したキーを選択します。

4. (オプション) [Other settings] (その他の設定) - このオプションを拡張して、ワークグループの [Publish CloudWatch metrics] (CloudWatch メトリクスの公開) オプションを有効または無効にします。このフィールドは、デフォルトで選択されています。詳細については、「CloudWatch メトリクスによる Apache Spark の計算のモニタリング」を参照してください。
5. (オプション) [Tags] (タグ) - このオプションを使用して、ワークグループにタグを追加します。詳細については、「Athena リソースへのタグ付け」を参照してください。
6. [Create workgroup] (ワークグループの作成) を選択します。ワークグループが正常に作成されたことを知らせるメッセージが表示され、そのワークグループはワークグループのリストに表示されます。

ノートブックエクスプローラーを開いてワークグループを切り替える

作成したばかりの Spark 対応ワークグループを使用する前に、そのワークグループに切り替える必要があります。Spark 対応のワークグループを切り替えるには、ノートブックエクスプローラー、またはノートブックエディタの [Workgroup] (ワークグループ) オプションを使用できます。

> **Note**
>
> 開始する前に、ブラウザがサードパーティの Cookie をブロックしていないことを確認してください。ブラウザのデフォルトまたはユーザーの設定でサードパーティのCookieをブロックしている場合、ノートブックは起動しません。Cookie の管理の詳細については、以下を参照してください。
>
> - Chrome
> - Firefox
> - Safari

ノートブックエクスプローラーを開いてワークグループを切り替える

1. ナビゲーションペインで、[Notebook Explorer] (ノートブックエクスプローラー) を選択します。
2. コンソールの右上にある [Workgroup] (ワークグループ) オプションを使用して、作成した Spark 対応ワークグループを選択します。ノートブックのサンプルがノートブックのリストに表示されます。

   ノートブックエクスプローラーは次の方法で使用できます。

   - ノートブックのリンクされた名前を選択して、そのノートブックを新しいセッションで開きます。
   - ノートブックの名前変更、削除、またはエクスポートを行うには、[Actions] (アクション) メニューを使用します。
   - ノートブックファイルをインポートするには、[Import file] (ファイルをインポート) を選択します。
   - ノートブックを作成するには、[Create notebook] (ノートブックの作成) を選択します。

ノートブックのサンプルの実行

ノートブックのサンプルは、一般公開されている二ューヨーク市のタクシー旅行データセットのデータをクエリします。ノートブックには、Spark DataFrames、Spark SQL、および AWS Glue Data Catalog の操作方法を示す例が示されています。

ノートブックのサンプルを実行するには

1. ノートブックエクスプローラーで、ノートブックのサンプルのリンクされた名前を選択します。

   これにより、デフォルトのパラメータでノートブックセッションが開始され、ノートブックエディタでノートブックが開きます。デフォルトのパラメータ (最大 20 DPU) を使用して新しい Apache Spark セッションが開始されたことを知らせるメッセージが表示されます。
2. セルを順番に実行して結果を確認するには、ノートブックのセルごとに [Run] (実行) ボタンを 1 回選択します。

   - 下へスクロールすると結果が表示され、新しいセルが表示されます。
   - 計算されたセルに対して、進行状況バーには完了率、経過時間、残り時間が表示されます。

- ノートブックのサンプルは、アカウントにデータベースのサンプルとテーブルを作成します。最後のセルでは、クリーンアップの手順としてこれらを削除します。

> **Note**
>
> ノートブックのサンプルのフォルダ、テーブル、またはデータベース名を変更する場合は、それらの変更が使用する IAM ロールに反映されていることを確認してください。そうしない場合、アクセス許可が不十分なため、ノートブックを実行できない可能性があります。

セッションの詳細を編集する

ノートブックセッションを開始後に、テーブルの形式、暗号化、セッションアイドルのタイムアウト、使用するデータ処理単位 (DPU) の最大同時数などのセッションの詳細を編集できます。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。

セッションの詳細を編集するには

1. ノートブックエディタで、右上の [Session] (セッション) メニューから [Edit session] (セッションの編集) を選択します。
2. [セッションの詳細の編集] ダイアログボックスにある [セッションパラメータ] セクションで、次のオプションの値を選択するか入力します。

   - その他のテーブル形式 — [Linux Foundation Delta Lake]、[Apache Hudi]、[Apache Iceberg]、または [カスタム] を選択します。
     - [Delta]、[Hudi]、または [Iceberg] のテーブルオプションの場合は、該当するテーブル形式に必要となるテーブルプロパティは [テーブルで編集] および [JSON で編集] オプションに自動的に提供されます。テーブル形式の使用に関する詳細な情報については、「Amazon Athena for Apache Spark で Hive 以外のテーブル形式を使用する」を参照してください。
     - [カスタム] またはその他のテーブルタイプのテーブルプロパティを追加または削除するには、[テーブルで編集] オプションと [JSON で編集] オプションを使用します。
     - [テーブルで編集] オプションを使用する場合は、[プロパティを追加] を選択してプロパティを追加するか、[削除] を選択してプロパティを削除します。[キー] ボックスと [値] ボックスを使用して、プロパティ名とその値を入力します。

- [JSON で編集] オプションを使用する場合は、JSON テキストエディタを使用して設定を直接編集します。
  - JSON テキストをクリップボードにコピーするには、[コピー] を選択します。
  - JSON エディタからすべてのテキストを削除するには、[クリア] を選択します。
  - 行折り返しの動作を設定するか、JSON エディタのカラーテーマを選択するには、設定 (歯車) アイコンを選択します。
- [Spark 暗号化を有効にする]- — ディスクに書き込まれ、Spark ネットワークノードを介して送信されるデータを暗号化する場合は、このオプションを選択します。詳細については、「Apache Spark 暗号化を有効にする」を参照してください。

3. [セッションパラメータ] セクションで、次のオプションの値を選択するか入力します。

   - [Session idle timeout] (セッションアイドルタイムアウト) - 1 ~ 480 分の値を選択または入力します。デフォルトは 20 です。
   - [Coordinator size] (コーディネーターサイズ) - コーディネーターは、ノートブックセッションで処理作業を調整し、他のエグゼキューターを管理する特別なエグゼキューターです。現在、1 つの DPU がデフォルトで、唯一の設定可能な値です。
   - [Executor size] (エグゼキューターサイズ) - エグゼキューターは、ノートブックセッションが Athena にリクエストできる最小の計算単位です。現在、1 つの DPU がデフォルトで、唯一の設定可能な値です。
   - [Max concurrent value] (同時実行の最大値) - 同時に実行できる DPU の最大数。デフォルトは 20 で、最小は 3、最大は 60 です。この値を増加しても自動的に追加のリソースが割り当てられるわけではありませんが、Athena では、計算負荷がそれを必要とし、リソースが使用可能になったときに、指定された最大数まで割り当てようとします。

4. [Save] を選択します。
5. [Confirm edit] (編集を確認) プロンプトで、[Confirm] (確認) を選択します。

   Athena はノートブックを保存し、指定したパラメータを使用して新しいセッションを開始します。ノートブックエディタのバナーに、変更されたパラメータで新しいセッションが開始されたことが通知されます。

> **Note**
>
> Athena はノートブックのセッション設定を記憶します。セッションのパラメータを編集してセッションを終了すると、Athena はノートブックのセッションを次に開始するときに設定したセッションパラメータを使用します。

セッションおよび計算の詳細の表示

ノートブックを実行すると、セッションおよび計算の詳細を表示できます。

セッションおよび計算の詳細を表示するには

1. 右上の [Session] (セッション) メニューから [View details] (詳細を表示) を選択します。
   - [Current session] (現在のセッション) タブには、セッション ID、作成時間、ステータス、ワークグループなど、現在のセッションに関する情報が表示されます。
   - [History] (履歴) タブには、以前のセッションのセッション ID が表示されます。以前のセッションの詳細を表示するには、[History] (履歴) タブを選択し、リストからセッション ID を選択します。
   - [Calculations] (計算) セクションには、セッションで実行された計算のリストが表示されます。
2. 計算の詳細を表示するには、計算 ID を選択します。
3. [Calculation details] (計算の詳細) ページで、以下の操作を実行できます。
   - 計算用のコードを確認するには、[Code] (コード) セクションを参照してください。
   - 計算の結果を確認するには、[Results] (結果) タブを選択します。
   - テキスト形式で表示された結果をダウンロードするには、[Download results] (結果をダウンロード) を選択します。
   - Amazon S3 の計算結果に関する情報を表示するには、[View in S3] (S3 で表示) を選択します。

セッションの終了

ノートブックセッションを終了するには

1. ノートブックエディタで、右上の [Session] (セッション) メニューから [Terminate] (終了) を選択します。
2. [Confirm session termination] (セッション終了の確認) プロンプトで、[Confirm] (確認) を選択します。ノートブックが保存され、ノートブックエディタに戻ります。

> **Note**
>
> ノートブックエディタのノートブックタブを閉じても、アクティブなノートブックのセッションは自動的には終了しません。セッションを確実に終了させたい場合は、[Session] (セッション)、[Terminate] (終了) オプションを使用してください。

独自のノートブックの作成

Spark 対応の Athena ワークグループを作成すると、独自のノートブックを作成できます。

ノートブックを作成するには

1. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
2. Athena コンソールのナビゲーションペインで、[Notebook explorer] (ノートブックエクスプローラー) または [Notebook editor] (ノートブックエディタ) を選択します。
3. 次のいずれかを行います。
   - [Notebook explorer] (ノートブックエクスプローラー) で、[Create notebook] (ノートブックの作成) を選択します。
   - [Notebook editor] (ノートブックエディタ) で、[Create notebook] (ノートブックの作成) を選択するか、プラスアイコン ([+]) を選択してノートブックを追加します。
4. [Create notebook] (ノートブックの作成) ダイアログボックスの [Notebook name] (ノートブック名) に名前を入力します。
5. (オプション) [Spark プロパティ] を展開し、次のオプションの値を選択するか入力します。
   - その他のテーブル形式 — [Linux Foundation Delta Lake]、[Apache Hudi]、[Apache Iceberg]、または [カスタム] を選択します。
     - [Delta]、[Hudi]、または [Iceberg] のテーブルオプションの場合は、該当するテーブル形式に必要となるテーブルプロパティは [テーブルで編集] および [JSON で編集] オプションに自動的に提供されます。テーブル形式の使用に関する詳細な情報については、「Amazon Athena for Apache Spark で Hive 以外のテーブル形式を使用する」を参照してください。
     - [カスタム] またはその他のテーブルタイプのテーブルプロパティを追加または削除するには、[テーブルで編集] オプションと [JSON で編集] オプションを使用します。

- [テーブルで編集] オプションを使用する場合は、[プロパティを追加] を選択してプロパティを追加するか、[削除] を選択してプロパティを削除します。[キー] ボックスと [値] ボックスを使用して、プロパティ名とその値を入力します。
- [JSON で編集] オプションを使用する場合は、JSON テキストエディタを使用して設定を直接編集します。
  - JSON テキストをクリップボードにコピーするには、[コピー] を選択します。
  - JSON エディタからすべてのテキストを削除するには、[クリア] を選択します。
  - 行折り返しの動作を設定するか、JSON エディタのカラーテーマを選択するには、設定 (歯車) アイコンを選択します。
- [Spark 暗号化を有効にする]- — ディスクに書き込まれ、Spark ネットワークノードを介して送信されるデータを暗号化する場合は、このオプションを選択します。詳細については、「Apache Spark 暗号化を有効にする」を参照してください。

6. (オプション) [Session parameters] (セッションパラメータ) を展開し、次のオプションの値を選択または入力します。

   - [Session idle timeout] (セッションアイドルタイムアウト) - 1 ~ 480 分の値を選択または入力します。デフォルトは 20 です。
   - [Coordinator size] (コーディネーターサイズ) - コーディネーターは、ノートブックセッションで処理作業を調整し、他のエグゼキューターを管理する特別なエグゼキューターです。現在、1 つの DPU がデフォルトで、唯一の設定可能な値です。DPU (Data Processing Unit) は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。
   - [Executor size] (エグゼキューターサイズ) - エグゼキューターは、ノートブックセッションが Athena にリクエストできる最小の計算単位です。現在、1 つの DPU がデフォルトで、唯一の設定可能な値です。
   - [Max concurrent value] (同時実行の最大値) - 同時に実行できる DPU の最大数。デフォルトは 20 で、最大は 60 です。この値を増加しても自動的に追加のリソースが割り当てられるわけではありませんが、Athena では、計算負荷がそれを必要とし、リソースが使用可能になったときに、指定された最大数まで割り当てようとします。

7. [Create] (作成) を選択します。ノートブックはノートブックエディタの新しいセッションで開きます。

以前に作成したノートブックを開く

以前に作成したノートブックを開くには

1. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
2. Athena コンソールのナビゲーションペインで、[Notebook editor] (ノートブックエディタ) または [Notebook explorer] (ノートブックエクスプローラー) を選択します。
3. 次のいずれかを行います。

   - [Notebook editor] (ノートブックエディタ) で、[Recent notebooks] (最近使用したノートブック) または [Saved notebooks] (保存したノートブック) リストからノートブックを選択します。新しいセッションでノートブックが開きます。
   - [Notebook explorer] (ノートブックエクスプローラー) で、リスト内のノートブックの名前を選択します。新しいセッションでノートブックが開きます。

ノートブックファイルの管理方法の詳細については、「ノートブックファイルの管理」を参照してください。

ノートブックの使用

Athena ノートブックエクスプローラーでノートブックを管理し、Athena ノートブックエディタを使用してセッションで編集および実行します。要件に応じて、ノートブックセッションの DPU 使用率を設定できます。

ノートブックを停止すると、関連するセッションが終了します。すべてのファイルは保存されますが、宣言された変数、関数、クラスで行われている変更は失われます。ノートブックを再起動すると、Athena がノートブックファイルをリロードし、コードを再実行できるようになります。

セッションと計算

各ノートブックは 1 つの Python カーネルに関連付けられており、Python コードを実行します。ノートブックは、コマンドを含むセルを 1 つ以上含めることができます。ノートブックでセルを実行するには、最初にノートブックのセッションを作成します。セッションは、ノートブックの変数と状態を追跡します。

ノートブックでセルを実行するということは、現在のセッションで計算を実行することを意味します。計算によってノートブックの状態が進行して、Amazon S3 からの読み込みや他のデータストア

への書き込みなどのタスクが実行される場合があります。セッションが実行されている限り、計算によってノートブックの状態は維持され、変更されます。

状態が不要になった場合、セッションを終了できます。セッションを終了してもノートブックは保持されますが、変数やその他の状態情報は破棄されます。同時に複数のプロジェクトに取り組む必要がある場合、プロジェクトごとにセッションを作成すると、セッションは互いに独立したものになります。

セッションには、DPU 単位で測定される専用の処理能力があります。セッションを作成する場合、そのセッションに複数の DPU を割り当てることができます。セッションが異なると、タスクの要件に応じて異なる容量を持つことができます。

Athena ノートブックエディタの使用

Athena ノートブックエディタは、コードを記述して実行するためのインタラクティブな環境です。以下のセクションでは、環境の機能について説明します。

コマンドモードと編集モードの違い

ノートブックエディタは、セルにテキストを入力するための編集モードと、コピー、貼り付け、実行などのコマンドをエディタ自体に発行するためのコマンドモードである、モーダルユーザーインターフェイスを備えています。

編集モードとコマンドモードを使用するには、次のタスクを実行できます。

- 編集モードに入るには、**ENTER** を押すか、セルを選択します。セルが編集モードの場合、セルの左側の余白は緑になります。
- コマンドモードに入るには、**ESC** を押すか、セルの外側をクリックします。コマンドは通常、現在選択されているセルのみに適用され、すべてのセルには適用されないことに注意してください。エディタがコマンドモードの場合、セルの左側の余白は青くなります。
- コマンドモードでは、キーボードショートカットとエディタの上にあるメニューを使用できますが、それぞれのセルにテキストを入力することはできません。
- セルを選択するには、セルを選択します。
- すべてのセルを選択するには、**Ctrl+A** (Windows) または **Cmd+A** (Mac) を押します。

ノートブックエディタメニュー

ノートブックエディタの上部にあるメニューのアイコンには、次のオプションがあります。

- 保存 - ノートブックの現在の状態を保存します。
- 下にセルを挿入 - 現在選択されているセルの下に新しい (空の) セルを追加します。
- 選択したセルを切り取る - 選択したセルを現在の場所から削除し、そのセルをメモリにコピーします。
- 選択したセルをコピー - 選択したセルをメモリにコピーします。
- セルを下に貼り付け - コピーしたセルを現在のセルの下に貼り付けます。
- 選択したセルを上に移動 - 現在のセルを上のセルの上に移動します。
- 選択したセルを下に移動 - 現在のセルを下のセルの下に移動します。
- 実行 - 現在の (選択済み) セルを実行します。出力は現在のセルのすぐ下に表示されます。
- すべて実行 - ノートブック内のすべてのセルを実行します。各セルの出力は、セルのすぐ下に表示されます。
- 停止 (カーネルを中断) - カーネルを中断して現在のノートブックを停止します。
- フォーマットオプション - 次のいずれかのセルのフォーマットを選択します。
  - コード - Python コードに使用します (デフォルト)。
  - マークダウン - GitHub スタイルのマークダウン形式でテキストを入力する場合に使用します。マークダウンをレンダリングするには、セルを実行します。
  - Raw NBConvert - 未修正の形式でテキストを入力する場合に使用します。[Raw NBConvert] (未加工の NBConvert) とマークされたセルは、Jupyter nbconvert コマンドラインツールで HTML などの別の形式に変換できます。
- 見出し - セルの見出しレベルを変更する場合に使用します。
- コマンドパレット - Jupyter Notebook コマンドとそのキーボードショートカットが含まれています。キーボードショートカットの詳細については、このドキュメント後半のセクションを参照してください。
- セッション - このメニューのオプションを使用して、セッションの詳細を表示したり、セッションパラメータを編集したり、セッションを終了したりします。

コマンドモードのキーボードショートカット

一般的なノートブックエディタコマンドモードのキーボードショートカットを次に示します。これらのショートカットは、**ESC** を押してからコマンドモードに入ると使用できます。エディタ後半のコマンドの一覧を表示するには、**ESC + H** を押します。

キー	アクション
1 - 6	セルタイプをマークダウンに変更し、見出しレベルを入力した数値に設定する
a	現在のセルの上にセルを作成する
b	現在のセルの下にセルを作成する
c	現在のセルをメモリにコピーする
d d	現在のセルを削除する
h	キーボードショートカットのヘルプ画面を表示する
j	1 つ下のセルに移動する
k	1 つ上のセルに移動する
m	現在のセル形式をマークダウンに変更する
r	現在のセル形式を raw に変更する
s	ノートブックを保存する
v	メモリのコンテンツを現在のセルの下に貼り付ける
x	選択した 1 つまたは複数のセルを切り取る
y	セルの形式をコードに変更する
z	[Undo] (元に戻す)
Ctrl+Enter	現在のセルを実行してコマンドモードに入る
Shift+Enter 、または **Alt+Enter**	現在のセルを実行して出力の下に新しいセルを作成し、編集モードで新しいセルを入力する
Space	ページダウン

キー	アクション
Shift+Space	ページアップ
Shift + L	セル内の行番号の表示/非表示を切り替える

コマンドモードのショートカットの編集

ノートブックエディタには、コマンドモードのキーボードショートカットをカスタマイズするオプションがあります。

コマンドモードのショートカットを編集するには

1. ノートブックエディタメニューから、[Command palette] (コマンドパレット) を選択します。
2. コマンドパレットから、[Edit command mode keyboard shortcuts] (コマンドモードのキーボードショートカット) コマンドを選択します。
3. [Edit command mode shortcuts] (編集コマンドモードのショートカット) インターフェイスを使用して、必要なコマンドをキーボードにマッピングまたは再マッピングします。

   コマンドモードのショートカットの編集方法を確認するには、[Edit command mode shortcuts] (コマンドモードのショートカットの編集) 画面の一番下までスクロールします。

Athena for Apache Spark のマジックコマンドを使用する方法については、「マジックコマンドを使用する」を参照してください。

マジックコマンドを使用する

マジックコマンド、またはマジックは、ノートブックのセルで実行できる特別なコマンドです。例えば、%env はノートブックセッションの環境変数を表示します。Athena は IPython 6.0.3 のマジック関数をサポートしています。

このセクションでは、Athena for Apache Spark 用の主要なマジックコマンドをいくつか取り上げて説明します。

- Athena のマジックコマンドのリストを表示するには、ノートブックのセルでコマンド **%lsmagic** を実行します。

- Athena ノートブックでマジックを使ってグラフを作成する方法については、データグラフを作成するためのマジック を参照してください。
- その他のマジックコマンドの詳細については、IPython ドキュメントの「組み込みマジックコマンド」を参照してください。

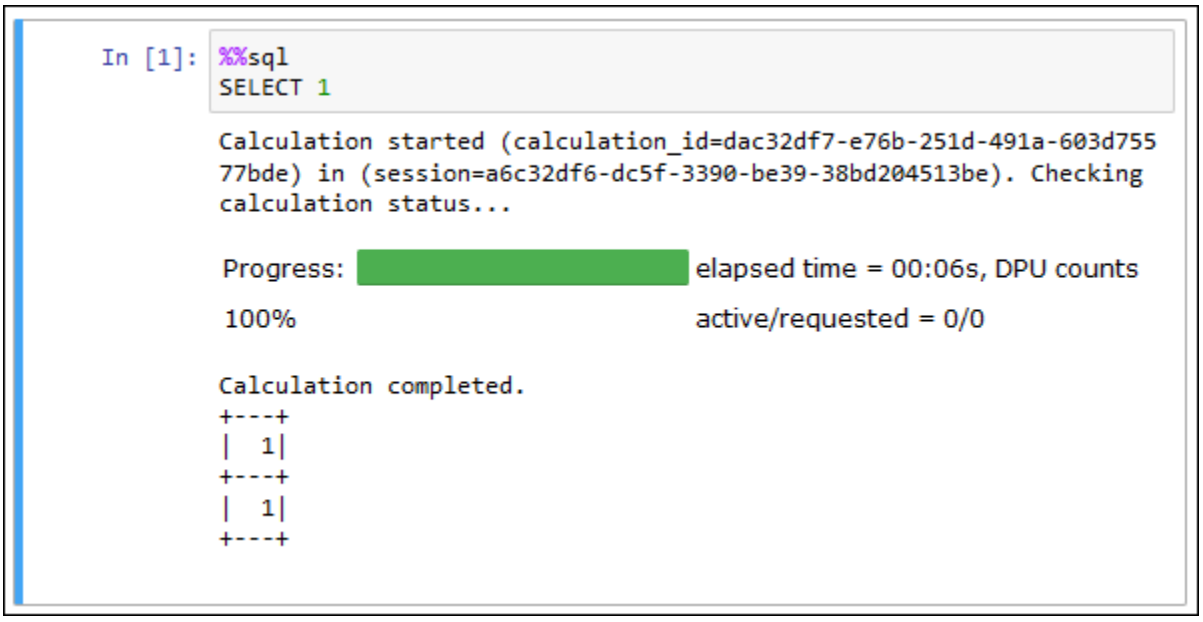
> **Note**
>
> 現在、`%pip` コマンドを実行すると失敗します。これは既知の問題です。

セルマジック

数行にまたがって書かれたマジックでは、その前に二重パーセント記号 (%%) が付き、セルマジック関数またはセルマジックと呼ばれます。

%%sql

このセルマジックを使用することで、Spark SQL ステートメントで修飾する必要なく、直接 SQL ステートメントを実行することができます。また、このコマンドは、返されたデータフレームで `.show()` を黙示的に呼び出すことで出力も表示します。



この `%%sql` コマンドは、自動的に列出力を 20 文字で切り捨てます。現在、この設定は構成されていません。この制限を回避するには、次の完全な構文を使用し、それに応じて show メソッドのパラメーターを変更してください。

```
spark.sql("""YOUR_SQL""").show(n=number, truncate=number, vertical=bool)
```

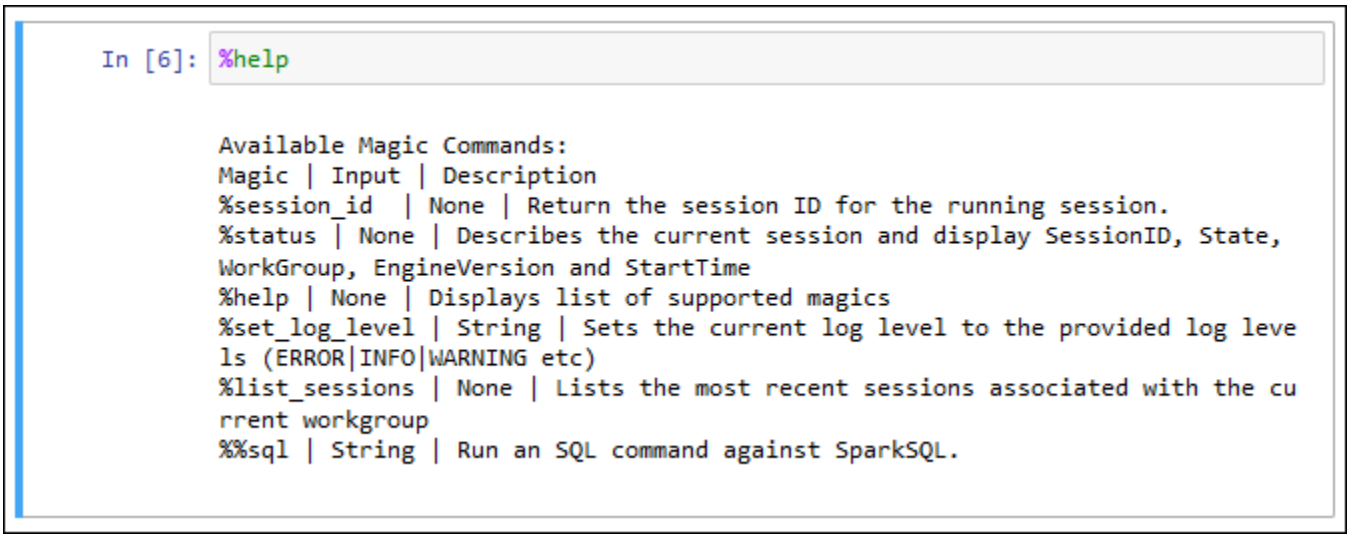
- n int (オプション)。出力する行数。
- truncate — bool または int (オプション) — true の場合、20 文字を超える文字列を切り捨てます。1 より大きい数値を設定すると、長い文字列を指定した長さに切り詰め、セルを右詰めにします。
- vertical — bool (オプション)。true の場合、出力の行が縦方向に (列値ごとに 1 行) 表示されます。

ラインマジック

1 行に記述されているマジックは、先頭にパーセント記号 (%) が付き、ラインマジック関数またはラインマジックと呼ばれます。

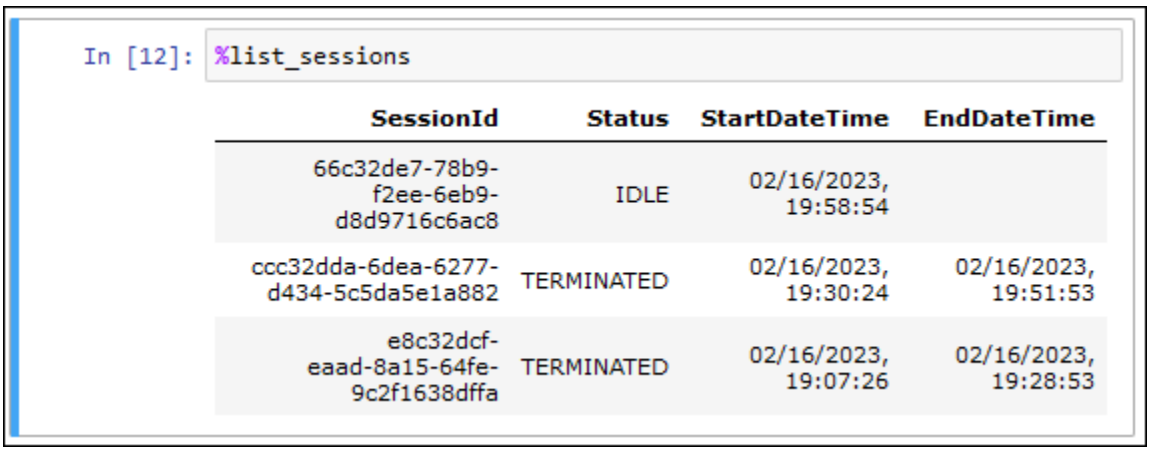
%help

利用できるマジックコマンドの説明が表示されます。



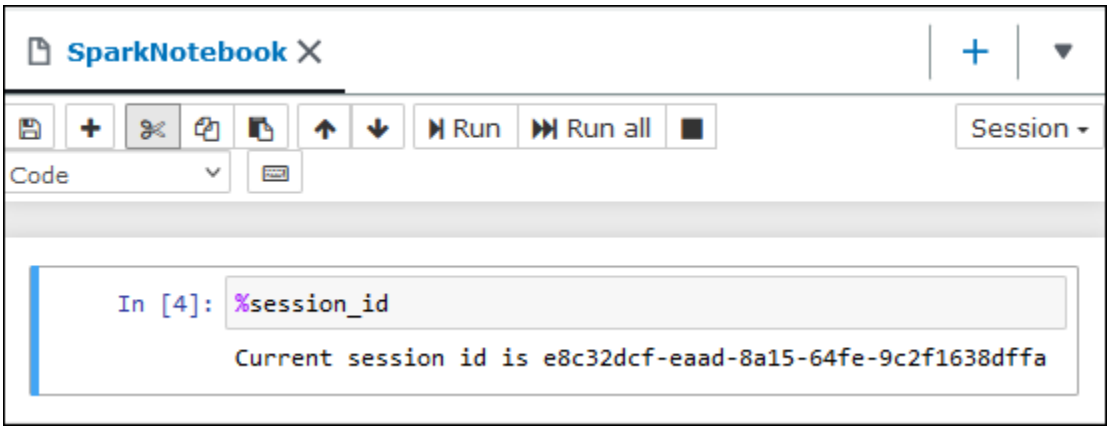
%list_sessions

ノートブックに関連するセッションを一覧表示します。各セッションの情報には、セッション ID、セッションステータス、セッションが開始および終了した日時が含まれます。



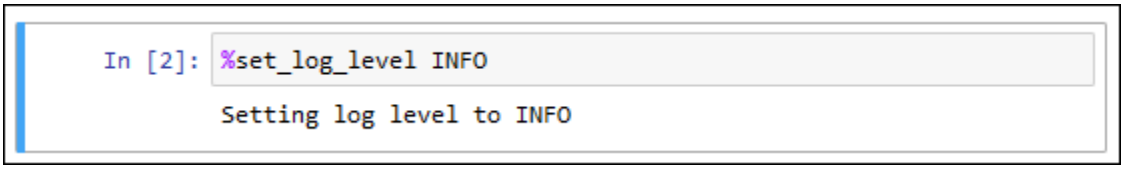
%session_id

現在のセッション ID を取得します。



%set_log_level

指定されたログレベルを使用するように、ロガーを設定またはリセットします。使用できる値は DEBUG、ERROR、FATAL、INFO、WARN、WARNING のいずれかです。値は大文字にする必要があり、一重引用符または二重引用符で囲むことはできません。



%status

現在のセッションについて説明します。出力には、セッション ID、セッション状態、ワークグループ名、PySpark エンジンバージョン、セッション開始時間が含まれます。このマジックコマンドは、セッションの詳細を取得するにあたりアクティブなセッションを必要とします。

ステータスに表示される可能性のある値は、以下のとおりです。

CREATING — セッションが開始されている状態です (リソースの取得を含む)。

CREATED — セッションが開始されました。

IDLE — セッションは計算を受け入れることができる状態にあります。

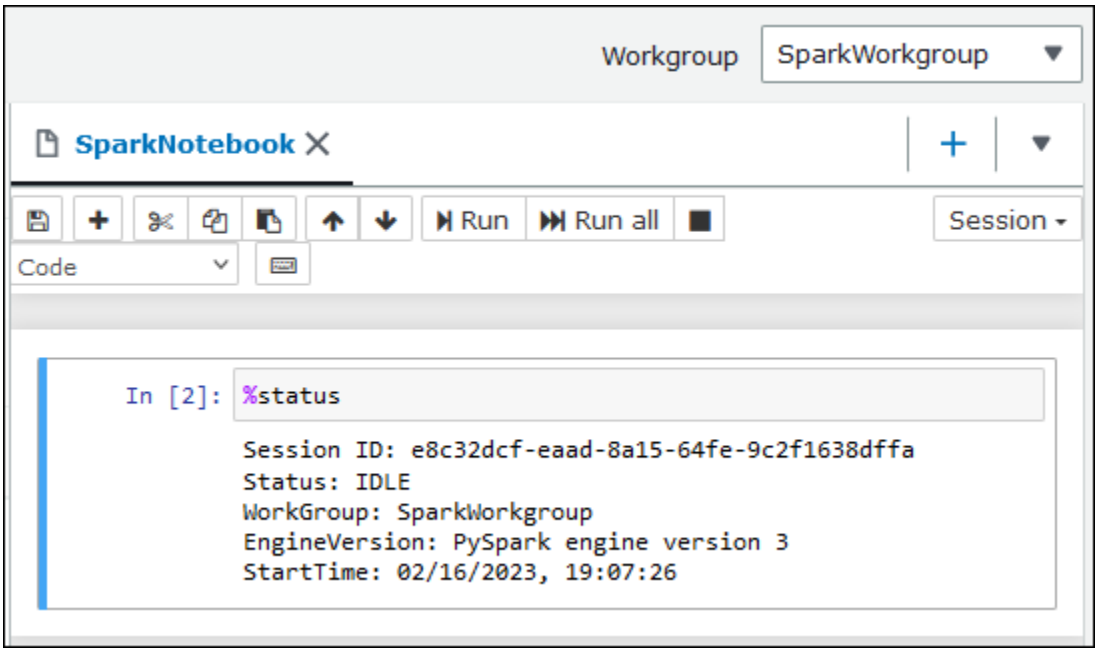
BUSY — セッションは別のタスクを処理中で、計算を受け入れることができません。

TERMINATING – セッションはシャットダウン処理の途中です。

TERMINATED – セッションとそのリソースはもう実行されていません。

DEGRADED — セッションには正常なコーディネーターがありません。

FAILED — 障害が発生したため、セッションとそのリソースは実行されなくなりました。



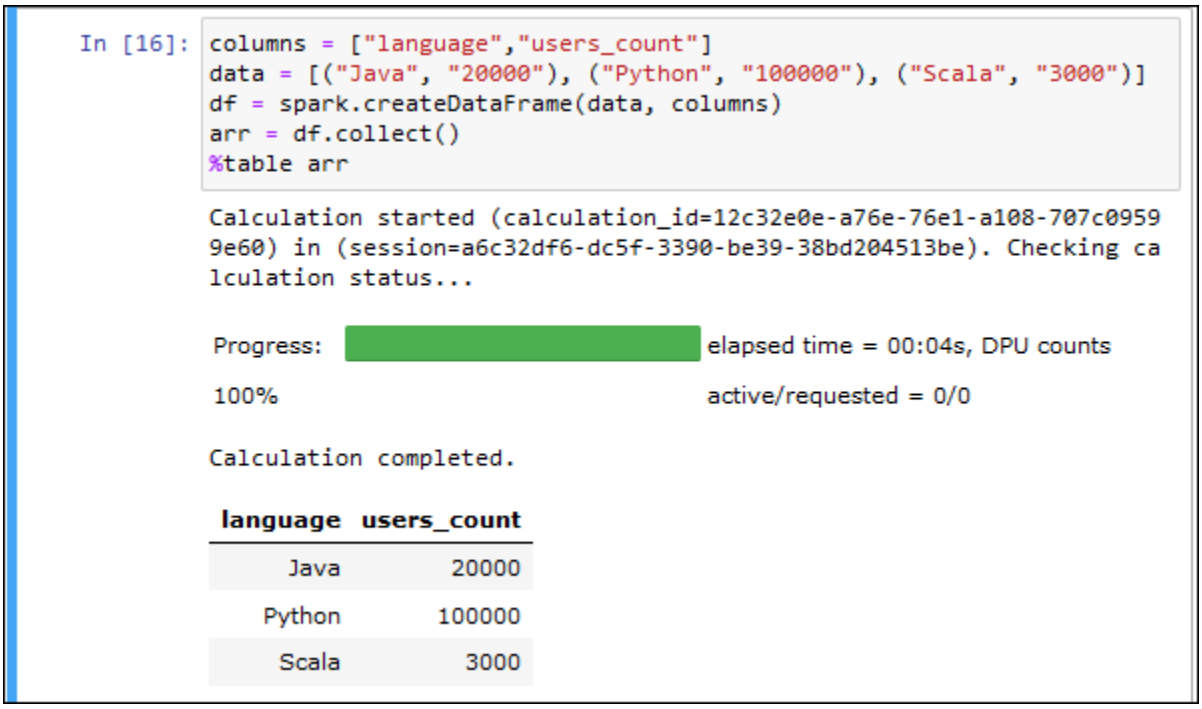
データグラフを作成するためのマジック

このセクションに記載されているラインマジックは、特定タイプのデータのレンダリングや、グラフライブラリとの関連付けに特化したものです。

%table

`%table` マジックコマンドを使用して、データフレームデータをテーブル形式で表示できます。

次の例では、2 列 3 行のデータが含まれるデータフレームを作成し、データを表形式で表示しています。



%matplot

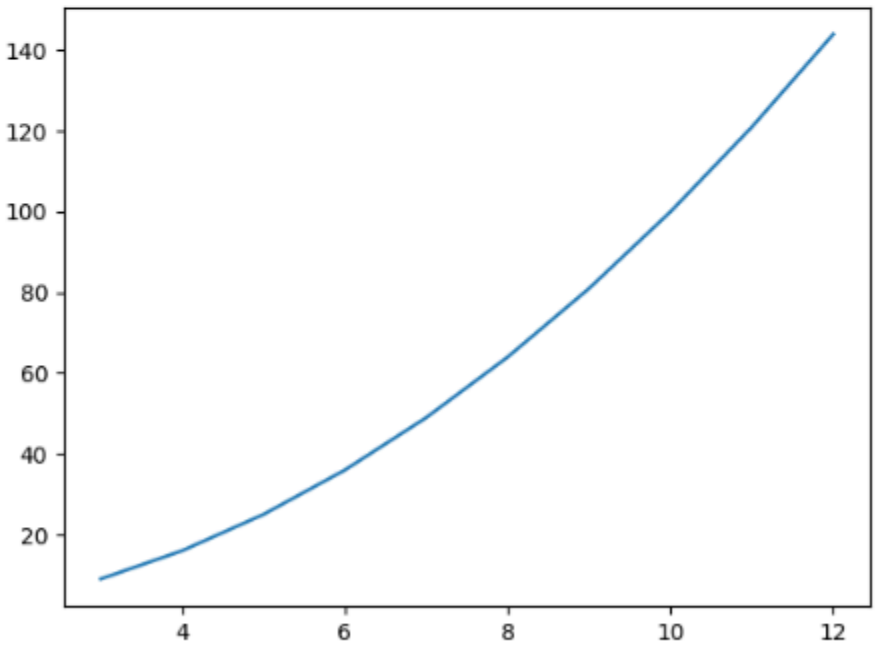
Matplotlib は Python で静的な視覚化、アニメーション化による視覚化、インタラクティブな視覚化を作成するための包括的なライブラリです。matplotlib ライブラリをノートブックセルにインポートした後、`%matplot` マジックコマンドを使用してグラフを作成することができます。

次の例では、matplotlib ライブラリをインポートして X Y 座標セットを作成し、その後 `%matplot` マジックコマンドを使用して点グラフを作成しています。

```
import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
%matplot plt
```

In [12]:
```
import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
%matplot plt
```

```
Calculation started (calculation_id=5ac32e04-81b6-
9ee7-ce55-539ee2ce383e) in (session=a6c32df6-dc5f-
3390-be39-38bd204513be). Checking calculation stat
us...

Progress:                                elapsed time =

100%                                     00:02s

Calculation completed.
```




```
[<matplotlib.lines.Line2D object at 0x7f6e29e580>]
```

matplotlib ライブラリと seaborn ライブラリを併用する

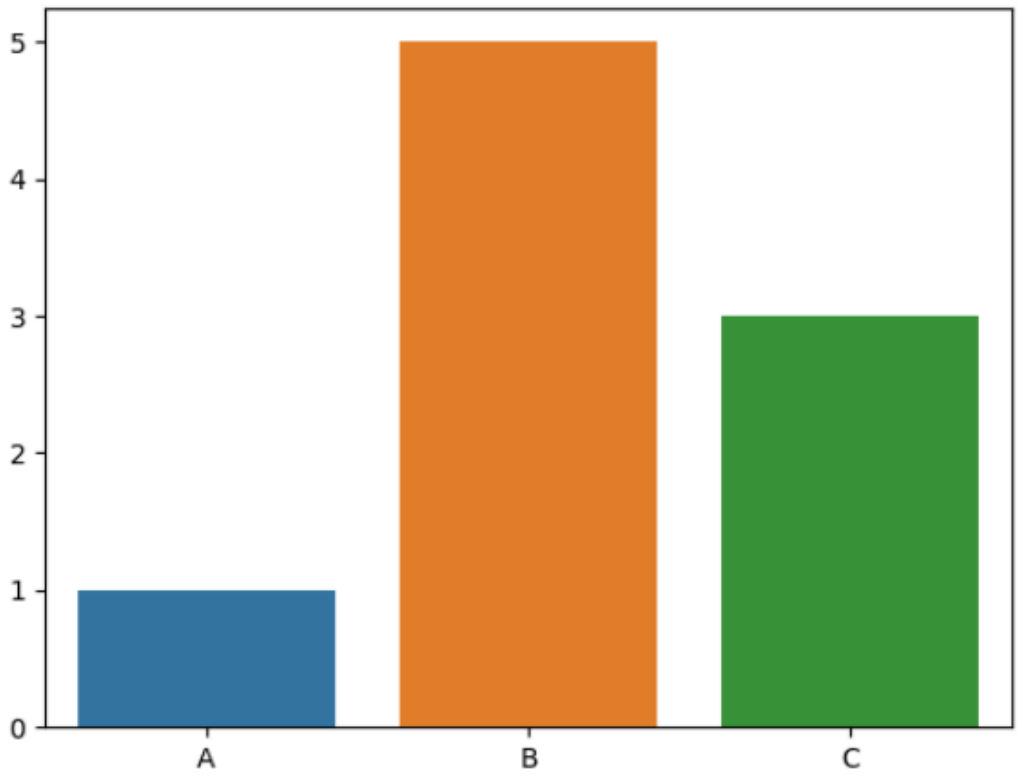
Seaborn は Python で統計グラフィックを作成するためのライブラリです。matplotlib 上に構築され、pandas (Python データ分析) データ構造と密接に統合されます。`%matplot` マジックコマンドを使用して、seaborn データをレンダリングすることもできます。

次の例では、matplotlib ライブラリと seaborn ライブラリの両方を使用して、単純な棒グラフを作成しています。

```
import matplotlib.pyplot as plt
import seaborn as sns

x = ['A', 'B', 'C']
y = [1, 5, 3]

sns.barplot(x, y)
%matplot plt
```

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns


x = ['A', 'B', 'C']
y = [1, 5, 3]

sns.barplot(x, y)
%matplot plt
```

```
Calculation started (calculation_id=08c32e1b-233b-4a72-6571-1ae7a28a7b78)
in (session=64c32e1a-f45e-1d52-b54b-85202e2a9233). Checking calculation st
atus...

Progress: 100% elapsed time = 00:04s

Calculation completed.
```

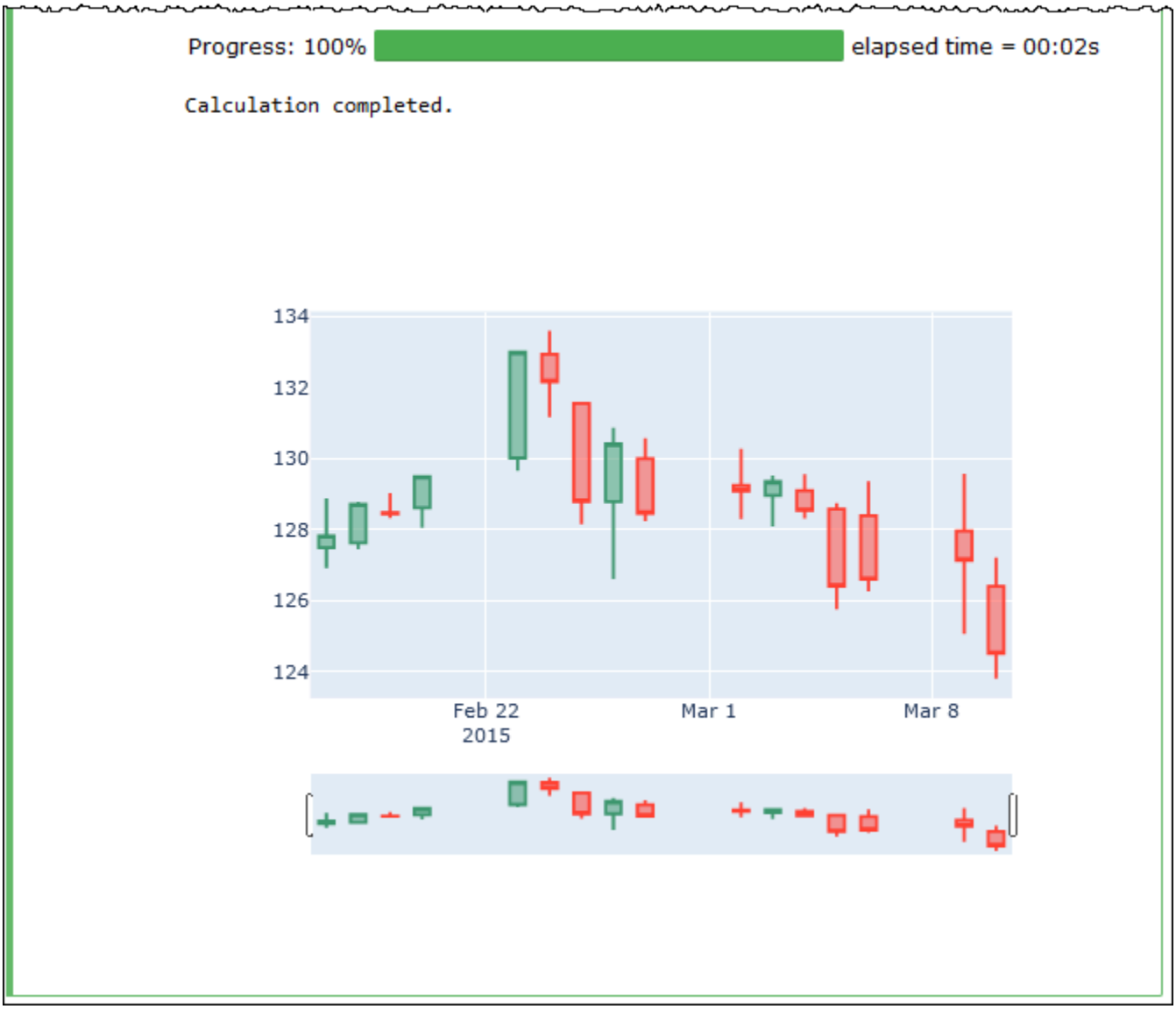


%plotly

Plotly は Python 用のオープンソースのグラフライブラリであり、インタラクティブなグラフを作成するために使用できます。%ploty マジックコマンドを使用して、ploty データをレンダリングすることができます。

次の例では、株価データに StringIO、plotly、pandas ライブラリを使用して、2015 年 2 月と 3 月の株式活動グラフを作成しています。

```
from io import StringIO
csvString = """
Date,AAPL.Open,AAPL.High,AAPL.Low,AAPL.Close,AAPL.Volume,AAPL.Adjusted,dn,mavg,up,direction
2015-02-17,127.489998,128.880005,126.919998,127.830002,63152400,122.905254,106.7410523,117.9276
2015-02-18,127.629997,128.779999,127.449997,128.720001,44891700,123.760965,107.842423,118.94033
2015-02-19,128.479996,129.029999,128.330002,128.449997,37362400,123.501363,108.8942449,119.8891
2015-02-20,128.619995,129.5,128.050003,129.5,48948400,124.510914,109.7854494,120.7635001,131.74
2015-02-23,130.020004,133,129.660004,133,70974100,127.876074,110.3725162,121.7201668,133.067817
2015-02-24,132.940002,133.600006,131.169998,132.169998,69228100,127.078049,111.0948689,122.6648
2015-02-25,131.559998,131.600006,128.149994,128.789993,74711700,123.828261,113.2119183,123.6296
2015-02-26,128.789993,130.869995,126.610001,130.419998,91287500,125.395469,114.1652991,124.2823
2015-02-27,130,130.570007,128.240005,128.460007,62014800,123.510987,114.9668484,124.8426669,134
2015-03-02,129.25,130.279999,128.300003,129.089996,48096700,124.116706,115.8770904,125.4036668,
2015-03-03,128.960007,129.520004,128.089996,129.360001,37816300,124.376308,116.9535132,125.9551
2015-03-04,129.100006,129.559998,128.320007,128.539993,31666300,123.587892,118.0874253,126.4730
2015-03-05,128.580002,128.75,125.760002,126.410004,56517100,121.539962,119.1048311,126.848667,1
2015-03-06,128.399994,129.369995,126.260002,126.599998,72842100,121.722637,120.190797,127.22883
2015-03-09,127.959999,129.570007,125.059998,127.139999,88528500,122.241834,121.6289771,127.6311
2015-03-10,126.410004,127.220001,123.800003,124.510002,68856600,119.71316,123.1164763,127.92350
"""
csvStringIO = StringIO(csvString)

from io import StringIO
import plotly.graph_objects as go
import pandas as pd
from datetime import datetime
df = pd.read_csv(csvStringIO)
fig = go.Figure(data=[go.Candlestick(x=df['Date'],
open=df['AAPL.Open'],
high=df['AAPL.High'],
low=df['AAPL.Low'],
close=df['AAPL.Close'])])
%plotly fig
```



ノートブックファイルの管理

ノートブックエクスプローラーを使用してノートブックを作成したり開いたりできるほか、ノートブックの名前変更、削除、エクスポート、インポート、またはノートブックのセッション履歴の表示にも使用できます。

ノートブックの名前を変更するには

1. 名前を変更するノートブックのアクティブなセッションをすべて終了します。ノートブックの名前を変更する前に、ノートブックのアクティブなセッションを終了する必要があります。
2. [Notebook explorer] (ノートブックエクスプローラー) を開きます。

3. [Notebooks] (ノートブック) リストで、名前を変更するノートブックのオプションボタンを選択します。
4. [Actions] (アクション) メニューで、[Rename] (名前を変更) を選択します。
5. [Rename notebook] (ノートブックの名前を変更) プロンプトで、新しい名前を入力してから、[Save] (保存) を選択します。新しいノートブック名がノートブックのリストに表示されます。

ノートブックを削除するには

1. 削除するノートブックのアクティブなセッションをすべて終了します。ノートブックを削除する前に、ノートブックのアクティブなセッションを終了する必要があります。
2. [Notebook explorer] (ノートブックエクスプローラー) を開きます。
3. [Notebooks] (ノートブック) の一覧で、削除するノートブックのオプションボタンを選択します。
4. [アクション] メニューから、[削除] を選択します。
5. [Delete notebook?] (ノートブックを削除しますか?) のプロンプトで、ノートブックの名前を入力し、[Delete] (削除) を選択して削除を確定します。ノートブック名はノートブックのリストから削除されています。

ノートブックをエクスポートするには

1. [Notebook explorer] (ノートブックエクスプローラー) を開きます。
2. [Notebooks] (ノートブック) リストで、エクスポートするノートブックのオプションボタンを選択します。
3. [Actions] (アクション) メニューで、[Export file] (ファイルのエクスポート) を選択します。

ノートブックをインポートするには

1. [Notebook explorer] (ノートブックエクスプローラー) を開きます。
2. [Import file] (ファイルのインポート) を選択します。
3. インポートするファイルのローカルコンピュータ上の場所を参照し、[Open] (開く) を選択します。インポートしたノートブックはノートブックのリストに表示されます。

ノートブックのセッション履歴を表示するには

1. [Notebook explorer] (ノートブックエクスプローラー) を開きます。
2. [Notebooks] (ノートブック) リストで、セッション履歴を表示するノートブックのオプションボタンを選択します。
3. [Actions] (アクション) メニューから、[Session history] (セッション履歴) を選択します。
4. [History] (履歴) タブで、[Session ID] (セッション ID) を選択すると、セッションとその計算に関する情報が表示されます。

Amazon Athena for Apache Spark で Hive 以外のテーブル形式を使用する

Athena for Spark でセッションやノートブックを使用する場合は、Apache Hive テーブルだけでなく Linux Foundation Delta Lake、Apache Hudi、Apache Iceberg の各テーブルも使用できます。

考慮事項と制約事項

Apache Hive 以外のテーブル形式を Athena for Spark で使用する場合は、次の点を考慮してください。

- 各ノートブックでサポートされるテーブル形式は、Apache Hive 以外は 1 つだけです。Athena for Spark で複数のテーブル形式を使用する場合は、各テーブル形式に対して個別のノートブックを作成してください。Athena for Spark でノートブックを作成するための情報については、「独自のノートブックの作成」を参照してください。
- Delta Lake、Hudi、および Iceberg テーブル形式は、AWS Glue をメタストアとして使用した Athena for Spark でテスト検証済みです。他のメタストアも使用できる可能性はありますが、その使用法は現在サポートされていません。
- その他のテーブル形式を使用するには、Athena コンソールとこのドキュメントの説明にあるように、デフォルトの `spark_catalog` プロパティを上書きしてください。これらの Hive 以外のカタログは、自身のテーブル形式に加えて Hive テーブルも読み取れます。

テーブルのバージョン

次の表は、Amazon Athena for Apache Spark でサポートされている Hive 以外のテーブルバージョンを示しています。

テーブル形式	サポートされる のバージョン
Apache Iceberg	1.2.1
Apache Hudi	0.13
Linux Foundation Delta Lake	2.0.2

Athena for Spark では、これらのテーブル形式 `.jar` ファイルとその依存関係は Spark ドライバーとエグゼキューターのクラスパスに読み込まれます。

トピック

- Apache Iceberg
- Apache Hudi
- Linux Foundation Delta Lake

Apache Iceberg

Apache Iceberg は、Amazon Simple Storage Service (Amazon S3) の大規模データセット用のオープンテーブル形式です。大きなテーブルに対する高速のクエリパフォーマンス、アトミックコミット、同時書き込み、および SQL 互換テーブルの進化を提供します。

Athena for Spark で Apache Iceberg テーブルを使用するには、次の Spark プロパティを設定します。これらのプロパティは、Athena for Spark コンソールでテーブル形式として Apache Iceberg を選択したときに、デフォルトで自動的に設定されます。手順については、「セッションの詳細を編集する」または「独自のノートブックの作成」を参照してください。

```
"spark.sql.catalog.spark_catalog": "org.apache.iceberg.spark.SparkSessionCatalog",
"spark.sql.catalog.spark_catalog.catalog-impl":
 "org.apache.iceberg.aws.glue.GlueCatalog",
"spark.sql.catalog.spark_catalog.io-impl": "org.apache.iceberg.aws.s3.S3FileIO",
"spark.sql.extensions":
 "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
```

以下の手順では、Athena for Spark ノートブックで Apache Iceberg テーブルを使用する方法を示しています。ノートブックの新しいセルで各ステップを実行します。

Athena for Spark で Apache Iceberg テーブルを使用する方法

1. ノートブックで使用する定数を定義します。

```
DB_NAME = "NEW_DB_NAME"
TABLE_NAME = "NEW_TABLE_NAME"
TABLE_S3_LOCATION = "s3://DOC-EXAMPLE-BUCKET"
```

2. Apache Spark DataFrame を作成します。

```
columns = ["language","users_count"]
data = [("Golang", 3000)]
df = spark.createDataFrame(data, columns)
```

3. データベースを作成します。

```
spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
```

4. 空の Apache Iceberg テーブルを作成します。

```
spark.sql("""
CREATE TABLE {}.{} (
language string,
users_count int
) USING ICEBERG
""".format(DB_NAME, TABLE_NAME))
```

5. テーブルにデータ行を挿入します。

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME, TABLE_NAME))
```

6. 新しいテーブルをクエリできることを確認します。

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Spark DataFrames と Iceberg テーブルの使用に関する詳細と例については、Apache Iceberg ドキュメントの「Spark クエリ」を参照してください。

Apache Hudi

Apache Hudi は、増分データ処理を簡素化するオープンソースのデータ管理フレームワークです。レコードレベルの挿入、更新、upsert、および削除アクションがより正確に処理されるため、オーバーヘッドが減少します。

Athena for Spark で Apache Hudi テーブルを使用するには、次の Spark プロパティを設定します。これらのプロパティは、Athena for Spark コンソールでテーブル形式として Apache Hudi を選択したときに、デフォルトで自動的に設定されます。手順については、「セッションの詳細を編集する」または「独自のノートブックの作成」を参照してください。

```
"spark.sql.catalog.spark_catalog": "org.apache.spark.sql.hudi.catalog.HoodieCatalog",
"spark.serializer": "org.apache.spark.serializer.KryoSerializer",
"spark.sql.extensions": "org.apache.spark.sql.hudi.HoodieSparkSessionExtension"
```

以下の手順では、Athena for Spark ノートブックで Apache Hudi テーブルを使用する方法を示しています。ノートブックの新しいセルで各ステップを実行します。

Athena for Spark で Apache Hudi テーブルを使用する方法

1. ノートブックで使用する定数を定義します。

   ```
   DB_NAME = "NEW_DB_NAME"
   TABLE_NAME = "NEW_TABLE_NAME"
   TABLE_S3_LOCATION = "s3://DOC-EXAMPLE-BUCKET"
   ```

2. Apache Spark DataFrame を作成します。

   ```
   columns = ["language","users_count"]
   data = [("Golang", 3000)]
   df = spark.createDataFrame(data, columns)
   ```

3. データベースを作成します。

   ```
   spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
   ```

4. 空の Apache Hudi テーブルを作成します。

   ```
   spark.sql("""
   CREATE TABLE {}.{} (
   language string,
   ```

```
users_count int
) USING HUDI
TBLPROPERTIES (
primaryKey = 'language',
type = 'mor'
);
""".format(DB_NAME, TABLE_NAME))
```

5. テーブルにデータ行を挿入します。

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME,TABLE_NAME))
```

6. 新しいテーブルをクエリできることを確認します。

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Linux Foundation Delta Lake

Linux Foundation Delta Lake は、ビッグデータ分析に使用できるテーブル形式です。Athena for Spark を使用して、Amazon S3 に保存されている Delta Lake テーブルを直接読み取れます。

Athena for Spark で Delta Lake テーブルを使用するには、次の Spark プロパティを設定します。これらのプロパティは、Athena for Spark コンソールでテーブル形式として Delta Lake を選択したときに、デフォルトで自動的に設定されます。手順については、「セッションの詳細を編集する」または「独自のノートブックの作成」を参照してください。

```
"spark.sql.catalog.spark_catalog" : "org.apache.spark.sql.delta.catalog.DeltaCatalog",
"spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension"
```

以下の手順では、Athena for Spark ノートブックで Delta Lake テーブルを使用する方法を示しています。ノートブックの新しいセルで各ステップを実行します。

Athena の Spark で Delta Lake テーブルを使用する方法

1. ノートブックで使用する定数を定義します。

```
DB_NAME = "NEW_DB_NAME"
TABLE_NAME = "NEW_TABLE_NAME"
TABLE_S3_LOCATION = "s3://DOC-EXAMPLE-BUCKET"
```

2. Apache Spark DataFrame を作成します。

```
columns = ["language","users_count"]
data = [("Golang", 3000)]
df = spark.createDataFrame(data, columns)
```

3. データベースを作成します。

```
spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
```

4. 空の Delta Lake テーブルを作成します。

```
spark.sql("""
CREATE TABLE {}.{} (
  language string,
  users_count int
) USING DELTA
""".format(DB_NAME, TABLE_NAME))
```

5. テーブルにデータ行を挿入します。

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME, TABLE_NAME))
```

6. 新しいテーブルをクエリできることを確認します。

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Amazon Athena for Apache Spark での Python ライブラリのサポート

このページでは、Amazon Athena for Apache Spark で使用されるランタイム、ライブラリ、パッケージで使用される用語とライフサイクル管理について説明します。

定義

- [Amazon Athena for Apache Spark] は、オープンソースの Apache Spark をカスタマイズしたバージョンです。現在のバージョンを確認するには、ノートブックのセルで `print(f'{spark.version}')` コマンドを実行します。

- [Athena runtime] (Athena ランタイム) とは、コードが実行される環境のことです。この環境には Python インタープリターと PySpark ライブラリが含まれています。
- [External library or package] (外部ライブラリまたはパッケージ) は、Athena ランタイムには含まれていない Java、Scala の JAR または Python ライブラリですが、Athena for Spark のジョブに含めることができます。外部パッケージは、Amazon が作成することも、ユーザー自身で作成することもできます。
- [Convenience package] (コンビニエンスパッケージ) とは、Athena が選択した外部パッケージのコレクションで、Spark アプリケーションに含めることを選択できます。
- [bundle] (バンドル) は Athena ランタイムと便利なパッケージを組み合わせたものです。
- [User library] (ユーザーライブラリ) は、Athena for Spark ジョブに明示的に追加する外部ライブラリまたはパッケージです。
  - ユーザーライブラリとは、コンビニエンスパッケージの一部ではない外部パッケージです。ユーザーライブラリは、いくつかの `.py` ファイルを作成して圧縮し、`.zip` ファイルをアプリケーションに追加する場合と同様に、ロードとインストールが必要です。
- [Athena for Spark application] (Athena for Spark アプリケーション) とは、ユーザーが Athena for Spark に送信するジョブまたはクエリです。

ライフサイクル管理

ランタイムのバージョニングと廃止予定

Athena ランタイムの主なコンポーネントは、Python インタープリターです。Python は進化が続いている言語であるため、新しいバージョンが定期的にリリースされ、古いバージョンのサポートは停止されます。Athena では、廃止予定の Python インタープリターバージョンでプログラムを実行することはお勧めしていません。また、可能な限り最新の Athena ランタイムを使用することを強くお勧めします。

Athena ランタイムの廃止予定のスケジュールは次のとおりです。

1. Athena が新しいランタイムを提供した後も、Athena は以前のランタイムを 6 か月間サポートし続けます。その間、Athena は以前のランタイムにセキュリティパッチとアップデートを適用します。
2. 6 か月後、Athena は以前のランタイムのサポートを終了します。Athena は、以前のランタイムにセキュリティパッチやその他の更新を適用しなくなります。以前のランタイムを使用している Spark アプリケーションは、テクニカルサポートの対象ではなくなります。

3. 12 か月後、以前のランタイムを使用するワークグループでは Spark アプリケーションを更新または編集することはできなくなります。この期間が終了する前に、Spark アプリケーションを更新することをお勧めします。期間終了後も既存のノートブックを実行できますが、以前のランタイムをまだ使用しているノートブックにはその旨の警告がログ記録されます。
4. 18 か月が経過すると、以前のランタイムを使用してワークグループでジョブを実行できなくなります。

コンビニエンスパッケージのバージョニングと廃止予定

コンビニエンスパッケージの内容は時間の経過とともに変化します。Athena では、これらのコンビニエンスパッケージを追加、削除、またはアップグレードする場合があります。

Athena はコンビニエンスパッケージについて以下のガイドラインを採用しています。

- コンビニエンスパッケージには、1、2、3 などのシンプルなバージョニングスキームがあります。
- 各コンビニエンスパッケージのバージョンには、特定のバージョンの外部パッケージが含まれています。Athena がコンビニエンスパッケージを作成しても、そのコンビニエンスパッケージの外部パッケージセットとそれに対応するバージョンは変更されません。
- Athena は、新しい外部パッケージを含めたり、外部パッケージを削除したり、1 つ以上の外部パッケージのバージョンをアップグレードしたりする場合に、新しいコンビニエンスパッケージバージョンを作成します。

Athena は、そのパッケージが使用する Athena ランタイムを廃止すると便利であるパッケージを廃止します。Athena は、サポートするバンドルの数を制限するために、パッケージをすぐに非推奨にすることができます。

コンビニエンスパッケージの非推奨スケジュールは、Athena ランタイムの非推奨スケジュールに従います。

プリインストールされている Python ライブラリのリスト

プリインストールされている Python ライブラリには以下が含まれます。

```
boto3==1.24.31
botocore==1.27.31
certifi==2022.6.15
charset-normalizer==2.1.0
cycler==0.11.0
```

```
cython==0.29.30
docutils==0.19
fonttools==4.34.4
idna==3.3
jmespath==1.0.1
joblib==1.1.0
kiwisolver==1.4.4
matplotlib==3.5.2
mpmath==1.2.1
numpy==1.23.1
packaging==21.3
pandas==1.4.3
patsy==0.5.2
pillow==9.2.0
plotly==5.9.0
pmdarima==1.8.5
pyathena==2.9.6
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.1
requests==2.28.1
s3transfer==0.6.0
scikit-learn==1.1.1
scipy==1.8.1
seaborn==0.11.2
six==1.16.0
statsmodels==0.13.2
sympy==1.10.1
tenacity==8.0.1
threadpoolctl==3.1.0
urllib3==1.26.10
pyarrow==9.0.0
```

メモ

- MLlib (Apache Spark 機械学習ライブラリ) および `pyspark.ml` パッケージはサポートされていません。
- 現在、Athena for Spark セッションで `pip install` はサポートされていません。

Amazon Athena for Apache Spark への Python ライブラリのインポートについては、「Amazon Athena for Apache Spark へのファイルおよび Python ライブラリのインポート」を参照してください。

Amazon Athena for Apache Spark へのファイルおよび Python ライブラリのインポート

このドキュメントでは、Amazon Athena for Apache Spark にファイルおよび Python ライブラリをインポートする方法の例を紹介します。

考慮事項と制約事項

- Python のバージョン — 現在、Athena for Spark では Python バージョン 3.9.16 が使用されています。Python パッケージは Python のマイナーバージョンにセンシティブであることに注意してください。
- Athena for Spark のアーキテクチャ — Athena for Spark は ARM64 アーキテクチャ上で Amazon Linux 2 を使用しています。Python ライブラリの中には、このアーキテクチャ用のバイナリを配布していないものがあることに注意してください。
- バイナリ共有オブジェクト (SO) — SparkContext addPyFile メソッドは、バイナリ共有オブジェクトを検出しないため、Athena for Spark では共有オブジェクトに依存する Python パッケージを追加するためには使用できません。
- Resilient Distributed Dataset (RDD) — RDD はサポートされていません。
- Dataframe.foreach — PySpark の DataFrame.foreach メソッドはサポートされていません。

例

例では次の規則を使用しています。

- Amazon S3 のプレースホルダーの場所 `s3://DOC-EXAMPLE-BUCKET`。これをユーザーの S3 バケットの場所に置き換えます。
- Unix シェルから実行されるすべてのコードブロックは *`directory_name`* $ として表示されます。例えば、ディレクトリ `/tmp` 内のコマンド `ls` とその出力は次のように表示されます。

```
/tmp $ ls
```

出力

```
file1 file2
```

- ローカル一時ディレクトリにファイルを書き込んだ後にノートブックに追加する
- Amazon S3 からファイルをインポートする
- Python ファイルの追加と UDF の登録
- Python .zip ファイルをインポートする
- Python のライブラリの 2 つのバージョンを別々のモジュールとしてインポートする
- PyPI から Python .zip ファイルをインポートする
- 依存関係のある PyPI から Python .zip ファイルをインポートする

計算に使用するテキストファイルのインポート

このセクションの例では、Athena for Spark 内のノートブックで計算に使用するテキストファイルをインポートする方法を示します。

ローカル一時ディレクトリにファイルを書き込んだ後にノートブックに追加する

次の例は、ローカルの一時ディレクトリにファイルを書き込み、ノートブックに追加してテストする方法です。

```
import os
from pyspark import SparkFiles
tempdir = '/tmp/'
path = os.path.join(tempdir, "test.txt")
with open(path, "w") as testFile:
    _ = testFile.write("5")
sc.addFile(path)

def func(iterator):
    with open(SparkFiles.get("test.txt")) as testFile:
        fileVal = int(testFile.readline())
        return [x * fileVal for x in iterator]

#Test the file
from pyspark.sql.functions import udf
from pyspark.sql.functions import col
```

```
udf_with_import = udf(func)
df = spark.createDataFrame([(1, "a"), (2, "b")])
df.withColumn("col", udf_with_import(col('_2'))).show()
```

出力

```
Calculation completed.
+---+---+-------+
| _1| _2|    col|
+---+---+-------+
|  1|  a|[aaaaa]|
|  2|  b|[bbbbb]|
+---+---+-------+
```

Amazon S3 からファイルをインポートする

次の例は、Amazon S3 からファイルをノートブックにインポートしてテストする方法です。

Amazon S3 からノートブックにファイルをインポートするには

1. 値 5 を含む 1 行を持つ、test.txt という名前のファイルを作成します。
2. Amazon S3 のバケットにファイルを追加します。この例では口ケーション s3://DOC-EXAMPLE-BUCKET を使用しています。
3. 次のコードを使用してファイルをノートブックにインポートし、ファイルをテストします。

```
from pyspark import SparkFiles
sc.addFile('s3://DOC-EXAMPLE-BUCKET/test.txt')

def func(iterator):
   with open(SparkFiles.get("test.txt")) as testFile:
       fileVal = int(testFile.readline())
       return [x * fileVal for x in iterator]

#Test the file
from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)
df = spark.createDataFrame([(1, "a"), (2, "b")])
df.withColumn("col", udf_with_import(col('_2'))).show()
```

出力

```
Calculation completed.
+---+---+-------+
| _1| _2|    col|
+---+---+-------+
|  1|  a|[aaaaa]|
|  2|  b|[bbbbb]|
+---+---+-------+
```

Python ファイルの追加

このセクションの例では、Athena の Spark ノートブックに Python ファイルとライブラリを追加する方法を示します。

Python ファイルの追加と UDF の登録

次の例は、Amazon S3 から Python ファイルをノートブックに追加し、UDF を登録する方法です。

Python ファイルをノートブックに追加して UDF を登録するには

1. 独自の Amazon S3 のロケーションを使用して、次の内容のファイル `s3://DOC-EXAMPLE-BUCKET/file1.py` を作成します。

```
def xyz(input):
    return 'xyz  - udf ' + str(input);
```

2. 同じ S3 のロケーションに、次の内容のファイル `s3://DOC-EXAMPLE-BUCKET/file2.py` を作成します。

```
from file1 import xyz
def uvw(input):
    return 'uvw -> ' + xyz(input);
```

3. Athena for Spark ノートブックで、次のコマンドを実行します。

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/file1.py')
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/file2.py')

def func(iterator):
```

```
    from file2 import uvw
    return [uvw(x) for x in iterator]

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)

df = spark.createDataFrame([(1, "a"), (2, "b")])

df.withColumn("col", udf_with_import(col('_2'))).show(10)
```

出力

```
Calculation started (calculation_id=1ec09e01-3dec-a096-00ea-57289cdb8ce7) in
 (session=c8c09e00-6f20-41e5-98bd-4024913d6cee). Checking calculation status...
Calculation completed.
+---+---+--------------------+
| _1| _2|                 col|
+---+---+--------------------+
| 1 |  a|[uvw -> xyz - ud... |
| 2 |  b|[uvw -> xyz - ud... |
+---+---+--------------------+
```

Python .zip ファイルをインポートする

Python addPyFile と import のメソッドを使用して、Python .zip ファイルをノートブックにインポートできます。

> **Note**
>
> Athena Spark にインポートする .zip ファイルには、Python パッケージのみを含める必要があります。例えば、C ベースのファイルが含まれるパッケージを含めることはサポートされていません。

Python **.zip** ファイルをノートブックにインポートするには

1. ローカルコンピュータの \tmp などのデスクトップディレクトリに、moduletest というディレクトリを作成します。

2. moduletest ディレクトリで、次の内容で hello.py というファイルを作成します。

```
def hi(input):
    return 'hi ' + str(input);
```

3. 同じディレクトリに、__init__.py という名前の空のファイルを追加します。

   ディレクトリの内容を一覧すると、以下のようになります。

```
/tmp $ ls moduletest
__init__.py       hello.py
```

4. zip コマンドを使用して、2 つのモジュールファイルを moduletest.zip というファイルに格納します。

```
moduletest $ zip -r9 ../moduletest.zip *
```

5. .zip のファイルを Amazon S3 のバケットにアップロードしてください。
6. 次のコードを使用して、Python.zip ファイルをノートブックにインポートします。

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/moduletest.zip')

from moduletest.hello import hi

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

hi_udf = udf(hi)

df = spark.createDataFrame([(1, "a"), (2, "b")])

df.withColumn("col", hi_udf(col('_2'))).show()
```

   出力

```
Calculation started (calculation_id=6ec09e8c-6fe0-4547-5f1b-6b01adb2242c) in
 (session=dcc09e8c-3f80-9cdc-bfc5-7effa1686b76). Checking calculation status...
Calculation completed.
+---+---+----+
| _1| _2| col|
+---+---+----+
```

```
|  1|  a|hi a|
|  2|  b|hi b|
+---+---+----+
```

Python のライブラリの 2 つのバージョンを別々のモジュールとしてインポートする

次のコード例は、Amazon S3 内の場所から 2 つの異なるバージョンの Python ライブラリを、2 つの独立したモジュールとして追加およびインポートする方法を示しています。このコードは、S3 から各ライブラリファイルを追加してインポートし、ライブラリのバージョンを出力してインポートを確認するものです。

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/python-third-party-libs-test/
simplejson_v3_15.zip')
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/python-third-party-libs-test/
simplejson_v3_17_6.zip')

import simplejson_v3_15
print(simplejson_v3_15.__version__)
```

出力

```
3.15.0
```

```
import simplejson_v3_17_6
print(simplejson_v3_17_6.__version__)
```

出力

```
3.17.6
```

PyPI から Python .zip ファイルをインポートする

この例では、pip コマンドを使用して、Python Package インデックス (PyPI) から bpabel/piglatin プロジェクトの Python .zip ファイルをダウンロードしています。

PyPI から Python .zip ファイルをインポートするには

1. ローカルデスクトップで、次のコマンドを使用して testpiglatin というディレクトリを作成し、仮想環境を作成します。

```
/tmp $ mkdir testpiglatin
/tmp $ cd testpiglatin
testpiglatin $ virtualenv .
```

出力

```
created virtual environment CPython3.9.6.final.0-64 in 410ms
creator CPython3Posix(dest=/private/tmp/testpiglatin, clear=False,
 no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle,
 via=copy, app_data_dir=/Users/user1/Library/Application Support/virtualenv)
added seed packages: pip==22.0.4, setuptools==62.1.0, wheel==0.37.1
activators
 BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonAct
```

2. プロジェクトを格納するために、unpacked という名前のサブディレクトリを作成します。

```
testpiglatin $ mkdir unpacked
```

3. pip コマンドを使用して、プロジェクトを unpacked ディレクトリにインストールします。

```
testpiglatin $ bin/pip install -t $PWD/unpacked piglatin
```

出力

```
Collecting piglatin
Using cached piglatin-1.0.6-py2.py3-none-any.whl (3.1 kB)
Installing collected packages: piglatin
Successfully installed piglatin-1.0.6
```

4. ディレクトリの内容を確認する。

```
testpiglatin $ ls
```

出力

```
bin lib pyvenv.cfg unpacked
```

5. unpacked のディレクトリに移動して、内容を表示します。

```
testpiglatin $ cd unpacked
unpacked $ ls
```

出力

```
piglatin piglatin-1.0.6.dist-info
```

6. zip コマンドを使用して、piglatin プロジェクトの内容を、library.zip という名前のファイルに格納します。

```
unpacked $ zip -r9 ../library.zip *
```

出力

```
adding: piglatin/ (stored 0%)
adding: piglatin/__init__.py (deflated 56%)
adding: piglatin/__pycache__/ (stored 0%)
adding: piglatin/__pycache__/__init__.cpython-39.pyc (deflated 31%)
adding: piglatin-1.0.6.dist-info/ (stored 0%)
adding: piglatin-1.0.6.dist-info/RECORD (deflated 39%)
adding: piglatin-1.0.6.dist-info/LICENSE (deflated 41%)
adding: piglatin-1.0.6.dist-info/WHEEL (deflated 15%)
adding: piglatin-1.0.6.dist-info/REQUESTED (stored 0%)
adding: piglatin-1.0.6.dist-info/INSTALLER (stored 0%)
adding: piglatin-1.0.6.dist-info/METADATA (deflated 48%)
```

7. (オプション) 次のコマンドを使用して、インポートをローカルでテストします。

   a. Python パスを library.zip ファイルの場所に設定し、Python を起動します。

```
/home $ PYTHONPATH=/tmp/testpiglatin/library.zip
/home $ python3
```

   出力

```
Python 3.9.6 (default, Jun 29 2021, 06:20:32)
[Clang 12.0.0 (clang-1200.0.32.29)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

   b. ライブラリをインポートし、テストコマンドを実行します。

```
>>> import piglatin
>>> piglatin.translate('hello')
```

出力

```
'ello-hay'
```

8. 次のようなコマンドを使用して Amazon S3 から .zip ファイルを追加し、Athena のノートブックにインポートしてテストします。

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/library.zip')

import piglatin
piglatin.translate('hello')

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

hi_udf = udf(piglatin.translate)

df = spark.createDataFrame([(1, "hello"), (2, "world")])

df.withColumn("col", hi_udf(col('_2'))).show()
```

出力

```
Calculation started (calculation_id=e2c0a06e-f45d-d96d-9b8c-ff6a58b2a525) in
 (session=82c0a06d-d60e-8c66-5d12-23bcd55a6457). Checking calculation status...
Calculation completed.
+---+-----+--------+
| _1|   _2|     col|
+---+-----+--------+
|  1|hello|ello-hay|
|  2|world|orld-way|
+---+-----+--------+
```

依存関係のある PyPI から Python .zip ファイルをインポートする

この例では、マークダウン内のテキストを Gemini テキスト形式に変換する md2gemini パッケージを PyPI からインポートします。このパッケージには次の依存関係があります。

```
cjkwrap
mistune
wcwidth
```

依存関係のある Python .zip ファイルをインポートするには

1. ローカルコンピュータで、次のコマンドを使用して testmd2gemini というディレクトリを作成し、仮想環境を作成します。

   ```
   /tmp $ mkdir testmd2gemini
   /tmp $ cd testmd2gemini
   testmd2gemini$ virtualenv .
   ```

2. プロジェクトを格納するために、unpacked という名前のサブディレクトリを作成します。

   ```
   testmd2gemini $ mkdir unpacked
   ```

3. pip コマンドを使用して、プロジェクトを unpacked ディレクトリにインストールします。

   ```
   /testmd2gemini $ bin/pip install -t $PWD/unpacked md2gemini
   ```

   出力

   ```
   Collecting md2gemini
     Downloading md2gemini-1.9.0-py3-none-any.whl (31 kB)
   Collecting wcwidth
     Downloading wcwidth-0.2.5-py2.py3-none-any.whl (30 kB)
   Collecting mistune<3,>=2.0.0
     Downloading mistune-2.0.2-py2.py3-none-any.whl (24 kB)
   Collecting cjkwrap
     Downloading CJKwrap-2.2-py2.py3-none-any.whl (4.3 kB)
   Installing collected packages: wcwidth, mistune, cjkwrap, md2gemini
   Successfully installed cjkwrap-2.2 md2gemini-1.9.0 mistune-2.0.2 wcwidth-0.2.5
   ...
   ```

4. unpacked のディレクトリに移動して、内容を確認します。

```
testmd2gemini $ cd unpacked
unpacked $ ls -lah
```

出力

```
total 16
drwxr-xr-x  13 user1  wheel   416B Jun  7 18:43 .
drwxr-xr-x   8 user1  wheel   256B Jun  7 18:44 ..
drwxr-xr-x   9 user1  staff   288B Jun  7 18:43 CJKwrap-2.2.dist-info
drwxr-xr-x   3 user1  staff    96B Jun  7 18:43 __pycache__
drwxr-xr-x   3 user1  staff    96B Jun  7 18:43 bin
-rw-r--r--   1 user1  staff   5.0K Jun  7 18:43 cjkwrap.py
drwxr-xr-x   7 user1  staff   224B Jun  7 18:43 md2gemini
drwxr-xr-x  10 user1  staff   320B Jun  7 18:43 md2gemini-1.9.0.dist-info
drwxr-xr-x  12 user1  staff   384B Jun  7 18:43 mistune
drwxr-xr-x   8 user1  staff   256B Jun  7 18:43 mistune-2.0.2.dist-info
drwxr-xr-x  16 user1  staff   512B Jun  7 18:43 tests
drwxr-xr-x  10 user1  staff   320B Jun  7 18:43 wcwidth
drwxr-xr-x   9 user1  staff   288B Jun  7 18:43 wcwidth-0.2.5.dist-info
```

5. zip コマンドを使用して、md2gemini プロジェクトの内容を、md2gemini.zip という名前のファイルに格納します。

```
unpacked $ zip -r9 ../md2gemini *
```

出力

```
  adding: CJKwrap-2.2.dist-info/ (stored 0%)
  adding: CJKwrap-2.2.dist-info/RECORD (deflated 37%)
  ....
  adding: wcwidth-0.2.5.dist-info/INSTALLER (stored 0%)
  adding: wcwidth-0.2.5.dist-info/METADATA (deflated 62%)
```

6. (オプション) 次のコマンドを使用して、ライブラリがローカルコンピュータで動作することをテストします。

   a. Python パスを md2gemini.zip ファイルの場所に設定し、Python を起動します。

```
/home $ PYTHONPATH=/tmp/testmd2gemini/md2gemini.zip
/home python3
```

b. ライブラリをインポートしてテストを実行します。

```
>>> from md2gemini import md2gemini
>>> print(md2gemini('[abc](https://abc.def)'))
```

出力

```
https://abc.def abc
```

7. 次のコマンドを使用して Amazon S3 から .zip ファイルを追加し、Athena のノートブックにインポートして、UDF 以外のテストを実行します。

```
# (non udf test)
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/md2gemini.zip')
from md2gemini import md2gemini
print(md2gemini('[abc](https://abc.def)'))
```

出力

```
Calculation started (calculation_id=0ac0a082-6c3f-5a8f-eb6e-f8e9a5f9bc44) in
 (session=36c0a082-5338-3755-9f41-0cc954c55b35). Checking calculation status...
Calculation completed.
=> https://abc.def (https://abc.def/) abc
```

8. 次のコマンドを使用して、UDF テストを実行します。

```
# (udf test)

from pyspark.sql.functions import udf
from pyspark.sql.functions import col
from md2gemini import md2gemini


hi_udf = udf(md2gemini)
df = spark.createDataFrame([(1, "[first website](https://abc.def)"), (2, "[second
 website](https://aws.com)")])
df.withColumn("col", hi_udf(col('_2'))).show()
```

出力

```
Calculation started (calculation_id=60c0a082-f04d-41c1-a10d-d5d365ef5157) in
 (session=36c0a082-5338-3755-9f41-0cc954c55b35). Checking calculation status...
Calculation completed.
+---+--------------------+--------------------+
| _1|                  _2|                col|
+---+--------------------+--------------------+
|  1|[first website](h...|=> https://abc.de...|
|  2|[second website](...|=> https://aws.co...|
+---+--------------------+--------------------+
```

JAR ファイルとカスタム Spark 設定を追加する

Amazon Athena for Apache Spark でセッションを作成または編集する場合、[Spark プロパティ] を使用してセッションの `.jar` ファイル、パッケージ、またはその他のカスタム設定を指定できます。Spark のプロパティを指定するには、Athena コンソール、AWS CLI、または Athena API を使用できます。

Athena コンソールを使用した Spark プロパティの指定

Athena コンソールでは、ノートブックの作成時や現在のセッションの編集時に Spark プロパティを指定できます。

[ノートブックの作成] または [セッションの詳細の編集] ダイアログボックスにプロパティを追加する方法

1. [Spark のプロパティ] を拡張します。
2. プロパティを追加するには、[テーブルで編集] または [JSON で編集] オプションを使用します。
   - [テーブルで編集] オプションを使用する場合は、[プロパティを追加] を選択してプロパティを追加するか、[削除] を選択してプロパティを削除します。[キー] ボックスと [値] ボックスを使用して、プロパティ名とその値を入力します。
     - カスタム `.jar` ファイルを追加するには、`spark.jars` プロパティを使用します。
     - パッケージファイルを指定するには、`spark.jars.packages` プロパティを使用します。
   - 設定を直接入力して編集するには、[JSON で編集] オプションを選択します。JSON テキストエディタでは、次のタスクを実行できます。
     - JSON テキストをクリップボードにコピーするには、[コピー] を選択します。
     - JSON エディタからすべてのテキストを削除するには、[クリア] を選択します。

- 設定 (歯車) アイコンを選択して行折り返しの動作を設定するか、JSON エディタのカラーテーマを選択します。

メモ

- Athena for Spark でプロパティを設定できます。これは [SparkConf] オブジェクトで [Spark properties] を直接設定する場合と同じです。
- `spark.` プレフィックスが付いたすべての Spark プロパティを開始します。他のプレフィックスが付いたプロパティは無視されます。
- Athena のカスタム設定では、すべての Spark プロパティを利用できるわけではありません。設定が制限されている `StartSession` リクエストを送信すると、セッションは開始時に失敗します。
  - `spark.athena.` プレフィックスは予約されているため使用できません。

AWS CLI または Athena API を使用してカスタム設定を提供する

AWS CLI または Athena API を使用してセッション設定を指定するには、StartSession API アクションまたは start-session CLI コマンドを使用します。`StartSession` リクエストでは、EngineConfiguration オブジェクトの `SparkProperties` フィールドを使用して設定情報を JSON 形式で渡します。これにより、指定した設定でセッションが開始されます。リクエストの構文については、「Amazon Athena API リファレンス」の「StartSession」を参照してください。

セッション開始エラーのトラブルシューティング

セッション開始時にカスタム設定エラーが発生すると、Athena for Spark コンソールにエラーメッセージバナーが表示されます。セッション開始エラーをトラブルシューティングするには、セッション状態の変更やログ情報を確認できます。

セッション状態の変更に関する情報を表示する

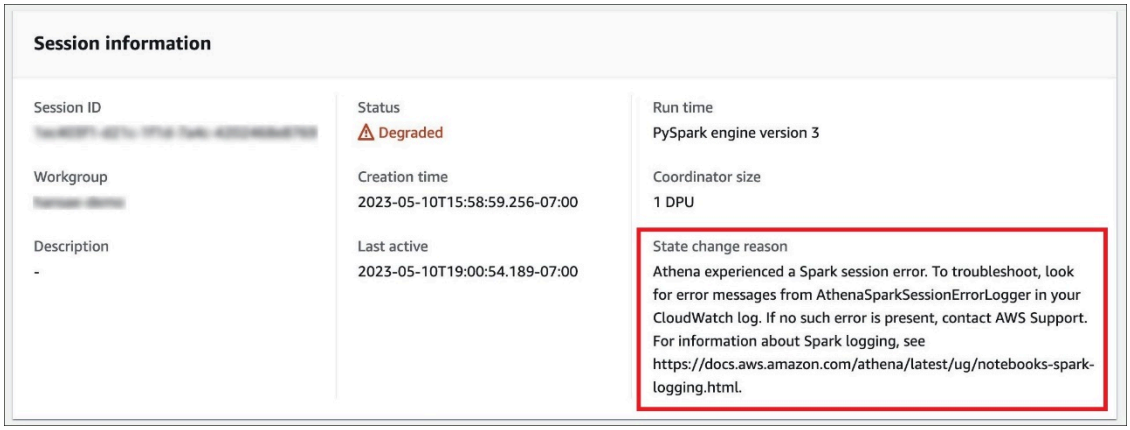
セッション状態の変更に関する詳細は、Athena ノートブックエディタまたは Athena API から取得できます。

Athena コンソールでセッション状態情報を表示する方法

1. Athena ノートブックエディタの右上にある [セッション] メニューから [詳細を表示] を選択します。

2. [現在のセッション] タブを表示します。[セッション情報] セクションには、セッション ID、ワークグループ、ステータス、状態変更理由などの情報が表示されます。

   次の画面キャプチャ例は、Athena の Spark セッションエラーに関連して、[セッション情報] ダイアログボックスの [状態変更の理由] セクションに表示される情報を示しています。



Athena API を使用してセッション状態情報を表示する方法

- Athena API では、SessionStatus オブジェクトの `StateChangeReason` フィールドでセッション状態の変更情報を確認できます。

> **Note**
>
> セッションを手動で停止した後、またはアイドルタイムアウト (デフォルトは 20 分) 後にセッションが停止した場合には、[StateChangeReason] の値が [リクエストに従ってセッションを終了済み] に変わります。

ロギングを使用したセッション開始エラーのトラブルシューティング

セッション開始時に発生するカスタム設定エラーは Amazon CloudWatch にログとして記録されます。CloudWatch Logs 内で `AthenaSparkSessionErrorLogger` からのエラーメッセージを検索し、失敗したセッションの開始をトラブルシューティングします。

Spark ロギングの詳細については、「Athena で Spark アプリケーションイベントのログを記録する」を参照してください。

Athena for Spark のトラブルシューティングセッションの詳細については、「セッションのトラブルシューティング」を参照してください。

サポートされているデータおよびストレージ形式

次の表に、Athena for Apache Spark でネイティブにサポートされている形式を示します。

[Data format] (データ形式)	読み込み	書き込み	[Write compression] (書き込み圧縮)
parquet	はい	はい	なし、非圧縮、snappy、gzip
orc	はい	はい	なし、snappy、zlib、lzo
json	はい	はい	bzip2、gzip、デフレート
csv	はい	はい	bzip2、gzip、デフレート
text	はい	はい	なし、bzip2、gzip、デフレート
バイナリファイル	はい	該当なし	該当なし

CloudWatch メトリクスによる Apache Spark の計算のモニタリング

Spark 対応のワークグループの [Publish CloudWatch metrics] オプションが選択されている場合、Athena は計算関連のメトリクスを Amazon CloudWatch に発行します。カスタムダッシュボードを作成して、CloudWatch コンソールでメトリクスにアラームおよびトリガーを設定できます。

Athena は、次のメトリクスを `AmazonAthenaForApacheSpark` 名前空間で CloudWatch コンソールに発行します。

- `DPUCount` - 計算を実行するために、セッション中に消費された DPU の数。

このメトリクスには、次のディメンションがあります。

- `SessionId` - 計算が送信されるセッションの ID。
- `WorkGroup` - ワークグループの名前。

Amazon CloudWatch コンソールで Spark 対応のワークグループ用メトリクスを表示する

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Metrics]、[All metrics] を選択します。
3. [AmazonAthenaForApacheSpark] 名前空間を選択してください。

CLI を使用してメトリクスを表示するには

- 次のいずれかを行います。
  - Athena Spark 対応のワークグループ用メトリクスを一覧表示するには、コマンドプロンプトを開き、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace "AmazonAthenaForApacheSpark"
```

  - すべての使用可能なメトリクスのリストを表示するには、次のコマンドを使用します。

```
aws cloudwatch list-metrics
```

Athena の Apache Spark 向けの CloudWatch メトリクスおよびディメンションのリスト

Spark 対応の Athena ワークグループで CloudWatch メトリクスを有効にしている場合、Athena はワークグループごとに次のメトリクスを CloudWatch に送信します。メトリクスは `AmazonAthenaForApacheSpark` 名前空間を使用します。

メトリクス名	説明
DPUCount	計算を実行するためにセッション中に消費された DPU (Data Processing Units) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。

このメトリクスには、次のディメンションがあります。

ディメンション	説明
SessionId	計算が送信されるセッションの ID。
WorkGroup	ワークグループの名前。

Athena for Spark でリクエスタによる Amazon S3 バケットの支払いを有効にする

Amazon S3 バケットをリクエスタが支払うように設定すると、クエリに関連するデータアクセス料金とデータ転送料金がクエリを実行したユーザーのアカウントに請求されます。詳細については、「Amazon S3 ユーザーガイド」の「ストレージ転送と使用量のリクエスタ支払いバケットの使用」を参照してください。

Athena for Spark では、リクエスタ支払いバケットがワークグループごとではなく、セッションごとに有効化されます。リクエスタ支払いバケットを有効にする大まかな手順は次のとおりです。

1. Amazon S3 コンソールで、バケットのプロパティに対するリクエスタによる支払いを有効にし、アクセスを指定するバケットポリシーを追加します。
2. IAM コンソールで、バケットへのアクセスを許可する IAM ポリシーを作成し、そのポリシーをリクエスタ支払いバケットへのアクセスに使用する IAM ロールにアタッチします。
3. Athena for Spark では、セッションプロパティを追加してリクエスタによる支払い機能を有効にします。

1. Amazon S3 バケットでリクエスタ支払いを有効にし、バケットポリシーを追加する

Amazon S3 バケットのリクエスタ支払いを有効にするには

1. https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. バケットのリストで、リクエスタ支払いを有効にするバケットのリンクを選択します。
3. バケットのページで、[プロパティ] タブを選択します。
4. [リクエスタ支払い] セクションまで下にスクロールし、[編集] を選択します。

5. [リクエスタ支払いを編集] ページで、[有効化] を選択し、[変更の保存] を選択します。
6. [アクセス許可] タブを選択します。
7. [バケットポリシー] セクションで、[編集] を選択します。
8. [バケットポリシーを編集] ページで、必要なバケットポリシーをソースバケットに適用します。以下のポリシー例では、すべての AWS プリンシパル ("AWS": "*") へのアクセスを許可していますが、アクセスをより細かく設定することもできます。例えば、別のアカウントの特定の IAM ロールのみを指定したい場合があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-
bucket",
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-bucket/
*"
            ]
        }
    ]
}
```

2. IAM ポリシーを作成して、それを IAM ロールにアタッチします。

次に、バケットへのアクセスを許可する IAM ポリシーを作成します。次に、リクエスタ支払いバケットへのアクセスに使用されるロールにポリシーをアタッチします。

リクエスタ支払いバケット用の IAM ポリシーを作成し、そのポリシーをロールにアタッチするには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. IAM コンソールのナビゲーションペインで、[ポリシー] を選択します。
3. [Create policy] を選択します。

4. [JSON] を選択します。
5. [ポリシーエディタ] で、以下のポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-
bucket",
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-bucket/
*"
            ]
        }
    ]
}
```

6. [Next] を選択します。
7. [確認と作成] ページで、ポリシーの 名前とオプションの説明を入力し、[ポリシーの作成] を選択します。
8. ナビゲーションペインで Roles (ロール) を選択します。
9. [ロール] ページで、使用したいロールを見つけて、ロール名のリンクを選択します。
10. [アクセス許可ポリシー] で、[アクセス許可を追加] と [ポリシーをアタッチ] を続けて選択します。
11. [その他の許可ポリシー] セクションで、作成したポリシーのチェックボックスをオンにし、[アクセス許可を追加] を選択します。

3. Athena for Spark セッションプロパティを追加する

Amazon S3 バケットとリクエスタ支払いに関連するアクセス許可を設定した後、Athena for Spark セッションでこの機能を有効にできます。

Athena for Spark セッションでリクエスタ支払いバケットを有効にするには

1. ノートブックエディタで、右上の [Session] (セッション) メニューから [Edit session] (セッションの編集) を選択します。
2. [Spark のプロパティ] を拡張します。
3. [JSON で編集] を選択します。
4. JSON テキストエディタで、以下を入力します。

```
{
  "spark.hadoop.fs.s3.useRequesterPaysHeader":"true"
}
```

5. [保存] を選択します。

Apache Spark 暗号化を有効にする

Athena で Apache Spark 暗号化を有効にできます。有効にすると、Spark ノード間で転送中のデータが暗号化され、Spark にローカル保存されている保管中のデータも暗号化されます。このデータのセキュリティを強化するために、Athena では以下の暗号化設定を使用しています。

```
spark.io.encryption.keySizeBits="256"
spark.io.encryption.keygen.algorithm="HmacSHA384"
```

Spark 暗号化を有効にするには、Athena コンソール、AWS CLI、または Athena API を使用できます。

Athena コンソールを使用して Spark 暗号化を有効にする

Spark 暗号化が有効になっているノートブックを新規作成する方法

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。
3. 次のいずれかを行います。

   - [Notebook explorer] (ノートブックエクスプローラー) で、[Create notebook] (ノートブックの作成) を選択します。

- [Notebook editor] (ノートブックエディタ) で、[Create notebook] (ノートブックの作成) を選択するか、プラスアイコン ([+]) を選択してノートブックを追加します。

4. [ノートブック名] に、ノートブックの名前を入力します。
5. [Spark プロパティ] オプションを拡張します。
6. [Spark 暗号化を有効にする] を選択します。
7. [Create] (作成) を選択します。

作成したノートブックセッションが暗号化されます。新しいノートブックは、通常どおりに使用してください。後ほど、ノートブックを使用する新しいセッションを起動すると、新しいセッションも暗号化されます。

また、Athena コンソールを使用して、既存のノートブックの Spark 暗号化を有効にできます。

既存のノートブックの暗号化を有効にする方法

1. 以前に作成したノートブックの [新しいセッションを開く] を実行します。
2. ノートブックエディタで、右上の [Session] (セッション) メニューから [Edit session] (セッションの編集) を選択します。
3. [セッションの詳細の編集] ダイアログボックスにある [Spark プロパティ] を拡張します。
4. [Spark 暗号化を有効にする] を選択します。
5. [Save] を選択します。

コンソールに、暗号化が有効になっている新しいセッションが起動します。このノートブック用に作成する後続のセッションでも、暗号化が有効になっています。

AWS CLI を使用して Spark 暗号化を有効にする方法

AWS CLI を使用して、適切な Spark プロパティを指定することで、セッションを起動するときに暗号化を有効にできます。

AWS CLI を使用して Spark 暗号化を有効にする

1. 次のようなコマンドを使用して、Spark 暗号化プロパティを指定するエンジン設定 JSON オブジェクトを作成します。

```
ENGINE_CONFIGURATION_JSON=$(
```

```
  cat <<EOF
{
    "CoordinatorDpuSize": 1,
    "MaxConcurrentDpus": 20,
    "DefaultExecutorDpuSize": 1,
    "SparkProperties": {
      "spark.authenticate": "true",
      "spark.io.encryption.enabled": "true",
      "spark.network.crypto.enabled": "true"
    }
}
EOF
)
```

2. AWS CLI で、次の例のように、athena start-session コマンドを使用して、作成した JSON オブジェクトを --engine-configuration 引数に渡します。

```
aws athena start-session \
   --region "region" \
   --work-group "your-work-group" \
   --engine-configuration "$ENGINE_CONFIGURATION_JSON"
```

Athena API を使用して Spark 暗号化を有効化

Athena API で Spark 暗号化を有効にするには、StartSession アクションとその EngineConfiguration SparkProperties パラメータを使用して、StartSession リクエスト内の暗号化設定を指定します。

Athena for Spark でのクロスアカウント AWS Glue アクセスの設定

このトピックでは、コンシューマーアカウント *666666666666* と所有者アカウント *999999999999* を AWS Glue へのクロスアカウントアクセス用に設定する方法を示します。アカウントを設定すると、コンシューマーアカウントで所有者の AWS Glue データベースとテーブルに対して Athena for Spark からのクエリを実行できます。

1. AWS Glue で、コンシューマーロールへのアクセスを許可してください

AWS Glue では、所有者がコンシューマーのロールに所有者の AWS Glue データカタログへのアクセスを許可するポリシーを作成します。

コンシューマーロールに所有者のデータカタログへのアクセスを許可する AWS Glue ポリシーを追加するには

1. カタログ所有者のアカウントを使用して、AWS Management Console にログインします。
2. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
3. ナビゲーションペインで、[データカタログ] を展開し、[カタログ設定] を選択します。
4. [データカタログ設定] ページの [アクセス許可] セクションで、次のようなポリシーを追加します。このポリシーでは、コンシューマーアカウント *666666666666* に、所有者アカウント *999999999999* のデータカタログへのアクセスを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Cataloguers",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::666666666666:role/Admin",
          "arn:aws:iam::666666666666:role/AWSAthenaSparkExecutionRole"
        ]
      },
      "Action": "glue:*",
      "Resource": [
        "arn:aws:glue:us-west-2:999999999999:catalog",
        "arn:aws:glue:us-west-2:999999999999:database/*",
        "arn:aws:glue:us-west-2:999999999999:table/*"
      ]
    }
  ]
}
```

2. アクセス用のコンシューマーアカウントの設定

コンシューマーアカウントで、所有者の AWS Glue Data Catalog、データベース、テーブルへのアクセスを許可するポリシーを作成し、そのポリシーをロールにアタッチします。次の例では、コンシューマーアカウント *666666666666* を使用しています。

所有者の AWS Glue Data Catalog にアクセスするための AWS Glue ポリシーを作成するには

1. コンシューマーアカウントを使用して、AWS Management Console にログインします。
2. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
3. ナビゲーションペインの [アクセス管理] を展開し、[ポリシー] を選択します。
4. [Create policy] を選択します。
5. [アクセス許可の指定] ページで、[JSON] を選択します。
6. ポリシーエディターで、所有者アカウントのデータカタログに対する AWS Glue アクションを許可する次のような JSON ステートメントを入力します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/*",
                "arn:aws:glue:us-east-1:999999999999:table/*"
            ]
        }
    ]
}
```

7. [次へ] をクリックします。
8. [確認と作成] ページで、[ポリシー名] にポリシーの名前を入力します。
9. [ポリシーを作成] を選択します。

次に、コンシューマーアカウントの IAM コンソールを使用して、作成したポリシーをIAM ロールまたはコンシューマーアカウントが所有者のデータカタログにアクセスするために使用するロールにアタッチします。

AWS Glue ポリシーをコンシューマーアカウントのロールにアタッチするには

1. コンシューマーアカウントの IAM コンソールのナビゲーションペインで [ロール] を選択します。
2. [ロール] ページで、ポリシーをアタッチしたいロールを探します。

3. [アクセス許可を追加]、[ポリシーをアタッチ] の順に選択します。
4. 作成したポリシーを見つけます。
5. ポリシーのチェックボックスを選択し、[アクセス許可を追加] を選択します。
6. 手順を繰り返して、使用したい他のロールにこのポリシーを追加します。

3. セッションの設定とクエリの作成

Athena Spark のリクエスタアカウントで、指定したロールを使用してノートブックの作成するか、または現在のセッションを編集することで、アクセスをテストするセッションを作成します。セッションプロパティを設定する場合は、次のいずれかを指定します。

- Glue カタログ区切り文字 — これを使用して、所有者アカウント ID をクエリに含めます。セッションを使用して別の所有者のデータカタログをクエリする場合は、この方法を使用してください。
- Glue カタログ ID — これを使用して、データベースに直接クエリを実行します。この方法は、セッションを使用して 単一の所有者のデータカタログのみをクエリする場合に便利です。

AWS Glue カタログ区切り文字アプローチの使用

セッションプロパティを編集するときは、以下を追加します。

```
{
    "spark.hadoop.aws.glue.catalog.separator": "/"
}
```

セルでクエリを実行するときは、次の例のような構文を使用します。この FROM 節では、データベース名の前にカタログ ID と区切り文字が必要であることに注意してください。

```
df = spark.sql('SELECT requestip, uri, method, status FROM `999999999999/
mydatabase`.cloudfront_logs LIMIT 5')
df.show()
```

AWS Glue カタログ ID アプローチの使用

セッションプロパティを編集するときは、次のプロパティを入力します。所有者アカウント ID は *999999999999* に置き換えてください。

```
{
    "spark.hadoop.hive.metastore.glue.catalogid": "999999999999"
}
```

セルでクエリを実行するときは、次のような構文を使用します。 この FROM 節では、データベース名の前にカタログ ID と区切り文字を挿入する必要がないことに注意してください。

```
df = spark.sql('SELECT * FROM mydatabase.cloudfront_logs LIMIT 10')
df.show()
```

追加リソース

AWS Glue データカタログへのクロスアカウントアクセス

AWS Lake Formation デベロッパーガイドのAWS Glue と Lake Formation の両方を使用したクロスアカウント許可の管理」を参照してください。

AWS 規範的ガイダンス パターンのAmazon Athena を使用して、共有 AWS Glue Data Catalog へのクロスアカウント アクセスを設定します。

Amazon Athena for Apache Spark のサービスクォータ

サービスクォータ (制限とも呼ばれます) とは、AWS アカウントで使用できるサービスリソースまたはオペレーションの最大数のことです。Amazon Athena for Spark と併用できる他の AWS サービスに関するサービスクォータの詳細については、「Amazon Web Services 全般のリファレンス」の「AWS サービスクォータ」を参照してください。

> **Note**
>
> 新規の AWS アカウントの低い初期クォータは、経時的に引き上げられる可能性があります。Amazon Athena for Apache Spark は、各 AWS リージョン内のアカウント使用率を常に監視し、使用率に基づいてクォータを自動的に引き上げます。要件が規定の制限を超える場合は、カスタマーサポートにお問い合わせください。

以下の表に Amazon Athena for Apache Spark のサービスクォータを示します。

名前	デフォルト	引き上げ可能	説明
Apache Spark DPU 同時実行	160	[いいえ]	現在の AWS リージョンの単一アカウントで Apache Spark の計算で同時に消費できるデータ処理単位 (DPU) の最大数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。
Apache Spark セッション DPU 同時実行	60	[いいえ]	セッション内の Apache Spark の計算で同時に消費できる DPU の最大数。

Athena ノートブック API

次のリストには、Athena ノートブック API アクションへの参照リンクが含まれています。データ構造とその他の Athena API アクションについては、「Amazon Athena API リファレンス」を参照してください。

- CreateNotebook
- CreatePresignedNotebookUrl
- DeleteNotebook
- ExportNotebook
- GetCalculationExecution
- GetCalculationExecutionCode
- GetCalculationExecutionStatus
- GetNotebookMetadata
- GetSession
- GetSessionStatus
- ImportNotebook
- ListApplicationDPUSizes
- ListCalculationExecutions
- ListExecutors

- ListNotebookMetadata
- ListNotebookSessions
- ListSessions
- StartCalculationExecution
- StartSession
- StopCalculationExecution
- TerminateSession
- UpdateNotebook
- UpdateNotebookMetadata

Athena for Spark の既知の問題

このページでは、Athena for Apache Spark に関する既知の問題についていくつか説明します。

テーブル作成時の不正な引数の例外

Spark では空のロケーションプロパティを使用して AWS Glue にデータベースを作成することはできませんが、Spark の外部で作成されたデータベースには、空の LOCATION プロパティを設定できます。

テーブルを作成し、空の LOCATION フィールドがを持つ AWS Glue データベースを指定すると、次の「IllegalArgumentException: Cannot create a path from an empty string」(IllegalArgumentException: 空の文字列からパスを作成することはできません) というような例外が発生する可能性があります。

例えば、次のコマンドは、AWS Glue のデフォルトデータベースに空の LOCATION フィールドが含まれている場合に例外をスローします。

```
spark.sql("create table testTable (firstName STRING)")
```

推奨される解決策 A - AWS Glue を使用して、使用しているデータベースに場所を追加します。

AWS Glue データベースに場所を追加するには

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。

2. ナビゲーションペインで、データベースを選択します。
3. データベースのリストで、編集するデータベースを選択します。
4. データベースの詳細ページで、[Edit] (編集) を選択します。
5. [Update a database] (データベースの更新) ページの [Location] (場所) に Amazon S3 の場所を入力します。
6. [Update Database] (データベースの更新) を選択します。

推奨される解決策 B - Amazon S3 に既存の有効な場所がある別の AWS Glue データベースを使用します。例えば、`dbWithLocation` という名前のデータベースがある場合は、コマンド `spark.sql("use dbWithLocation")` を使用してそのデータベースに切り替えます。

推奨される解決法 C - Spark SQL を使用してテーブルを作成する場合、次の例のように `location` の値を指定します。

```
spark.sql("create table testTable (firstName STRING)
       location 's3://DOC-EXAMPLE-BUCKET/'").
```

推奨される解決策 D - テーブルを作成したときに場所を指定しても問題が解決しない場合は、指定する Amazon S3 パスの末尾にスラッシュが付いていることを確認してください。例えば、次のコマンドは不正な引数の例外をスローします。

```
spark.sql("create table testTable (firstName STRING)
       location 's3://DOC-EXAMPLE-BUCKET'")
```

これを修正するには、場所の末尾にスラッシュを追加します (例: `'s3:// DOC-EXAMPLE-BUCKET/'`)。

ワークグループの場所に作成されたデータベース

データベースを作成するために、`spark.sql('create database db')` などのコマンドを使用してデータベースの場所を指定しない場合、Athena はワークグループの場所にサブディレクトリを作成し、その場所を新しく作成されたデータベースに使用します。

AWS Glue デフォルトデータベースの Hive 管理対象テーブルに関する問題

AWS Glue のデフォルトデータベースの `Location` プロパティが空でなく、Amazon S3 内の有効な場所を指定している場合、Athena for Spark を使用して AWS Glue デフォルトデータベースに

Hive 管理対象テーブルを作成すると、データは AWS Glue データベースで指定された場所ではなく、Athena Spark ワークグループで指定された Amazon S3 がある場所に書き込まれます。

この問題は、Apache Hive がデフォルトデータベースを処理する方法に起因しています。Apache Hive は、Hive ウェアハウスのルートロケーションにテーブルデータを作成します。このルートロケーションは、実際のデフォルトのデータベースロケーションとは異なる場合があります。

Athena for Spark を使用して AWS Glue のデフォルトデータベース内に Hive 管理対象テーブルを作成すると、AWS Glue テーブルのメタデータが 2 つの異なる場所を指すことがあります。これにより、`INSERT` または `DROP TABLE` 操作を試みたときに、予期しない動作が発生する可能性があります。

問題を再現する手順は次のとおりです。

1. Athena for Spark では、次の方法のうちの 1 つを選択して、Hive 管理対象テーブルを作成または保存します。
   - `CREATE TABLE $tableName` などの SQL ステートメント
   - Dataframe API で `path` オプションを指定しない、`df.write.mode("overwrite").saveAsTable($tableName)` などの PySpark コマンド

   この時点で、AWS Glue コンソールに Amazon S3 のテーブルの場所が正しく表示されない場合があります。
2. Athena for Spark では、`DROP TABLE $table_name` ステートメントを使用して作成したテーブルをドロップします。
3. `DROP TABLE` ステートメントを実行すると、Amazon S3 下にあるファイルが依然として存在していることがわかります。

この問題を解決するには、次のいずれかを実行します。

解決策 A — Hive 管理対象テーブルを作成するときには、別の AWS Glue データベースを使用します。

解決策 B — AWS Glue のデフォルトデータベースに空の場所を指定します。次に、デフォルトのデータベースに管理対象テーブルを作成します。

Athena for Spark と Athena SQL 間の CSV ファイル形式と JSON ファイル形式の非互換性

オープンソースである Spark に存在する既知の問題により、Athena for Spark で CSV データまたは JSON データのテーブルを作成すると、そのテーブルが Athena SQL からテーブルを読み取ることができない可能性があり、その逆の可能性もあります。

例えば、以下のうちのいずれかの方法で Athena for Spark にテーブルを作成したとします。

- 次の `USING csv` 構文を使用する:

```
spark.sql('''CREATE EXTERNAL TABLE $tableName (
$colName1 $colType1,
$colName2 $colType2,
$colName3 $colType3)
USING csv
PARTITIONED BY ($colName1)
LOCATION $s3_location''')
```

- 次の DataFrame API 構文を使用する:

```
df.write.format('csv').saveAsTable($table_name)
```

オープンソースである Spark には既知の問題があるため、結果テーブルに対して Athena SQL からクエリした場合に成功しない可能性があります。

推奨される解決策 — Apache Hive 構文を使用して Athena for Spark でテーブルを作成してみてください。詳細については、Apache Hive ドキュメントの「HIVEFORMAT テーブルを作成する」を参照してください。

Athena for Spark のトラブルシューティング

次の情報を使用して、Athena でノートブックとセッションを使用するときに発生する可能性のある問題をトラブルシューティングします。

トピック

- Spark 対応ワークグループのトラブルシューティング
- Spark EXPLAIN ステートメントを使用して Spark SQL をトラブルシューティングする

- Athena での Spark アプリケーションイベントのログ
- CloudTrail を使用して Athena ノートブック API 呼び出しをトラブルシューティングする
- 68,000 文字というコードブロックサイズの制限を克服する
- セッションのトラブルシューティング
- テーブルのトラブルシューティング
- サポート情報

Spark 対応ワークグループのトラブルシューティング

次の情報を使用して、Athena の Spark 対応ワークグループのトラブルシューティングを行います。

既存の IAM ロールを使用すると、セッションは応答を停止する

Spark 対応のワークグループ用に新しい AWSAthenaSparkExecutionRole を作成せず、代わりに既存の IAM ロールを更新または選択した場合、セッションが応答しなくなる可能性があります。この場合、Spark 対応のワークグループ実行ロールに次の信頼ポリシーとアクセス許可ポリシーを追加する必要がある可能性があります。

以下の信頼ポリシーの例を追加します。このポリシーには、実行ロールに混同される代理チェックが含まれています。*111122223333*、*aws-region*、および *workgroup-name* の値を、AWS アカウント ID、AWS リージョン、使用しているワークグループに置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "athena.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:athena:aws-
region:111122223333:workgroup/workgroup-name"
                }
            }
```

```
        }
    ]
}
```

ノートブック対応のワークグループには、次のデフォルトポリシーのようなアクセス許可ポリシーを追加します。プレースホルダーの Amazon S3 ロケーションと AWS アカウント ID を使用しているロケーションに対応するように変更します。DOC-EXAMPLE-BUCKET、*aws-region*、*111122223333*、および *workgroup-name* の値を、Amazon S3 バケット、AWS リージョン、AWS アカウント ID および使用しているワークグループに置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:CreatePresignedNotebookUrl",
                "athena:TerminateSession",
                "athena:GetSession",
                "athena:GetSessionStatus",
                "athena:ListSessions",
                "athena:StartCalculationExecution",
                "athena:GetCalculationExecutionCode",
                "athena:StopCalculationExecution",
                "athena:ListCalculationExecutions",
                "athena:GetCalculationExecution",
                "athena:GetCalculationExecutionStatus",
                "athena:ListExecutors",
                "athena:ExportNotebook",
```

```
                "athena:UpdateNotebook"
            ],
            "Resource": "arn:aws:athena:aws-region:111122223333:workgroup/workgroup-
name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:aws-region:111122223333:log-group:/aws-athena:*",
                "arn:aws:logs:aws-region:111122223333:log-group:/aws-athena*:log-
stream:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "logs:DescribeLogGroups",
            "Resource": "arn:aws:logs:aws-region:111122223333:log-group:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "AmazonAthenaForApacheSpark"
                }
            }
        }
    ]
}
```

Spark EXPLAIN ステートメントを使用して Spark SQL をトラブルシューティングする

Spark SQL により Spark EXPLAIN ステートメントを使用して、Spark コードのトラブルシューティングを行うことができます。次のコードと出力例は、この使用量を示しています。

Example - Spark SELECT ステートメント

```
spark.sql("select * from select_taxi_table").explain(True)
```

出力

```
Calculation started (calculation_id=20c1ebd0-1ccf-ef14-db35-7c1844876a7e) in
(session=24c1ebcb-57a8-861e-1023-736f5ae55386).
Checking calculation status...

Calculation completed.
== Parsed Logical Plan ==
'Project [*]
+- 'UnresolvedRelation [select_taxi_table], [], false

== Analyzed Logical Plan ==
VendorID: bigint, passenger_count: bigint, count: bigint
Project [VendorID#202L, passenger_count#203L, count#204L]
+- SubqueryAlias spark_catalog.spark_demo_database.select_taxi_table
   +- Relation spark_demo_database.select_taxi_table[VendorID#202L,
       passenger_count#203L,count#204L] csv

== Optimized Logical Plan ==
Relation spark_demo_database.select_taxi_table[VendorID#202L,
passenger_count#203L,count#204L] csv

== Physical Plan ==
FileScan csv spark_demo_database.select_taxi_table[VendorID#202L,
passenger_count#203L,count#204L]
Batched: false, DataFilters: [], Format: CSV,
Location: InMemoryFileIndex(1 paths)
[s3://DOC-EXAMPLE-BUCKET/select_taxi],
PartitionFilters: [], PushedFilters: [],
ReadSchema: struct<VendorID:bigint,passenger_count:bigint,count:bigint>
```

Example - Spark データフレーム

次の例では、Spark データフレームで EXPLAIN を使用する方法を示しています。

```
taxi1_df=taxi_df.groupBy("VendorID", "passenger_count").count()
taxi1_df.explain("extended")
```

出力

```
Calculation started (calculation_id=d2c1ebd1-f9f0-db25-8477-3effc001b309) in
(session=24c1ebcb-57a8-861e-1023-736f5ae55386).
Checking calculation status...

Calculation completed.
== Parsed Logical Plan ==
'Aggregate ['VendorID, 'passenger_count],
['VendorID, 'passenger_count, count(1) AS count#321L]
+- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,
extra#60,mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Analyzed Logical Plan ==
VendorID: bigint, passenger_count: bigint, count: bigint
Aggregate [VendorID#49L, passenger_count#52L],
[VendorID#49L, passenger_count#52L, count(1) AS count#321L]
+- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,extra#60,
mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Optimized Logical Plan ==
Aggregate [VendorID#49L, passenger_count#52L],
[VendorID#49L, passenger_count#52L, count(1) AS count#321L]
+- Project [VendorID#49L, passenger_count#52L]
   +- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,extra#60,
mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[VendorID#49L, passenger_count#52L], functions=[count(1)],
output=[VendorID#49L, passenger_count#52L, count#321L])
   +- Exchange hashpartitioning(VendorID#49L, passenger_count#52L, 1000),
      ENSURE_REQUIREMENTS, [id=#531]
      +- HashAggregate(keys=[VendorID#49L, passenger_count#52L],
         functions=[partial_count(1)], output=[VendorID#49L,
         passenger_count#52L, count#326L])
         +- FileScan parquet [VendorID#49L,passenger_count#52L] Batched: true,
            DataFilters: [], Format: Parquet,
            Location: InMemoryFileIndex(1 paths)[s3://DOC-EXAMPLE-BUCKET/
            notebooks/yellow_tripdata_2016-01.parquet], PartitionFilters: [],
            PushedFilters: [],
            ReadSchema: struct<VendorID:bigint,passenger_count:bigint>
```

Athena での Spark アプリケーションイベントのログ

Athena ノートブックエディタでは、Jupyter、Spark、Python の標準ログが可能です。`df.show()` を使用して PySpark DataFrame のコンテンツを表示したり、`print("Output")` を使用してセル出力に値を表示したりできます。計算の `stdout`、`stderr`、および `results` 出力は Amazon S3 のクエリ結果バケットの場所に書き込まれます。

Spark アプリケーションイベントの Amazon CloudWatch へのログ

Athena セッションでは、使用しているアカウントの Amazon CloudWatch にログを書き込むこともできます。

ログストリームとロググループの詳細

CloudWatch は、ログアクティビティをログストリームとロググループに整理します。

ログストリーム - CloudWatch ログストリーミングは、同じ出典を共有する一連のログイベントです。CloudWatch Logs でのログの各ソースで各ログストリームが構成されます。

ロググループ - CloudWatch ログで、ロググループは保持、モニタリング、アクセス制御について同じ設定を共有するログストリーミングのグループです。

1 つのロググループに属することができるログストリーミングの数に制限はありません。

Athena で初めてノートブックセッションを開始すると、次の例のように、Athena は Spark 対応のワークグループの名前を使用して CloudWatch にロググループを作成します。

```
/aws-athena/workgroup-name
```

このロググループは、少なくとも 1 つのログイベントを生成するセッションのエグゼキューターごとに 1 つのログストリームを受け取ります。エグゼキューターとは、ノートブックセッションが Athena にリクエストできる最小の計算単位です。CloudWatch では、ログストリームの名前はセッション ID とエグゼキューター ID で始まります。

CloudWatch ロググループとログストリームの詳細については、Amazon CloudWatch Logs ユーザーガイドの「ロググループとログストリームの操作」を参照してください。

Athena for Spark の標準ロガーオブジェクトを使用する

Athena for Spark セッションでは、次の 2 つのグローバルスタンダードロガーオブジェクトを使用して、Amazon CloudWatch にログを書き込むことができます。

- athena_user_logger - ログを CloudWatch にのみ送信します。このオブジェクトは、次の例のように、Spark アプリケーションの情報を CloudWatch に直接ログ記録する場合に使用します。

  ```
  athena_user_logger.info("CloudWatch log line.")
  ```

  この例では、次のようにログイベントを CloudWatch に書き込みます。

  ```
  AthenaForApacheSpark: 2022-01-01 12:00:00,000 INFO builtins: CloudWatch log line.
  ```

- athena_shared_logger - 同じログをサポート目的で CloudWatch および AWS と両方に送信します。このオブジェクトを使用して、次の例のように、トラブルシューティングのために AWS サービスチームとログを共有できます。

  ```
  athena_shared_logger.info("Customer debug line.")
  var = [...some variable holding customer data...]
  athena_shared_logger.info(var)
  ```

  この例では、debug の行と `var` の変数の値を CloudWatch Logs に記録し、各行のコピーを AWS Support に送信します。

> **Note**
>
> プライバシー保護のため、計算コードと結果は AWS と共有されません。AWS Support に表示する情報のみを書き込む `athena_shared_logger` への呼び出しを行ってください。

提供されているロガーは、Apache Log4j を介してイベントを書き込み、このインターフェイスのログレベルを継承します。指定できるログレベル値は `DEBUG`、`ERROR`、`FATAL`、`INFO`、`WARN`、`WARNING` です。ロガーで対応する名前付き関数を使用して、これらの値を生成できます。

> **Note**
>
> 名前 `athena_user_logger` または `athena_shared_logger` を再バインドしないでください。これを行うと、ログオブジェクトはセッションの残りの部分で CloudWatch に書き込むことができなくなります。

例: ノートブックイベントを CloudWatch にログする

次の手順は、Athena ノートブックイベントを Amazon CloudWatch Logs にログ記録する方法を示しています。

Athena ノートブックイベントを Amazon CloudWatch Logs にログ記録するには

1. Amazon Athena で Apache Spark を開始 に従って、一意の名前で Spark 対応のワークグループを Athena に作成します。このチュートリアルでは、ワークグループ名 `athena-spark-example` を使用します。
2. 独自のノートブックの作成 の手順に従ってノートブックを作成し、新しいセッションを開始します。
3. Athena ノートブックエディタの新しいノートブックセルに、次のコマンドを入力します。

```
athena_user_logger.info("Hello world.")
```

4. セルを実行します。
5. 次のいずれかを実行して、現在のセッション ID を取得します。

- セル出力 (例: `... session=72c24e73-2c24-8b22-14bd-443bdcd72de4`) を表示します。
- 新しいセルで、マジックコマンド `%session_id` を実行します。

6. セッション ID を保存します。
7. ノートブックセッションの実行に使用しているのと同じ AWS アカウント で、https://console.aws.amazon.com/cloudwatch/ にある CloudWatch コンソールを開きます。
8. CloudWatch コンソールのナビゲーションペインで、[Log groups] (ロググループ) を選択します。
9. ロググループのリストで、次の例のように Spark 対応の Athena ワークグループの名前を持つロググループを選択します。

```
/aws-athena/athena-spark-example
```

[Log streams] (ログストリーム) セクションには、ワークグループの 1 つ以上のログストリームリンクのリストが含まれています。各ログストリーム名には、セッション ID、エグゼキューター ID、および一意の UUID がスラッシュ文字によって区切られています。

例えば、セッション ID が `5ac22d11-9fd8-ded7-6542-0412133d3177` であり、エグゼキューター ID が `f8c22d11-9fd8-ab13-8aba-c4100bfba7e2` の場合、ログストリームの名前は次の例のようになります。

```
5ac22d11-9fd8-ded7-6542-0412133d3177/f8c22d11-9fd8-ab13-8aba-c4100bfba7e2/f012d7cb-cefd-40b1-90b9-67358f003d0b
```

10. セッションのログストリームのログストリームを選択します。
11. [Log events] (イベントのログ) ページで、[Message] (メッセージ) 列を表示します。

実行したセルのログイベントは次のようになります。

```
AthenaForApacheSpark: 2022-01-01 12:00:00,000 INFO builtins: Hello world.
```

12. Athena ノートブックエディタに戻ります。
13. 新しいセルに、次のコードを入力します。このコードは変数を CloudWatch にログします。

```
x = 6
athena_user_logger.warn(x)
```

14. セルを実行します。
15. 同じログストリームの CloudWatch コンソールの [Log events] (ログイベント) ページに戻ります。
16. ログストリームには、次のようなメッセージを含むログイベントエントリが含まれるようになりました。

```
AthenaForApacheSpark: 2022-01-01 12:00:00,000 WARN builtins: 6
```

CloudTrail を使用して Athena ノートブック API 呼び出しをトラブルシューティングする

ノートブック API 呼び出しのトラブルシューティングを行うため、Athena CloudTrail のログを調べて異常を調査したり、ユーザーが開始したアクションを発見したりできます。Athena での CloudTrail の使用についての詳細は、「AWS CloudTrail を使用した Amazon Athena API コールのログ記録」を参照してください。

次の例は、Athena ノートブック API に関する CloudTrail ログエントリを示しています。

- StartSession
- TerminateSession
- ImportNotebook
- UpdateNotebook
- StartCalculationExecution

StartSession

次の例は、ノートブックの StartSession イベントの CloudTrail ログを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
```

```
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:05:36Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "StartSession",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.10",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "workGroup": "notebook-workgroup",
        "engineConfiguration": {
            "coordinatorDpuSize": 1,
            "maxConcurrentDpus": 20,
            "defaultExecutorDpuSize": 1,
            "additionalConfigs": {
                "NotebookId": "b8f5854b-1042-4b90-9d82-51d3c2fd5c04",
                "NotebookIframeParentUrl": "https://us-east-1.console.aws.amazon.com"
            }
        },
        "notebookVersion": "KeplerJupyter-1.x",
        "sessionIdleTimeoutInMinutes": 20,
        "clientRequestToken": "d646ff46-32d2-42f0-94d1-d060ec3e5d78"
    },
    "responseElements": {
        "sessionId": "a2c1ebba-ad01-865f-ed2d-a142b7451f7e",
        "state": "CREATED"
    },
    "requestID": "d646ff46-32d2-42f0-94d1-d060ec3e5d78",
    "eventID": "b58ce998-eb89-43e9-8d67-d3d8e30561c9",
    "readOnly": false,
    "eventType": "AwsApiCall",
```

```
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

TerminateSession

次の例は、ノートブックの TerminateSession イベントの CloudTrail ログを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:21:03Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "TerminateSession",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.11",
```

```
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "sessionId": "a2c1ebba-ad01-865f-ed2d-a142b7451f7e"
    },
    "responseElements": {
        "state": "TERMINATING"
    },
    "requestID": "438ea37e-b704-4cb3-9a76-391997cf42ee",
    "eventID": "49026c5a-bf58-4cdb-86ca-978e711ad238",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

ImportNotebook

次の例は、ノートブック ImportNotebook イベントの CloudTrail ログを示しています。セキュリティ上、一部のコンテンツは非表示になっています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
```

```
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:08:54Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "ImportNotebook",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "workGroup": "notebook-workgroup",
        "name": "example-notebook-name",
        "payload": "HIDDEN_FOR_SECURITY_REASONS",
        "type": "IPYNB",
        "contentMD5": "HIDDEN_FOR_SECURITY_REASONS"
    },
    "responseElements": {
        "notebookId": "05f6225d-bdcc-4935-bc25-a8e19434652d"
    },
    "requestID": "813e777f-6dac-41f4-82a7-e99b7b33f319",
    "eventID": "4abec837-143b-4458-9c1f-fa9fb88ab69b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

UpdateNotebook

次の例は、ノートブック UpdateNotebook イベントの CloudTrail ログを示しています。セキュリティ上、一部のコンテンツは非表示になっています。


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:AthenaExecutor-9cc1ebb2-aac5-
b1ca-8247-5d827bd8232f",
        "arn": "arn:aws:sts::123456789012:assumed-role/AWSAthenaSparkExecutionRole-
om0yj71w5l/AthenaExecutor-9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-om0yj71w5l",
                "accountId": "123456789012",
                "userName": "AWSAthenaSparkExecutionRole-om0yj71w5l"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:48:06Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T16:52:22Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "UpdateNotebook",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.13",
    "userAgent": "Boto3/1.24.84 Python/3.8.14 Linux/4.14.225-175.364.amzn2.aarch64
 Botocore/1.27.84",
    "requestParameters": {
        "notebookId": "c87553ff-e740-44b5-884f-a70e575e08b9",
        "payload": "HIDDEN_FOR_SECURITY_REASONS",
        "type": "IPYNB",
        "contentMD5": "HIDDEN_FOR_SECURITY_REASONS",
```

```
        "sessionId": "9cc1ebb2-aac5-b1ca-8247-5d827bd8232f"
    },
    "responseElements": null,
    "requestID": "baaba1d2-f73d-4df1-a82b-71501e7374f1",
    "eventID": "745cdd6f-645d-4250-8831-d0ffd2fe3847",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    }
}
```

StartCalculationExecution

次の例は、StartCalculationExecution の CloudTrail ログを示しています。セキュリティ上、一部のコンテンツは非表示になっています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:AthenaExecutor-9cc1ebb2-aac5-
b1ca-8247-5d827bd8232f",
        "arn": "arn:aws:sts::123456789012:assumed-role/AWSAthenaSparkExecutionRole-
om0yj71w5l/AthenaExecutor-9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-om0yj71w5l",
                "accountId": "123456789012",
                "userName": "AWSAthenaSparkExecutionRole-om0yj71w5l"
            },
            "webIdFederationData": {},
            "attributes": {
```

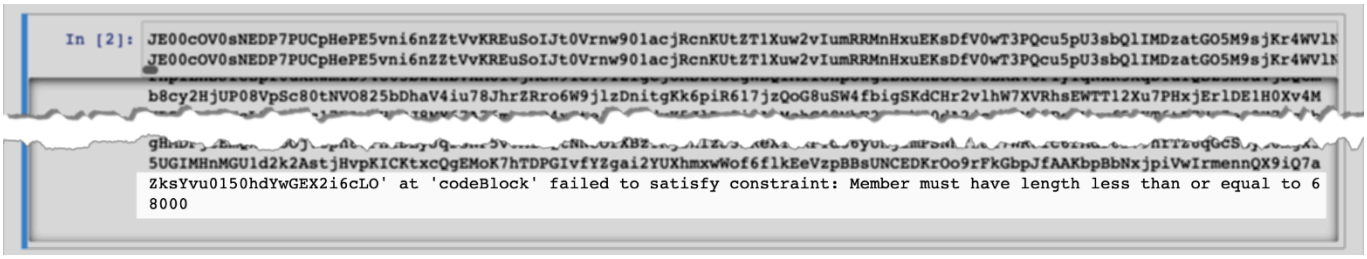
```
                "creationDate": "2022-10-14T16:48:06Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T16:52:37Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "StartCalculationExecution",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.14",
    "userAgent": "Boto3/1.24.84 Python/3.8.14 Linux/4.14.225-175.364.amzn2.aarch64
 Botocore/1.27.84",
    "requestParameters": {
        "sessionId": "9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
        "description": "Calculation started via Jupyter notebook",
        "codeBlock": "HIDDEN_FOR_SECURITY_REASONS",
        "clientRequestToken": "0111cd63-4fd0-4ad8-a738-fd350115fc21"
    },
    "responseElements": {
        "calculationExecutionId": "82c1ebb4-bd08-e4c3-5631-a662fb2ff2c5",
        "state": "CREATING"
    },
    "requestID": "1a107461-3f1b-481e-b8a2-7fbd524e2373",
    "eventID": "b74dbd00-e839-4bd1-a1da-b75fbc70ab9a",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    }
}
```

68,000 文字というコードブロックサイズの制限を克服する

Athena for Spark には、計算コードのブロックサイズの上限が 68,000 文字という既知の制限があります。この制限を超えるコードブロックで計算を実行すると、次のエラーメッセージが表示されることがあります。

「codeBlock」の「...」は制約を満たすことができませんでした。メンバーの長さは 68,000 以下である必要があります

このエラーは、Athena コンソールのノートブックエディターで以下の画像のように表示されます。



次の例のように、AWS CLI を使用して大きなコードブロックが含まれる計算を実行した場合にも、同じエラーが発生する場合があります。

```
aws athena start-calculation-execution \
    --session-id "{SESSION_ID}" \
    --description "{SESSION_DESCRIPTION}" \
    --code-block "{LARGE_CODE_BLOCK}"
```

このコマンドでは、次のエラーメッセージが表示されます。

{LARGE_CODE_BLOCK}「codeBlock」の「...」は制約を満たすことができませんでした。メンバーの長さは 68,000 以下である必要があります

回避方法

この問題を回避するには、クエリや計算コードが含まれるファイルを Amazon S3 にアップロードします。次に、boto3 を使用してファイルを読み取り、SQL またはコードを実行します。

以下の例では、SQL クエリまたは Python コードが含まれるファイルが Amazon S3 にすでにアップロードされていることを前提としています。

SQL 例

以下のコード例では、Amazon S3 バケットから `large_sql_query.sql` ファイルを読み取ってから、ファイルに含まれている大きなクエリを実行しています。

```
s3 = boto3.resource('s3')
def read_s3_content(bucket_name, key):
    response = s3.Object(bucket_name, key).get()
    return response['Body'].read()
```

```
# SQL
sql = read_s3_content('bucket_name', 'large_sql_query.sql')
df = spark.sql(sql)
```

PySpark 例

以下のコード例では、Amazon S3 から large_py_spark.py ファイルを読み取ってから、ファイルに含まれている大きなコードブロックを実行しています。

```
s3 = boto3.resource('s3')

def read_s3_content(bucket_name, key):
    response = s3.Object(bucket_name, key).get()
    return response['Body'].read()

# PySpark
py_spark_code = read_s3_content('bucket_name', 'large_py_spark.py')
exec(py_spark_code)
```

セッションのトラブルシューティング

このトピックの情報を使用して、セッションの問題をトラブルシューティングします。

異常状態のセッション

「Session in unhealthy state」(セッションが異常な状態です) というエラーメッセージが表示された場合。「Please create a new session」(新しいセッションを作成してください)、既存のセッションを終了してから、新しいセッションを作成してください。

ノートブックサーバーへの接続を確立できませんでした

ノートブックを開くと、次のエラーメッセージが表示されることがあります。

```
A connection to the notebook server could not be established.
The notebook will continue trying to reconnect.
Check your network connection or notebook server configuration.
```

原因

Athena がノートブックを開くと、Athena はセッションを作成し、署名済みのノートブックの URL を使用してノートブックに接続します。ノートブックへの接続には、WSS (WebSocket セキュア) プロトコルを使用します。

エラーは、次の原因で発生することがあります。

- ローカルファイアウォール (会社全体のファイアウォールなど) が WSS トラフィックをブロックしています。
- ローカルコンピュータ上のプロキシまたはアンチウイルスソフトウェアが WSS 接続をブロックしています。

ソリューション

us-east-1 リージョンに次のような WSS 接続があるとします。

```
wss://94c2bcdf-66f9-4d17-9da6-7e7338060183.analytics-gateway.us-east-1.amazonaws.com/
api/kernels/33c78c82-b8d2-4631-bd22-1565dc6ec152/channels?session_id=
7f96a3a048ab4917b6376895ea8d7535
```

エラーを解決するには、次の戦略のいずれかを使用します。

- ワイルドカードパターン構文を使用すると、AWS リージョン とAWS アカウント を経由するポート 443 の WSS トラフィックを一覧表示できます。

  ```
  wss://*amazonaws.com
  ```

- ワイルドカードパターン構文を使用すると、指定した AWS リージョン で 1 つの AWS リージョン と AWS アカウント を経由するポート 443 で WSS トラフィックを一覧表示できます。次の例では us-east-1 を使用しています。

  ```
  wss://*analytics-gateway.us-east-1.amazonaws.com
  ```

テーブルのトラブルシューティング

テーブルの作成時にパスエラーを作成できない

エラーメッセージ:「IllegalArgumentException: Cannot create a path from an empty string.」(IllegalArgumentException: 空の文字列からパスを作成することはできません)

原因: このエラーは、Athena の Apache Spark を使用してデータベースにテーブルを作成したときに、AWS Glue データベースのプロパティが空の LOCATION である場合に発生する可能性があります。

推奨される解決策: 詳細と解決策については、「テーブル作成時の不正な引数の例外」を参照してください。

AWS Glue テーブルをクエリする場合の AccessDeniedException

エラーメッセージ:「pyspark.sql.utils.AnalysisException: Unable to verify existence of default database: com.amazonaws.services.glue.model.AccessDeniedException: User: arn:aws:sts::*aws-account-id*:assumed-role/AWSAthenaSparkExecutionRole-*unique-identifier*/AthenaExecutor-*unique-identifier* is not authorized to perform: glue:GetDatabase on resource: arn:aws:glue:*aws-region*:*aws-account-id*:catalog because no identity-based policy allows the glue:GetDatabase action (Service: AWSGlue; Status Code: 400; Error Code: AccessDeniedException; Request ID: *request-id*; Proxy: null)」(pyspark.sql.utils.AnalysisException: 次のデフォルトデータベースの存在を確認できません: com.amazonaws.services.glue.model.AccessDeniedException: ユーザー: arn:aws:sts::aws-account-id:assumed-role/AWSAthenaSparkExecutionRole-unique-identifier/AthenaExecutor-unique-identifier は実行を許可されていません: リソース上の glue:GetDatabase: arn:aws:glue:aws-region:aws-account-id:catalog ID ベースのポリシーでは glue:GetDatabase アクションが許可されていないため (サービス: AWSGlue、ステータスコード: 400、エラーコード: AccessDeniedException、リクエスト ID: request-id、プロキシ: null))

原因: Spark 対応ワークグループの実行ロールに、AWS Glue リソースにアクセスする許可がありません。

推奨される解決策: この問題を解決するには、実行ロールに AWS Glue リソースへのアクセスを許可し、Amazon S3 バケットポリシーを編集して実行ロールへのアクセスを許可します。

以下の手順では、これらのステップがさらに詳しく説明されています。

実行ロールに AWS Glue リソースへのアクセス権を付与するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. コンソールのナビゲーションペインが表示されない場合は、左側の展開メニューをクリックします。

3. Athena コンソールのナビゲーションペインで、[Workgroups] (ワークグループ) をクリックします。
4. [Workgroups] (ワークグループ) ページで、表示するワークグループのリンクを選択します。
5. ワークグループの [Overview Details] (概要詳細) ページで、[Role ARN] (ロール ARN) リンクを選択します。このリンクは、IAM コンソールで Spark 実行ロールを開きます。
6. [Permissions policies] (アクセス許可ポリシー) セクションで、リンクされたロールポリシー名を選択します。
7. [Edit policy] (ポリシーの編集)、[JSON] の順に選択します。
8. ロールに AWS Glue アクセス許可を追加します。通常は、`glue:GetDatabase` および `glue:GetTable` アクションに対してアクセス許可を追加します。IAM ロールの設定の詳細については、IAM ユーザーガイドの「IAM ID アクセス許可の追加と削除」を参照してください。
9. [ポリシーの確認] を選択し、[変更の保存] を選択します。
10. Amazon S3 バケットポリシーを編集して、実行ロールへのアクセスを許可します。ロールには、バケットおよびバケット内のオブジェクトの両方へのアクセスを許可する必要があることに注意してください。手順については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 コンソールを使用してバケットポリシーを追加する」を参照してください。

サポート情報

AWS からの支援については、AWS Management Console から [Support] (サポート)、[Support Center] (サポートセンター) の順に選択します。体験を円滑に進めるため、次の情報をご用意ください。

- Athena クエリ ID
- セッション ID
- 計算 ID

リリースノート

Amazon Athena の特徴、改良点、およびバグ修正についてリリース日順に説明します。

トピック

Athena リリースノート (2024 年)

2024 年 4 月 26 日

公開日: 2024 年 4 月 26 日

Athena が JDBC ドライバーバージョン 3.2.0 をリリースしました。このバージョンのドライバーの詳細については、「Amazon Athena ODBC 3.x リリースノート」を参照してください。JDBC 3.x ドライバーをダウンロードするには、「JDBC 3.x ドライバーのダウンロード」を参照してください。

2024 年 4 月 24 日

公開日: 2024 年 4 月 24 日

Athena は、次の修正と改善を発表しました。

- Parquet – Athena は、リストやマップグループに含まれていない、注釈のない繰り返しプリミティブフィールドに対して、Parquet での下位互換性のある読み取りをサポートするようになりました。この変更により、正しくない結果がそのまま返されることがなくなり、スキーマの不一致に対するエラーメッセージが改善されました。

詳細については、GitHub.com の Parquet の「Support backwards compatible reads for unannotated repeated primitive fields in Parquet」(注釈のない繰り返しプリミティブフィールドに対する下位互換性のある読み取りのサポート) を参照してください。

- Iceberg OPTIMIZE – 非パーティションキーフィルターが WHERE 句で使用されたときにデータが失われる原因となる OPTIMIZE クエリの問題を解決しました。詳細については、「OPTIMIZE」を参照してください。

2024 年 4 月 16 日

公開日: 2024 年 4 月 16 日

新しい Amazon Athena フェデレーテッドクエリパススルー機能を使用して、クエリの全体を基盤となるデータソースで直接実行します。フェデレーテッドパススルークエリは、元のデータソース固有の関数、クエリ言語、パフォーマンス機能を活用できるようにします。たとえば、PartiQL 言語を使用して、DynamoDB で Athena クエリを実行することができます。フェデレーテッドパススルークエリは、Athena では使用できないデータソースの関数を集約、結合、または呼び出す SELECT クエリを実行する場合にも役立ちます。パススルークエリを使用することで Athena が処理するデータの量が低減するため、クエリ時間の短縮につながります。

詳細については、「フェデレーテッドパススルークエリの実行」を参照してください。現在使用しているコネクタを最新バージョンにアップグレードするには、「データソースコネクタの更新」を参照してください。

2024 年 4 月 10 日

公開日: 2024 年 4 月 10 日

Athena は、次の機能と改善点を発表しました。

ODBC 1.2.3.1000 ドライバー

Athena 用の ODBC 1.2.3.1000 ドライバーがリリースされました。

解決された問題:

- プロキシサーバーの接続問題 – ルート証明書なしでプロキシサーバーを使用すると、コネクタが接続の確立に失敗していました。

ODBC 1.x ドライバー、リリースノート、ドキュメントのダウンロード、および詳細については、「Athena ODBC 1.x ドライバー」を参照してください。

JDBC 2.1.5 ドライバー

Athena 用の JBDC 2.1.5 ドライバーがリリースされました。

更新と機能拡張:

- バージョン 1.12.687 を使用するように AWS Java SDK を更新しました。
- バージョン 2.16.0 を使用するように Jackson ライブラリを更新しました。
- バージョン 1.3.14 を使用するように Logback ライブラリを更新しました。

JDBC 2.x ドライバー、リリースノート、ドキュメントのダウンロード、および詳細については、「Athena JDBC 2.x ドライバー」を参照してください。

2024 年 4 月 8 日

公開日: 2024 年 4 月 8 日

Athena が ODBC ドライバーバージョン 2.0.3.0. を発表しました。詳細については、2.0.3.0 のリリースノートを参照してください。新しい ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

2024 年 3 月 15 日

公開日: 2024 年 3 月 18 日

Amazon Athena が、カナダ西部 (カルガリー) リージョンでの Athena SQL の提供開始を発表しました。

各 AWS リージョン で利用できる AWS のサービス の一覧については、「地域別の AWS サービス」を参照してください。

2024 年 2 月 15 日

公開日: 2024 年 2 月 15 日

Athena が JDBC ドライバーバージョン 3.1.0 をリリースしました。

Amazon Athena JDBC ドライバーバージョン 3.1.0 は、Microsoft Active Directory Federation Services (AD FS) Windows 統合認証と、フォームベース認証のサポートを追加します。3.1.0 リリースには、その他の軽微な改善とバグ修正も含まれています。

JDBC v3 ドライバーをダウンロードするには、「JDBC 3.x ドライバーのダウンロード」を参照してください。

2024 年 1 月 31 日

公開日: 2024 年 1 月 31 日

Athena は、次の機能と改善点を発表しました。

- Hudi アップグレード – Athena SQL を使用して Hudi 0.14.0 テーブルをクエリできるようになりました。HUDI テーブルをクエリするための Athena SQL の使用については、「Athena を使用した Apache Hudi データセットのクエリ」を参照してください。

Athena リリースノート (2023 年)

2023 年 12 月 14 日

公開日: 2023 年 12 月 14 日

Athena は、次の修正と改善を発表しました。

Athena は、JDBC ドライバーのバージョン 2.1.3 をリリースしました。ドライバーは次の問題を解決します。

- Spring Boot や Gradle アプリケーションのロギングとの競合を避けるため、ロギングが改善されました。
- `executeBatch()` JDBC メソッドを使用してレコードを挿入する際、ドライバーが誤って 1 つのレコードのみを挿入しました。Athena はクエリのバッチ実行をサポートしていないため、`executeBatch()` 使用時にドライバーがエラーを報告するようになりました。この制限を回避するには、1 つのクエリをループで送信できます。

新しい JDBC ドライバー、リリースノート、およびドキュメントをダウンロードするには、「Athena JDBC 2.x ドライバー」を参照してください。

2023 年 12 月 9 日

公開日: 2023 年 12 月 9 日

Athena 用の ODBC 1.2.1.1000 ドライバーがリリースされました。

特徴と拡張機能:

- RStudio サポートの更新 — ODBC ドライバーが macOS 上の RStudio をサポートするようになりました。
- 単一のカタログとスキーマのサポート — コネクタが 1 つのカタログとスキーマを返すことができるようになりました。詳細については、ダウンロード可能なインストールおよび設定ガイドを参照してください。

解決された問題:

- 準備済みステートメント — 列単位のスキーマを使用するパラメータ配列を含む準備済みステートメントを実行すると、コネクタが誤ったクエリ結果を返しました。
- 列サイズ — `$file_modified_time` システム列を選択すると、コネクタが誤った列サイズを返しました。
- SQLPrepare — `SELECT` クエリに対して `SQLPrepare` に関連するパラメータをバインドすると、コネクタがエラーを返しました。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「Athena ODBC 1.x ドライバー」を参照してください。

2023 年 12 月 7 日

公開日: 2023 年 12 月 7 日

Athena は ODBC ドライバーバージョン 2.0.2.1 を発表しました。詳細については、2.0.2.1 のリリースノートを参照してください。新しい ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

2023 年 12 月 5 日

公開日: 2023 年 12 月 5 日

AWS IAM Identity Center 認証モードを使用する Athena SQL ワークグループを作成できるようになりました。これらのワークグループは IAM アイデンティティセンターの信頼できる ID 伝達機能をサポートしています。信頼できる ID 伝達により、Amazon Athena や Amazon EMR Studio などの AWS 分析サービスでアイデンティティを使用できるようになります。

詳細については、「IAM アイデンティティセンターが有効な Athena ワークグループの使用」を参照してください。

2023 年 11 月 28 日

公開日: 2023 年 11 月 28 日

Amazon S3 Express One Zone ストレージクラス内のデータをクエリして、高速にクエリ結果を得られるようになりました。S3 Express One Zone は、最も頻繁にアクセスされるデータやレイテンシーの影響を受けやすいアプリケーションに、1 桁のミリ秒単位で一貫したデータアクセスを提供することを目的とした、高性能で単一アベイラビリティーゾーンのストレージクラスです。まず、データを S3 Express One Zone ストレージに移動し、Athena でシームレスなクエリを実行できるようにデータを AWS Glue Data Catalog とカタログ化します。

詳細については、「S3 Express One Zone データのクエリ」を参照してください。

2023 年 11 月 27 日

公開日: 2023 年 11 月 27 日

Athena は、次の機能と改善点を発表しました。

- Glue データカタログビュー — Glue データカタログビューは、Amazon Athena や Amazon Redshift などの AWS サービスに共通する 1 つのビューを提供します。Glue データカタログビューでは、アクセス許可はビューをクエリするユーザーではなく、ビューを作成したユーザーによって定義されます。これらのビューを使用することで、アクセスコントロールの強化、完全なレコードの確保、セキュリティの強化、基礎となるテーブルへのアクセスの防止を実現することができます。

  詳細については、「AWS Glue Data Catalog ビューの使用」を参照してください。
- CloudTrail Lake のサポート — Amazon Athena を使用して AWS CloudTrail LakeAWS CloudTrail のデータを分析できるようになりました。Lake は CloudTrail 用のマネージドデータレイクであり、監査、セキュリティ、運用調査のためのアクティビティログの集約、不変的な保存、分析に使用できます。Athena から CloudTrail Lake アクティビティログをクエリするために、データを移動

したり、個別のデータ処理パイプラインを構築したりする必要はありません。ETL オペレーションは必要ありません。

開始するには、CloudTrail Lake でデータフェデレーションを有効にします。CloudTrail Lake イベントデータストアのメタデータを AWS Glue Data Catalog と共有すると、CloudTrail は必要な AWS Glue Data Catalog リソースを作成し、データを AWS Lake Formation に登録します。Lake Formation では、Athena を使用してイベントデータストアをクエリできるユーザーとロールを指定できます。

詳細については、「AWS CloudTrail ユーザーガイド」の「Enable Lake query federation」を参照してください。

2023 年 11 月 17 日

公開日: 2023 年 11 月 17 日

Athena は、次の機能と改善点を発表しました。

機能

- コストベースオプティマイザー — Athena は、AWS Glue の統計を使用したコストベースの最適化の一般提供を発表しました。Athena SQL のクエリを最適化するには、Athena がテーブルを収集するようにリクエストするか、AWS Glue のテーブルの列レベル統計を収集するようにリクエストできます。クエリのすべてのテーブルに統計がある場合、Athena はその統計を使用して代替の実行プランを調べたうえ、最も速いと思われる実行プランを選択します。

  詳細については、「コストベースオプティマイザーの使用」を参照してください。
- Amazon EMR Studio の統合 — Athena コンソールを直接使用することなく、Amazon EMR Studio で Athena を使用できるようになりました。Amazon EMR で Athena 統合を使用すると、次のタスクを実行できます。
  - Athena SQL クエリの実行
  - クエリ結果を表示する
  - クエリ履歴の表示
  - 保存されたクエリの表示
  - パラメータ化されたクエリの実行
  - データカタログのデータベース、テーブル、ビューの表示

詳細については、「AWS のサービス における Athena との統合」トピックの「Amazon EMR Studio」を参照してください。

- ネストされたアクセス制御 — Athena が、ネストされたデータの Lake Formation アクセス制御のサポートを発表しました。Lake Formation では、`struct` データ型を持つネストされた列にデータフィルターを定義して適用できます。データフィルタリングを使用し、ネストされた列のサブ構造へのユーザーアクセスを制限できます。ネストされたデータにデータフィルターを作成する方法の詳細については、「AWS Lake Formation デベロッパーガイド」の「データフィルターの作成」を参照してください。
- プロビジョニングされたキャパシティーの使用状況メトリクス — Athena は、キャパシティ予約の新しい CloudWatch メトリクスを発表しました。新しいメトリクスを使用して、プロビジョニングした DPU の数とクエリによって使用されている DPU 数を管理できます。クエリが終了すると、クエリが消費した DPU 数も表示できます。

  詳細については、「CloudWatch メトリクスによる Athena クエリのモニタリング」を参照してください。

改良点

- エラーメッセージの変更 — `Insufficient Lake Formation permissions` エラーメッセージは `Table not found` または `Schema not found` として表示されるようになりました。この変更は、悪意のある攻撃者がエラーメッセージからテーブルまたはデータベースリソースの存在を推測するのを防ぐために行われました。

2023 年 11 月 16 日

公開日: 2023 年 11 月 16 日

Athena は、互換性のある SQL 開発およびビジネスインテリジェンスアプリケーションからデータへの接続、クエリ、視覚化のエクスペリエンスを向上させる新しい JDBC ドライバーをリリースしました。新しいドライバーは簡単にアップグレードできます。ドライバーは Amazon S3 から直接クエリ結果を読み取り、クエリ結果をより早く利用可能にします。

詳細については、「Athena JDBC 3.x ドライバー」を参照してください。

2023 年 10 月 31 日

公開日: 2023 年 10 月 31 日

Amazon Athena は、プロビジョニングされたキャパシティの 1 時間の予約を発表しました。本日より、1 時間後にプロビジョニングされたキャパシティを予約して解放できます。この変更により、時間とともに需要が変化するワークロードのコスト最適化が容易になります。

プロビジョニングされたキャパシティは Atena の機能であり、最も重要なインタラクティブワークロードの優先、制御、スケーリングを行えるようにするワークロード管理機能 いつでも容量を追加して、同時に実行するクエリの数を増やしたり、そのキャパシティを使用するワークロードを制御したり、ワークロード間でキャパシティを共有したりできます。

詳細については、「クエリ処理キャパシティの管理」を参照してください。料金の詳細については、「Amazon Athena 料金表」ページを参照してください。

2023 年 10 月 25 日

公開日: 2023 年 10 月 26 日

Athena は、次の修正と改善を発表しました。

jackson-core パッケージ — 1,000 文字を超える数値の JSON テキストは失敗するようになりました。この修正は「sonatype-2022-6438」のセキュリティ問題を解決します。

2023 年 10 月 17 日

公開日: 2023 年 10 月 17 日

Athena は ODBC ドライバーバージョン 2.0.2.0 を発表しました。詳細については、2.0.2.0 のリリースノートを参照してください。新しい ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

2023 年 9 月 26 日

公開日: 2023 年 9 月 26 日

Athena は、次の機能と改善点を発表しました。

- Delta Lake テーブルの Lake Formation 読み取りサポート。Athena で Delta Lake テーブルの使用に関する詳細については、「Linux Foundation Delta Lake テーブルへのクエリ」を参照してください。

2022 年 8 月 23 日

公開日: 2023 年 8 月 23 日

Amazon Athena は、イスラエル (テルアビブ) リージョンで Athena SQL が利用可能になったことを発表しました。

各 AWS リージョン で利用できる AWS のサービス の一覧については、「地域別の AWS サービス」を参照してください。

2023 年 8 月 10 日

公開日: 2023 年 8 月 10 日

Athena は、次の修正と改善を発表しました。

ODBC ドライバーバージョン 2.0.1.1

Athena は ODBC ドライバーバージョン 2.0.1.1 を発表しました。詳細については、2.0.1.1 のリリースノートを参照してください。新しい ODBC v2 ドライバーをダウンロードするには、「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

JDBC ドライバーバージョン 2.1.1

Athena は、JDBC ドライバーのバージョン 2.1.1 をリリースしました。ドライバーは次の問題を解決します。

- 正規表現を含むステートメントを使用してテーブルが作成されたときに発生していたエラー。
- `ApplicationName` 接続パラメータが正しく適用されない原因となっていた問題。

新しい JDBC ドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2023 年 7 月 31 日

公開日: 2023 年 7 月 31 日

Amazon Athena は、追加の AWS リージョンで Athena SQL が利用可能になったことを発表しました。

このリリースでは、Athena SQL の提供地域を、アジアパシフィック (ハイデラバード)、アジアパシフィック (メルボルン)、欧州 (スペイン)、欧州 (チューリッヒ) に拡大します。

各 AWS リージョン で利用できる AWS のサービス の一覧については、「地域別の AWS サービス」を参照してください。

2023 年 7 月 27 日

公開日: 2023 年 7 月 27 日

Athena では、Google BigQuery コネクタバージョン 2023.30.1 がリリースされました。このバージョンのコネクタでは、クエリの実行時間が短縮され、BigQuery プライベートエンドポイントに対するクエリのサポートが追加されました。

Google BigQuery コネクタの詳細については、「Amazon Athena Google BigQuery コネクタ」を参照してください。既存のデータソースコネクタの更新については、「データソースコネクタの更新」を参照してください。

2023 年 7 月 24 日

公開日: 2023 年 7 月 24 日

Athena は、次の修正と改善を発表しました。

- UNION クエリ — 特定のUNION クエリのパフォーマンスが向上しました。
- 型比較による結合 — 2 つの異なる型の比較を含む `JOIN` ステートメントの潜在的なクエリの失敗を修正しました。
- ネストされた列のサブクエリ — サブクエリがネストされた列に相関しているときにクエリが失敗する問題を修正しました。
- アイスバーグビュー — Apache Iceberg ビューのタイムスタンプ列の精度に関する互換性の問題を修正しました。タイムスタンプ列を持つ Iceberg ビューが、その列が Athena エンジンバージョン 2 で作成されたか、または Athena エンジンバージョン 3 で作成されたかに関係なく読み取りが可能になりました。

2023 年 7 月 20 日

公開日: 2023 年 7 月 20 日

Athena は、JDBC ドライバーのバージョン 2.1.0 をリリースします。ドライバーに新しい機能拡張が追加されて、問題が解決されました。

機能強化

以下の Jackson JSON パーサーライブラリがアップグレードしました。

- jackson-annotations 2.15.2 (以前は 2.14.0)
- jackson-core 2.15.2 (以前は 2.14.0)
- jackson-databind 2.15.2 (以前は 2.14.0)

解決された問題

- sql2o ライブラリ使用時に配列パラメータを渡す際の問題を修正しました。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2023 年 7 月 13 日

公開日: 2023 年 9 月 19 日

Athena は、次の機能と改善点を発表しました。

- EXPLAIN ANALYZE — `EXPLAIN ANALYZE` の出力にキュー、分析、計画、実行時間のサポートを追加しました。
- **`EXPLAIN`**EXPLAIN — クエリに集約が含まれている場合、出力に統計が表示されるようになりました。
- Parquet Hive SerDe — Parquet データを読み取る際に処理統計を無視できるようにする `parquet.ignore.statistics` プロパティを追加しました。詳細については、Parquet 統計を無視する を参照してください。

`EXPLAIN` と `EXPLAIN ANALYZE` の詳細については、「Athena での EXPLAIN および EXPLAIN ANALYZE の使用」を参照してください。Parquet Hive については、「Parquet SerDe」を参照してください。

2023 年 7 月 3 日

公開日: 2023 年 7 月 25 日

Athena では、2023 年 7 月 3 日から CloudTrail ログのクエリ文字列を編集できるようになりました。クエリ文字列の値は ***OMITTED*** になりました。機密情報を含む可能性のあるテーブル名やフィルター値の意図しない漏洩を防止するために、この変更を行いました。これまで CloudTrail ログを使用してクエリ文字列全体にアクセスしていた場合は、Athena::GetQueryExecution API を使用し、CloudTrail ログから responseElements.queryExecutionId の値を渡すことをお勧めします。詳細については、「Amazon Athena API リファレンス」の「GetQueryExecution」ページのアクションを参照してください。

2023 年 6 月 30 日

公開日: 2023 年 6 月 30 日

Athena クエリエディタで、クエリの作成を高速化するための先行入力コード候補がサポートされるようになりました。次の機能を使用して、SQL クエリをより正確に、より効率的に記述できるようになりました。

- 入力すると、キーワード、ローカル変数、スニペット、およびカタログ項目の候補がリアルタイムで表示されます。
- データベース名またはテーブル名の後にドットを入力すると、エディタはテーブルまたは列のリストを表示して選択できるため便利です。
- スニペットの候補にカーソルを合わせると、概要にスニペットの構文と使用法の概要が表示されます。
- コードの読みやすさを向上させるため、キーワードとその強調表示ルールも Trino と Hive の最新構文に合わせて更新されました。

この機能は、デフォルトでご利用になれます。コードエディタの環境設定でこの機能を有効/無効にできます。

Athena クエリエディタで先行入力コード候補を試すには、Athena コンソール https://console.aws.amazon.com/athena/ にアクセスしてください。

2023 年 6 月 29 日

公開日: 2023 年 6 月 29 日

- Athena は ODBC ドライバーバージョン 2.0.1.0 を発表しました。詳細については、2.0.1.0 のリリースノートを参照してください。新しい ODBC v2 ドライバーをダウンロードするには、

「ODBC 2.x ドライバーのダウンロード」を参照してください。接続情報については、「Amazon Athena ODBC 2.x」を参照してください。

- Athena とその機能 が中東 (UAE) 地域で利用できるようになりました。各 AWS リージョン で利用できる AWS のサービス の一覧については、「地域別の AWS サービス」を参照してください。

2023 年 6 月 28 日

公開日: 2023 年 6 月 28 日

Amazon Athena を使用して、S3 Glacier Flexible Retrieval (以前の Glacier) と S3 Glacier Deep Archive Amazon S3 ストレージクラスから復元されたオブジェクトをクエリできるようになりました。テーブルごとにこの機能を設定します。この機能は、Athena エンジンバージョン 3 の Apache Hive テーブルでのみサポートされています。

詳細については、「復元された Amazon S3 Glacier オブジェクトへのクエリ」を参照してください。

2023 年 6 月 12 日

公開日: 2023 年 6 月 12 日

Athena は、次の修正と改善を発表しました。

- Parquet Reader のタイムスタンプ — Parquet Reader のタイムスタンプを `bigint` (ミリ秒) として読み取るサポートが追加されました。この更新により、Athena エンジンバージョン 2 のサポートと同等に提供されます。
- EXPLAIN ANALYZE — `EXPLAIN ANALYZE` のクエリ統計と出力に物理入力の読み取り時間を追加しました。`EXPLAIN ANALYZE` の詳細については、「Athena での EXPLAIN および EXPLAIN ANALYZE の使用」を参照してください。
- INSERT — `INSERT` で書き込まれたテーブルのクエリパフォーマンスが向上しました。`INSERT` の詳細については、「INSERT INTO」を参照してください。
- Delta Lake テーブル — Delta Lake テーブルの `DROP TABLE` が同時に変更された場合に完全に削除できないという問題が修正されました。

2023 年 6 月 8 日

公開日: 2023 年 6 月 8 日

Amazon Athena for Apache Spark が、以下の新機能を発表しました。

- カスタム Java ライブラリと設定のサポート — Athena の Apache Spark セッションに独自の Java パッケージとカスタム設定を使用できるようになりました。Spark プロパティを使用して、Athena コンソール、AWS CLI、または Athena API で `.jar` ファイル、パッケージ、またはその他のカスタム設定を指定します。詳細については、「JAR ファイルとカスタム Spark 設定を追加する」を参照してください。
- Apache Hudi、Apache Iceberg、Delta Lake テーブルのサポート — Athena for Spark は、Apache Iceberg、Apache Hudi、および Linux Foundation Delta Lake のオープンソースデータレイクストレージテーブル形式をサポートするようになりました。詳細については、「Athena for Spark」での Apache Iceberg、Apache Hudi、および Linux Foundation Delta Lake テーブルの使用に関する「Amazon Athena for Apache Spark で Hive 以外のテーブル形式を使用する」および個々のトピックを参照してください。
- Apache Spark の暗号化サポート — Athena for Spark では、Spark ノード間で転送中のデータと、Spark によってディスクに保存されている保管中のローカルデータの暗号化を有効にできるようになりました。Spark 暗号化を有効にするには、Athena コンソール、AWS CLI、または Athena API を使用できます。詳細については、「Apache Spark 暗号化を有効にする」を参照してください。

Amazon Athena for Apache Spark の詳細については、「Amazon Athena での Apache Spark の使用」を参照してください。

2023 年 6 月 2 日

公開日: 2023 年 6 月 2 日

Athena でキャパシティ予約を削除し、AWS CloudFormation テンプレートを使用して Athena キャパシティ予約を指定できるようになりました。

- キャパシティ予約の削除 – Athena でキャンセルされたキャパシティ予約を削除できるようになりました。予約は削除する前にキャンセルする必要があります。キャパシティ予約を削除すると、予約はアカウントからすぐに削除されます。削除された予約は参照できなくなります (ARN を使用する場合を含む)。予約を削除するために、Athena コンソールまたは Athena API を使用できます。詳細については、「Amazon Athena ユーザーガイド」の「キャパシティ予約の削除」、および「Amazon Athena API リファレンス」の「DeleteCapacityReservationページのアクションを参照してください。」を参照してください。

- キャパシティ予約に AWS CloudFormation テンプレートを使用 – AWS CloudFormation テンプレートと `AWS::Athena::CapacityReservation` リソースを使用して Athena キャパシティ予約を指定でき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::Athena::CapacityReservation」を参照してください。

キャパシティ予約を使用して Athena でキャパシティをプロビジョニングする方法の詳細については、「クエリ処理キャパシティの管理」を参照してください。

2023 年 5 月 25 日

公開日: 2023 年 5 月 25 日

Athena は、データソースコネクタのアップデートをリリースしました。このアップデートにより、フェデレーテッドクエリのパフォーマンスが改善されます。新しいプッシュダウンの最適化と動的フィルタリングにより、Athena ではなくソースデータベースでより多くのオペレーションを実行できるようになります。これらの最適化により、クエリのランタイムとスキャンされるデータ量が削減されます。これらの改善の恩恵を受けるには、Athena エンジンバージョン 3 が必要です。

次のコネクタが更新されました。

- Azure Data Lake Storage
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- Db2
- DynamoDB
- Google BigQuery
- Hortonworks
- MySQL
- Oracle
- PostgreSQL
- Redshift
- SAP HANA
- Snowflake
- SQL Server

- Teradata

データソースコネクタのアップグレードについては、「データソースコネクタの更新」を参照してください。

2023 年 5 月 18 日

公開日: 2023 年 5 月 18 日

Amazon Athena への IPv6 インバウンド接続に AWS PrivateLink を使用できるようになりました。

Amazon Athena は、AWS PrivateLink を含めるために、インターネットプロトコルバージョン 6 (IPv6) エンドポイントを通じたインバウンド接続のサポートを拡張しました。本日より、以前から利用可能であったパブリック IPv6 エンドポイントに加えて、AWS PrivateLink を使用して Amazon Virtual Private Cloud (Amazon VPC) からも安全かつプライベートに Athena に接続できるようになりました。

インターネットの急速な成長により、利用可能なインターネットプロトコルバージョン 4 (IPv4) アドレスが枯渇しつつあります。IPv6 では、使用可能なアドレスの数が数倍に増加するため、VPC 内で重複するアドレス空間を管理する必要がなくなりました。このリリースでは、IPv6 アドレス指定の利点と、AWS PrivateLink のセキュリティおよびパフォーマンスの利点を組み合わせることができるようになりました。

プログラムにより AWS サービスに接続するには、AWS CLI または AWS SDK を使用してエンドポイントを指定できます。サービスエンドポイントと Athena サービスエンドポイントの詳細については、「Amazon Web Services 全般のリファレンス」の「AWSサービスエンドポイント」と「Amazon Athena エンドポイントとクォータ」を参照してください。

2023 年 5 月 15 日

公開日: 2023 年 5 月 15 日

Athena は、DynamoDB、CloudWatch Logs、CloudWatch Metrics、AWS CMDB 用の Apache Spark DataSourceV2 (DSV2) コネクタのリリースを発表しました。新しい DSV2 コネクタで Spark を使用してこれらのデータソースをクエリできます。DSV2 コネクタは、対応する Athena フェデレーテッドコネクタと同じパラメータを使用します。DSV2 コネクタは Spark ワーカー上で直接実行されるため、使用するために Lambda 関数をデプロイする必要はありません。

詳細については、「Apache Spark 用の Athena データソースコネクタ」を参照してください。

2023 年 5 月 10 日

公開日: 2023 年 5 月 10 日

Athena 用の ODBC 1.1.20 ドライバーがリリースされました。

特徴と拡張機能:

- Lake Formation エンドポイントのオーバーライドのサポート。
- ADFS 認証プラグインには、証明書利用者の値 (LoginToRP) を設定するための新しいパラメータがあります。
- AWS ライブラリの更新。

バグ修正:

- SQLPrepare() メソッドの送信に失敗した場合における、プリペアドステートメントの割り当て解除の失敗。
- C 型を SQL 型に変換する場合における、プリペアドステートメントパラメータのバインドエラー。
- EXPLAIN および EXPLAIN ANALYZE クエリが SQLPrepare() および SQLExecute() を使用した場合における、データが返されないエラー。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「ODBC を使用した Amazon Athena への接続」を参照してください。

2023 年 5 月 8 日

公開日: 2023 年 5 月 8 日

Athena は、次の修正と改善を発表しました。

- Hudi 統合の更新 – Athena は、Apache Hudi との統合を更新しました。Athena を使用して Hudi 0.12.2 テーブルをクエリできるようになり、Hudi テーブルの Hudi メタデータのリストがサポートされるようになりました。詳細については、「Athena を使用した Apache Hudi データセットのクエリ」および「Hudi メタデータのリスト」を参照してください。
- タイムスタンプ変換の修正 - 精度の低いデータ型へのタイムスタンプ変換の処理を修正しました。以前は、Athena エンジンバージョン 3 は、キャスト中に値を切り捨てるのではなく、ターゲットタイプに誤って丸めていました。

次の例は、修正前の誤った処理を示しています。

例 1: マイクロ秒単位のタイムスタンプからミリ秒単位へのキャスト

サンプルデータ

```
A, 2020-06-10 15:55:23.383
B, 2020-06-10 15:55:23.382
C, 2020-06-10 15:55:23.383345
D, 2020-06-10 15:55:23.383945
E, 2020-06-10 15:55:23.383345734
F, 2020-06-10 15:55:23.383945278
```

次のクエリは、特定の値に一致するタイムスタンプの取得を試みます。

```
SELECT *
FROM table
WHERE timestamps.col = timestamp'2020-06-10 15:55:23.383'
```

クエリは次の結果を返しました。

```
A, 2020-06-10 15:55:23.383
C, 2020-06-10 15:55:23.383
E, 2020-06-10 15:55:23.383
```

修正前は、Athena に値 2020-06-10 15:55:23.383945 または 2020-06-10 15:55:23.383945278 が含まれていません。これは、この値が 2020-06-10 15:55:23.384 に丸められていたためです。

例 2: タイムスタンプから日付へのキャスト

次のクエリは誤った結果を返していました。

```
SELECT date(timestamp '2020-12-31 23:59:59.999')
```

結果

```
2021-01-01
```

修正前は、Athena が値を切り上げていたため、日付が進んでいました。このような値は、切り上げるのではなく、切り捨てられるようになりました。

2023 年 4 月 28 日

公開日: 2023 年 4 月 28 日

Amazon Athena でキャパシティ予約を使用して、フルマネージドコンピューティング性能で SQL クエリを実行できるようになりました。

プロビジョンドキャパシティのワークロード管理機能により、最も重要なインタラクティブなワークロードの優先順位付け、制御、スケーリングを行えます。いつでも容量を追加して、同時に実行するクエリの数を増やしたり、そのキャパシティを使用するワークロードを制御したり、ワークロード間でキャパシティを共有したりできます。

詳細については、「クエリ処理キャパシティの管理」を参照してください。料金の詳細については、「Amazon Athena 料金表」ページを参照してください。

2023 年 4 月 17 日

公開日: 2023 年 4 月 17 日

Athena は、JDBC ドライバーのバージョン 2.0.36 をリリースします。ドライバーに新機能を追加し、問題を解決しました。

新機能

- AD FS 認証でカスタマイズ可能な依存パーティ識別子を使用できるようになりました。
- コネクタを使用しているアプリケーションの名前をユーザーエージェント文字列に追加できるようになりました。

解決された問題

- `getSchema()` が存在しないスキーマの取得に使用されたときに発生したエラーを修正しました。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2023 年 4 月 14 日

公開日: 2023 年 6 月 20 日

Athena は、次の修正と改善を発表しました。

- 文字列をタイムスタンプにキャストする場合、日付と時刻またはタイムゾーンの間にスペースが必要です。詳細については、「文字列からタイムスタンプにキャストする際に日付と時刻の値の間に必要なスペース」を参照してください。
- タイムスタンプの精度の処理方法において、互換性を失う可能性のある変更を削除しました。Athena エンジンバージョン 2 と Athena エンジンバージョン 3 との整合性を維持するために、タイムスタンプの精度は、デフォルトでマイクロ秒ではなくミリ秒単位を使用するようになりました。
- Athena は、クエリを実行するときに、クエリ出力バケットへのアクセスを常時強制するようになりました。StartQueryExecution アクションを実行するすべての IAM プリンシパルが、クエリ出力バケットに対する S3:GetBucketLocation 権限を持っていることを確認してください。

2023 年 4 月 4 日

公開日: 2023 年 4 月 4 日

Amazon Athenaを使用して、フェデレーティッドデータソースでビューを作成してクエリできるようになりました。1 つのフェデレーティッドビューを使用して、複数の外部テーブルまたはデータのサブセットをクエリできます。これにより、必要な SQL が簡略化され、SQL を使用してデータをクエリする必要があるエンドユーザーからのデータソースを柔軟に難読化できます。

詳細については、ビューの使用およびフェデレーティッドクエリの実行を参照してください。

2023 年 3 月 30 日

公開日: 2023 年 3 月 30 日

Amazon Athena からのお知らせとして、 Amazon Athena for Apache Spark を利用できる AWS リージョン が追加されました。

このリリースでは、Amazon Athena for Apache Spark の提供地域を、アジアパシフィック (ムンバイ)、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) に拡大します。

Amazon Athena for Apache Spark の詳細については、「Amazon Athena での Apache Spark の使用」を参照してください。

2023 年 3 月 28 日

公開日: 2023 年 3 月 28 日

Athena は、次の修正と改善を発表しました。

- GetQueryExecution とBatchGetQueryExecution Athena API アクションに対応して、新しい subStatementType フィールドには、実行されたクエリのタイプ (SELECT、INSERT、UNLOAD、CREATE_TABLE、CREATE_TABLE_AS_SELECT など) が表示されます。
- Apache Hive の書き込み操作でマニフェストファイルが正しく暗号化されなかったバグを修正しました。
- Athena エンジンバージョン 3 は approx_percentile 関数内の NaN および Infinity 値を正しく処理するようになりました。この approx_percentile 関数は、指定されたパーセンテージでデータセットの近似パーセンタイルを返します。

  Athena エンジンバージョン 2 では、NaN を Infinity より大きい値として誤って処理します。Athena エンジンバージョン 3 では、NaN および Infinity の扱いを、他の分析関数や統計関数における扱いに従って処理します。以下で、新しい動作について詳しく説明します。

  - NaN がデータセットに存在する場合、Athena は NaN を返します。
  - NaN が存在せず Infinity が存在する場合、Athena は Infinity を非常に大きな数として扱います。
  - Infinity 値が複数存在する場合、Athena はそれらを非常に大きな、同一の数として扱います。必要に応じて、Athena は Infinity を出力します。
  - 1 つのデータセットに -Infinity と -Double.MAX_VALUE が両方含まれ、パーセンタイルの結果が -Double.MAX_VALUE の場合、Athena は -Infinity を返します。
  - 1 つのデータセットに Infinity と Double.MAX_VALUE が両方含まれ、パーセンタイルの結果が Double.MAX_VALUE の場合、Athena は Infinity を返します。
  - Infinity と NaN を計算から除外するには、次の例のように is_finite() 関数を使用します。

    ```
    approx_percentile(x, 0.5) FILTER (WHERE is_finite(x))
    ```

2023 年 3 月 27 日

公開日: 2023 年 3 月 27 日

Amazon Athena の Athena SQL ワークグループに対して、最小レベルの暗号化を指定できるようになりました。この機能により、Athena SQL ワークグループのすべてのクエリの結果が、指定した暗号化レベル以上で暗号化されます。暗号化強度を複数レベルから選択し、データを保護できます。暗号化レベルの下限を設定するには、Athena コンソール、AWS CLI、API、または SDK を使用します。

Apache Spark 対応のワークグループでは、暗号化の下限設定機能は使用できません。詳細については、「ワークグループの最小暗号化の設定」を参照してください。

2023 年 3 月 17 日

公開日: 2023 年 3 月 17 日

Athena は、次の修正と改善を発表しました。

- Amazon Athena DynamoDB コネクタで、「KeyConditionExpressions にはキーごとに 1 つの条件のみ含める必要があります」というエラーメッセージが表示されてクエリが失敗する問題を修正しました。

  この問題が発生するのは、Athena エンジンバージョン 3 では、Athena エンジンバージョン 2 よりも多くの種類の述語をプッシュダウンする可能性があるためです。Athena エンジンバージョン 3 では、`some_column LIKE 'someprefix%` のような句は、特定の列に下限と上限を適用するフィルター述語としてプッシュダウンされます。Athena エンジンのバージョン 2 では、これらの述語がプッシュダウンされることはありませんでした。Athena エンジンバージョン 3 では、`some_column` がソートキー列の場合、エンジンはフィルター述語を DynamoDB コネクタにプッシュダウンします。その後、フィルター述語はさらに DynamoDB サービスにプッシュダウンされます。DynamoDB はソートキーに対して複数のフィルター条件をサポートしていないため、DynamoDB はエラーを返します。

  この問題を修正するには、Amazon Athena DynamoDB コネクタをバージョン 2023.11.1 に更新してください。コネクタを更新する手順については、「データソースコネクタの更新」を参照してください。

2023 年 3 月 8 日

公開日: 2023 年 3 月 8 日

Athena は、次の修正と改善を発表しました。

- タイムスタンプ述語値がミリ秒ではなくマイクロ秒で送信されるフェデレーティッドクエリの問題を修正しました。

2023 年 2 月 15 日

公開日: 2023 年 2 月 15 日

Athena は、次の修正と改善を発表しました。

- クライアント側の暗号化を使用して Amazon S3 のデータを暗号化し、Iceberg 書き込みオペレーションを実行できるようになりました。
- Amazon S3 の Iceberg 書き込みオペレーションに関して、サーバー側の暗号化に影響する問題を修正しました。

2023 年 1 月 31 日

公開日: 2023 年 1 月 31 日

Amazon Athena を使用して、Google Cloud Storage 内のデータをクエリできるようになりました。Amazon S3 と同様に、Google Cloud Storage とはデータをバケットに保存するマネージドサービスです。Google Cloud Storage の Athena コネクタを使用して、外部データでインタラクティブなフェデレーションクエリを実行します。

詳細については、「Amazon Athena Google Cloud Storage コネクタ」を参照してください。

2023 年 1 月 20 日

公開日: 2023 年 1 月 20 日

Athena の圧縮サポートの拡張ドキュメントが表示できるようになりました。Hive テーブル圧縮、Iceberg テーブル圧縮 および ZSTD 圧縮レベル に個別のトピックは追加されました。

詳細については、「Athena での圧縮のサポート」を参照してください。

2023 年 1 月 3 日

公開日: 2023 年 1 月 3 日

Athena は、次の更新を発表しました。

- Hive メタストア用の追加コマンド - Athena を使用すると、自己管理型の Apache Hive メタストアにメタデータカタログとして接続し、Amazon S3 に保存されているデータをクエリできます。今回のリリースでは、`CREATE TABLE AS` (CTAS)、`INSERT INTO` および 12 個の追加データ定義言語 (DDL) コマンドを使用して Apache Hive メタストアを操作できるようになりました。この拡張された SQL 機能セットを使用して、Athena から Hive メタストアのスキーマを直接管理できます。

  詳細については、「外部 Hive メタストア用の Athena データコネクタの使用」を参照してください。
- JDBC ドライバーのバージョン 2.0.35 - Athena が JDBC ドライバーのバージョン 2.0.35 をリリースします。JDBC 2.0.35 ドライバーには、次の更新が含まれています。
  - ドライバーは Jackson JSON パーサーに対して次のライブラリを使用するようになりました。
    - jackson-annotations 2.14.0 (以前は 2.13.2)
    - jackson-core 2.14.0 (以前は 2.13.2)
    - jackson-databind 2.14.0 (以前は 2.13.2.2)
  - JDBC バージョン 4.1 のサポートは中止されました。

  詳細については、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

Athena リリースノート (2022 年)

2022 年 12 月 14 日

公開日: 2022 年 12 月 14 日

Kafka 用 Amazon Athena コネクタを使用して、ストリーミングデータに対する SQL クエリを実行できるようになりました。たとえば、Amazon Managed Streaming for Apache Kafka (Amazon MSK) でリアルタイムストリーミングデータに対して分析クエリを実行し、Amazon S3 のデータレイクの履歴データと結合できます。

Kafka 用 Amazon Athena コネクタは、複数のストリーミングエンジンでのクエリをサポートします。Athena を使用すると、Amazon MSK でプロビジョニングされたサーバーレスクラスタ、セルフマネージドの Kafka デプロイ、および Confluent Cloud のストリーミングデータで SQL クエリを実行できます。

詳細については、「Amazon Athena MSK コネクタ」を参照してください。

2022 年 12 月 2 日

公開日: 2022 年 12 月 2 日

Athena は、JDBC ドライバーのバージョン 2.0.34 をリリースします。JDBC 2.0.34 ドライバーには、次の新機能と解決された問題が含まれています。

- クエリ結果の再利用のサポート - クエリを実行するたびに Athena に結果を再計算させる代わりに、以前に実行したクエリの結果を指定した制限時間まで再利用できるようになりました。詳細については、JDBC のダウンロードページから入手できる「インストールおよび設定ガイド」と「クエリの結果を再利用する」を参照してください。
- EC2InstanceMetadata サポート - JDBC ドライバーは、IAM インスタンスプロファイルを使用する Ec2InstanceMetadata 認証方法をサポートするようになりました。
- 文字ベースの例外の修正 - 特定の言語文字を含むクエリで発生した例外を修正しました。
- 脆弱性の修正 - コネクタにパッケージされた AWS 依存関係に関連する脆弱性を修正しました。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2022 年 11 月 30 日

公開日: 2022 年 11 月 30 日

Athena で Apache Spark アプリケーションと Jupyter 互換ノートブックをインタラクティブに作成および実行できるようになりました。リソースの計画、設定、または管理を行うことなく、Spark を使用して Athena でデータ分析を実行します。処理用の Spark コードを送信し、結果を直接受け取ります。Amazon Athena コンソールのシンプルなノートブックエクスペリエンスを使用して、Python または Athena ノートブック API を使用して Apache Spark アプリケーションを開発できます。

Amazon Athena の Apache Spark はサーバーレスであり、オンデマンドで自動的にスケーリングされるため、データ量や処理要件の変化に合わせて瞬時に処理できます。

詳細については、「Amazon Athena での Apache Spark の使用」を参照してください。

2022 年 11 月 18 日

公開日: 2022 年 11 月 18 日

IBM Db2 用の Amazon Athena コネクタを使用して Athena から Db2 をクエリできるようになりました。たとえば、Db2 上のデータウェアハウス、および Amazon S3 内のデータレイクに対して分析クエリを実行できます。

Amazon Athena Db2 コネクタは、Lambda 環境変数を使用して、いくつかの設定オプションを公開します。設定オプション、パラメータ、接続文字列、デプロイ、および制限事項については、「Amazon Athena IBM Db2 コネクタ」を参照してください。

2022 年 11 月 17 日

公開日: 2022 年 11 月 17 日

Athena エンジンのバージョン 3 の Apache Iceberg サポートでは、次の強化された ACID トランザクション機能が提供されるようになりました。

- ORC と Avro のサポート - Apache Avro と Apache ORC の行および列ベースのファイル形式を使用して Iceberg テーブルを作成できます。これらの形式に対するサポートは、Parquet 向けの既存のサポートに追加されます。
- MERGE INTO - `MERGE INTO` コマンドを使用して、大規模なデータを効率的にマージします。`MERGE INTO` は `INSERT`、`UPDATE`、および `DELETE` のオペレーションを 1 つのトランザクションにまとめます。これにより、データパイプラインの処理オーバーヘッドが軽減され、SQL の記述も少なくなります。詳細については、Iceberg テーブルデータの更新およびMERGE INTOを参照してください。
- CTAS と VIEW のサポート - `CREATE TABLE AS SELECT` (CTAS) および `CREATE VIEW` ステートメントを Iceberg テーブルで使用します。詳細については、CREATE TABLE ASおよびCREATE VIEWを参照してください。
- VACUUM サポート - `VACUUM` ステートメントを使用して、不要になったスナップショットやデータを削除することで、データレイクを最適化できます。この機能を使用すると、読み込みパフォーマンスを向上し、GDPR などのルールの要件を満たすことができます。詳細については、Iceberg テーブルの最適化およびVACUUMを参照してください。

これらの新機能には Athena エンジンバージョン 3 が必要であり、Athena がサポートされているすべてのリージョンで利用できます。これらは Athena コンソール、ドライバー、または API で使用できます。

Athena での Iceberg の使用に関する詳細については、「Apache Iceberg テーブルの使用」を参照してください。

2022 年 11 月 14 日

公開日: 2022 年 11 月 14 日

Amazon Athena は、IPv6 経由で Athena 関数を呼び出すために使用できるインバウンド接続用の IPv6 エンドポイントをサポートするようになりました。この機能を使用して、IPv6 コンプライアンス要件を満たすことができます。また、IPv4 と IPv6 間のアドレス変換を処理するための追加のネットワーク機器が不要になります。

この機能を使用するには、IPv4 と IPv6 の両方をサポートする新しい Athena デュアルスタックエンドポイントを使用するようにアプリケーションを設定します。デュアルスタックのエンドポイントは、形式 `athena.region.api.aws` を使用します。たとえば、米国東部 (バージニア北部) リージョンのデュアルスタックのエンドポイントは `athena.us-east-1.api.aws` です。

デュアルスタックの Athena エンドポイントにリクエストを行うと、エンドポイントは、ネットワークとクライアントが使用するプロトコルに応じて IPv6 または IPv4 アドレスに解決されます。プログラムにより AWS サービスに接続するには、AWS CLI または AWS SDK を使用してエンドポイントを指定できます。

サービスエンドポイントの詳細については、「AWS サービスエンドポイント」を参照してください。Athena のサービスエンドポイントの詳細については、AWS ドキュメントの「Amazon Athena エンドポイントとクォータ」を参照してください。

インバウンド接続に新しい Athena デュアルスタックエンドポイントを追加料金なしで使用できます。デュアルスタックのエンドポイントは、一般的にすべての AWS リージョン で使用できます。

2022 年 11 月 11 日

公開日: 2022 年 11 月 11 日

Athena は、次の修正と改善を発表しました。

- Lake Formation のきめ細かいアクセス制御の拡張 - サポートされている任意のファイルまたはテーブル形式で保存されているデータに対して、Athena クエリで AWS Lake Formation のきめ細

かいアクセス制御ポリシーを使用できるようになりました。データフィルターを使用してクエリ結果のデータへのアクセスを制限し、列レベル、行レベル、およびセルレベルのセキュリティを実現するために、Lake Formation では、きめ細かなアクセスコントロールを使用できます。Athena でサポートされているテーブル形式には、Apache Iceberg、Apache Hudi、Apache Hive などがあります。Athena がサポートするすべてのリージョンで、きめ細かなアクセス制御を使用できます。新しい機能が提供され、クエリのパフォーマンスが向上する、拡張されたテーブルとファイル形式のサポートには Athena エンジンバージョン 3 が必須ですが、Lake Formation でのきめ細かいアクセス制御ポリシーのセットアップ方法は変わりません。

この拡張されたきめ細かいアクセス制御を Athena で使用する場合、次の考慮事項があります。

- 説明 - Lake Formation で定義されている行またはセルのフィルタリング情報とクエリ統計情報は、`EXPLAIN` および `EXPLAIN ANALYZE` の出力に表示されません。Athena の `EXPLAIN` については、「Athena での EXPLAIN および EXPLAIN ANALYZE の使用」を参照してください。
- 外部 Hive メタストア - Apache Hive の非表示列はきめ細かいアクセス制御フィルタリングには使用できません。また、Apache Hive の非表示システムテーブルはきめ細かなアクセス制御ではサポートされていません。詳細については、トピック「外部 Hive メタストア用の Athena データコネクタの使用」の「考慮事項と制約事項」を参照してください。
- クエリ統計 - Lake Formation で行レベルのフィルターが定義されている場合、ステージレベルの入出力行数とデータサイズ情報は Athena クエリ統計に表示されません。Athena クエリの統計情報を表示する方法については、「完了したクエリの統計と実行の詳細の表示」および「getQueryRuntimeStatistics」を参照してください。
- ワークグループ - 同じ Athena ワークグループのユーザーは、Lake Formation のきめ細かいアクセス制御によって、ワークグループがアクセスできるように設定したデータを見ることができます。Athena を使用して Lake Formation に登録されたデータをクエリする方法については、「Athena を使用して AWS Lake Formation に登録されたデータをクエリする」を参照してください。

Lake Formation でのきめ細かなアクセス制御の使用方法については、「AWS ビッグデータブログ」の「AWS Lake Formation を使用したきめ細かなアクセス制御の管理」を参照してください。

- Athena フェデレーションクエリ - Athena フェデレーションクエリでは、`struct` オブジェクト内のフィールド名の元の大文字と小文字が保持されるようになりました。以前は、`struct` フィールド名は自動的に小文字になっていました。

2022 年 11 月 8 日

公開日: 2022 年 11 月 8 日

クエリ結果の再利用キャッシュ機能を使用して、Athena でのクエリの繰り返しを高速化できるようになりました。リピートクエリは、最近送信されたクエリと同じ SQL クエリであり、同じ結果が得られます。複数の同じクエリを実行する必要がある場合、結果を再利用するキャッシュを使用すると、結果の生成に必要な時間を短縮できます。結果再利用キャッシュにより、スキャンされるバイト数を減らすことでコストも削減されます。

詳細については、「クエリの結果を再利用する」を参照してください。

2022 年 10 月 13 日

公開日: 2022 年 10 月 13 日

Athena は、Athena エンジンバージョン 3 を発表しました。

Athena では、SQL クエリエンジンがアップグレードされ、Trino オープンソースプロジェクトの最新機能が利用できるようになりました。Athena エンジンバージョン 2 のすべての機能をサポートしているのに加え、Athena エンジンバージョン 3 には、50 以上の新しい SQL 関数、30 の新機能、90 を超えるクエリパフォーマンスの改善が含まれています。本日のリリースで、オープンソースソフトウェア管理に対する継続的な統合アプローチも Athena に導入されました。これにより、Trino と Presto のプロジェクトの流通性が向上するため、Athena エンジン内で統合、チューニングされたコミュニティの改善にすばやくアクセスできるようになります

詳細については、「Athena エンジンバージョン 3」を参照してください。

2022 年 10 月 10 日

公開日: 2022 年 10 月 10 日

Athena は、JDBC ドライバーのバージョン 2.0.33 をリリースします。JDBC 2.0.33 ドライバーには、次の変更が含まれています。

- 新しいドライバーバージョン、JDBC バージョン、およびプラグイン名のプロパティが、認証情報プロバイダクラスのユーザーエージェント文字列に追加されました。
- エラーメッセージが修正され、必要な情報が追加されました。
- 接続が閉じられたり、Athena のプリペアドステートメントの実行が失敗したりすると、プリペアドステートメントの割り当てが解除されるようになりました。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2022 年 9 月 23 日

公開日: 2022 年 9 月 26 日

Amazon Athena Neptune コネクタは、列名とテーブル名の大文字と小文字を区別しない照合をサポートするようになりました。

- Neptune データソースコネクタは、AWS Glue のテーブルの列名がすべて小文字であっても、大文字と小文字を使用する Neptune テーブルの列名を解決できます。 この動作を有効にするには、Neptune コネクタの Lambda 関数で `enable_caseinsensitivematch` 環境変数を `true` に設定します。
- AWS Glue は小文字のテーブル名のみをサポートしているため、Neptune 用の AWS Glue テーブルを作成する場合は、AWS Glue テーブルパラメータ `"glabel" = table_name` を指定します。

Neptune コネクタの詳細については、「Amazon Athena Neptune コネクタ」を参照してください。

2022 年 9 月 13 日

公開日: 2022 年 9 月 13 日

Athena は、次の修正と改善を発表しました。

- 外部 Hive メタストア — 外部 Hive メタストア (EHMS) に存在しないパーティションが `WHERE` 句に含まれている場合、Athena で例外をスローする代わりに `NULL` を返すようになりました。 新しい動作は AWS Glue Data Catalog の動作と一致します。
- パラメータ化されたクエリ — パラメータ化されたクエリの値を `DOUBLE` データ型にキャストできるようになりました。
- Apache Iceberg — Amazon S3 バケットでオブジェクトロックが有効である場合に、Iceberg テーブルに書き込みオペレーションができるようになりました。

2022 年 8 月 31 日

公開日: 2022 年 8 月 31 日

Amazon Athena では、アジアパシフィック (ジャカルタ) リージョンで Athena とその機能が利用可能になりました。

このリリースで、アジアパシフィックでの Athena の提供が拡張され、アジアパシフィック (香港)、アジアパシフィック (ジャカルタ)、アジアパシフィック (ムンバイ)、アジアパシフィック (大阪)、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)、およびアジアパシフィック (東京) が含まれるようになりました。これらのリージョンや他のリージョンで利用可能な AWS のサービス の詳細な一覧については、「AWS リージョン 別のサービス表」を参照してください。

2022 年 8 月 23 日

公開日: 2022 年 8 月 23 日

Athena Query Federation SDK のリリース v2022.32.1 には、次の変更が含まれています。

- Amazon RDS インスタンスへの SSL ベースの接続用に、Amazon Athena Oracle データソースコネクタのサポートが追加されました。このサポートは、Transport Layer Security (TLS) プロトコルと、クライアントによるサーバーの認証に限定されています。相互認証は Amazon RDS でサポートされていないため、相互認証のサポートは更新に含まれていません。

  詳細については、「Amazon Athena Oracle コネクタ」を参照してください。

2022 年 8 月 3 日

公開日: 2022 年 8 月 3 日

Athena で、JDBC ドライバーのバージョン 2.0.32 がリリースされます。JDBC 2.0.32 ドライバーには、次の変更が含まれています。

- Athena SDK に送信される `User-Agent` 文字列が、ドライバーのバージョン、JDBC 仕様のバージョン、認証プラグインの名前を含むように拡張されました。
- `CheckNonProxyHost` パラメータに値が提供されていない場合にスローされる `NullPointerException` を修正しました。
- BrowserSAML 認証プラグインでの `login_url` 解析に関する問題を修正しました。
- `UseProxyforIdp` パラメータが `true` に設定された場合に発生するプロキシのホストの問題を修正しました。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2022 年 8 月 1 日

公開日: 2022 年 8 月 1 日

Athena が Athena Query Federation SDK と Athena の事前構築済みデータソースコネクタの改善を発表しました。改善には次が含まれます。

- 構造体の解析 – 特定の複雑な構造体がすべてのデータを表示できないという Athena Query Federation SDK の `GlueFieldLexer` の解析の問題を修正しました。この問題は、Athena Query Federation SDK 上に構築されたコネクタに影響を与えました
- AWS Glue テーブル – AWS Glue テーブルの `set` および `decimal` の列タイプのサポートがさらに追加されました。
- DynamoDB コネクタ – DynamoDB 属性名の大文字と小文字を無視する機能が追加されました。詳細については、Amazon Athena DynamoDB コネクタ ページの パラメータ セクションにある「`disable_projection_and_casing`」を参照してください。

詳細については、GitHub の「Release v2022.30.2 of Athena Query Federation」(Athena Query Federation のリリース v2022.30.2) を参照してください。

2022 年 7 月 21 日

公開日: 2022 年 7 月 21 日

Athena コンソールで、パフォーマンスメトリクスとインタラクティブで視覚的なクエリ分析ツールを使用して、クエリを分析およびデバッグできるようになりました。クエリのパフォーマンスデータと実行の詳細は、クエリのボトルネックの特定、クエリの各ステージの演算子と統計の検査、ステージ間を流れるデータ量の追跡、クエリ述語の影響の検証に役立ちます。これで、次の操作を実行できます。

- 1 回クリックするだけで、クエリの分散型および論理実行プランにアクセスできます。
- ステージが実行される前に、各ステージでのオペレーションを詳しく確認します。
- キュー、計画、実行の各段階で費やされた時間のメトリクスを使用して、完了したクエリのパフォーマンスを視覚化します。
- クエリで処理および出力されたソースデータの行数と量に関する情報を取得します。
- コンテキストで示され、インタラクティブなグラフとしてフォーマットされたクエリのきめ細かい実行の詳細を表示します。
- ステージレベルの正確な実行の詳細を使用して、クエリを通じたデータの流れを理解します。

- 新しい API を使用してプログラムでクエリパフォーマンスデータを分析し、クエリランタイム統計を取得します。これも本日リリースされました。

クエリでこれらの機能を使用する方法については、AWS YouTube チャンネルの動画のチュートリアル「Optimize Amazon Athena Queries with New Query Analysis Tools」(新しいクエリ分析ツールを使用して Amazon Athena クエリを最適化する) をご覧ください。

ドキュメントについては、「SQL クエリの実行プランの表示」および「完了したクエリの統計と実行の詳細の表示」を参照してください。

2022 年 7 月 11 日

公開日: 2022 年 7 月 11 日

SQL ステートメントを事前に準備することなく、パラメータ化されたクエリを Athena コンソールまたは API から直接実行できるようになりました。

疑問符形式のパラメータを含むクエリを Athena コンソールで実行すると、パラメータの値を直接入力することを促すプロンプトがユーザーインターフェイスで表示されるようになりました。これにより、クエリを実行するたびにクエリエディタでリテラル値を変更する必要がなくなります。

強化された query execution API を使用する場合、1 回の呼び出しで実行パラメータとその値を指定できるようになりました。

詳細については、このユーザーガイドの「パラメータ化されたクエリの使用」と「AWS ビッグデータブログ」の記事「Amazon Athena のパラメータ化されたクエリを使用して Data as a Service (DaaS) を提供する」を参照してください。

2022 年 7 月 8 日

公開日: 2022 年 7 月 8 日

Athena は、次の修正と改善を発表しました。

- クエリの失敗を引き起こしていた SageMaker エンドポイント (UDF) の DATE 列変換処理の問題を修正しました。

2022 年 6 月 6 日

公開日: 2022 年 6 月 6 日

Athena が JDBC ドライバーのバージョン 2.0.31 をリリース。JDBC 2.0.31 ドライバーには、次の変更が含まれています。

- log4j 依存関係の問題 — log4j 依存関係によって引き起こされる「ドライバークラスが見つからない」エラーメッセージを解決しました。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2022 年 5 月 25 日

公開日: 2022 年 5 月 25 日

Athena は、次の修正と改善を発表しました。

- Iceberg のサポート
  - クロスリージョンクエリのサポートが導入されました。これにより、使用している AWS リージョンとは異なる AWS リージョンにある Iceberg テーブルをクエリできるようになりました。クロスリージョンクエリは、中国リージョンではサポートされていません。
  - サーバー側暗号化設定のサポートが導入されました。これにより、Amazon S3 での Iceberg 書き込み操作からのデータの暗号化に SSE-S3/SSE-KMS を使用できるようになりました。

  Athena での Apache Iceberg の使用に関する詳細については、「Apache Iceberg テーブルの使用」を参照してください。
- JDBC 2.0.30 ドライバーリリース

  Athena 用の JDBC 2.0.30 ドライバーでは、以下の点が改善されました。
  - パラメータ化されたプリペアドステートメントに影響を及ぼしていたデータ競合の問題を修正。
  - Gradle ビルド環境で発生していたアプリケーションの起動問題を修正。

  JDBC 2.0.30 ドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2022 年 5 月 6 日

公開日: 2022 年 5 月 6 日

Athena 用の JDBC 2.0.29 ドライバー と ODBC 1.1.17 ドライバーがリリースされました。

これらのドライバーには、以下の変更が含まれます。

- SAML プラグインブラウザの起動プロセスを更新しました。

これらの変更についての詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

2022 年 4 月 22 日

公開日: 2022 年 4 月 22 日

Athena は、次の修正と改善を発表しました。

- 次の条件が満たされたときにパーティションキャッシュで発生していた、パーティションインデックスとフィルタリング機能の問題を修正しました。
  - テーブルの AWS Glue テーブルプロパティで `partition_filtering.enabled` キーが `true` に設定されている。
  - 同じテーブルが異なるパーティションフィルター値で複数回使用されている。

2022 年 4 月 21 日

公開日: 2022 年 4 月 21 日

Amazon Athena を使用して、Google BigQuery、Azure Synapse、Snowflake などの新しいデータソースに対してフェデレーティッドクエリを実行できるようになりました。新しいデータソースコネクタには、次のものがあります。

- Azure Data Lake Storage (ADLS) Gen2
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- Google BigQuery
- Hortonworks
- Microsoft SQL Server
- Oracle

- SAP HANA (Express Edition)
- Snowflake
- Teradata

Athena でサポートされるデータソースの完全なリストについては、「使用可能なデータソースコネクタ」を参照してください。

使用可能なソースを参照してデータに接続しやすくなるように、Athena コンソールの新しくなった [Data Sources] (データソース) 画面から使用可能なコネクタを検索、並べ替え、およびフィルターできるようになりました。

フェデレーティッドソースに対するクエリについては、「Amazon Athena 横串検索の使用」および「フェデレーティッドクエリの実行」を参照してください。

2022 年 4 月 13 日

公開日: 2022 年 4 月 13 日

Athena は JDBC ドライバーのバージョン 2.0.28 をリリースします。JDBC 2.0.28 ドライバーには、次の変更が含まれています。

- JWT のサポート - ドライバーが、認証に使用される JSON ウェブトークン (JWT) をサポートするようになりました。JDBC ドライバーでの JWT の使用については、「JDBC ドライバーページ」からダウンロードできるインストールと設定のガイドを参照してください。
- Log4j ライブラリの更新 - JDBC ドライバーが、次の Log4j ライブラリを使用するようになりました。
  - Log4j-API 2.17.1 (以前は 2.17.0)
  - Log4j-core 2.17.1 (以前は 2.17.0)
  - Log4j-jcl 2.17.2
- その他の改善点 - 新しいドライバーには、以下の改善とバグ修正も含まれています。
  - JDBC を通じて Athena 準備済みステートメント機能を使用できるようになりました。準備済みステートメントについては、「パラメータ化されたクエリの使用」を参照してください。
  - 中国リージョンで Athena JDBC SAML フェデレーションが機能するようになりました。
  - その他の小さな改善。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2022 年 3 月 30 日

公開日: 2022 年 3 月 30 日

Athena は、次の修正と改善を発表しました。

- クロスリージョンクエリ— Athena を使用して、アジアパシフィック (香港)、中東 (バーレーン)、アフリカ (ケープタウン)、欧州 (ミラノ) を含む AWS リージョン 全体で Amazon S3 バケット内のデータをクエリできるようになりました。クロスリージョンクエリは、中国リージョンではサポートされていません。
  - Athenaが利用できる AWS リージョン のリストについては、「Amazon Athena エンドポイントとクォータ」を参照してください。
  - デフォルトで無効になっている AWS リージョン の有効化の詳細については、「リージョンを有効にする」を参照してください。
  - リージョン間のクエリについては、「リージョン間のクエリ」を参照してください。

2022 年 3 月 18 日

公開日: 2022 年 3 月 18 日

Athena は、次の修正と改善を発表しました。

- 動的フィルタリング – 動的フィルタリングは、対応するテーブルの各レコードにフィルタを効率的に適用することで、整数列に対して改善されました。
- Iceberg – 2GB を超える Iceberg Parquet ファイルを書き込むときにエラーが発生する問題を修正しました。
- 非圧縮出力 – CREATE TABLE ステートメントが、非圧縮ファイルの書き込みをサポートするようになりました。非圧縮ファイルを書き込むには、次の構文を使用します。
  - CREATE TABLE (テキストファイルまたは JSON) – TBLPROPERTIES で、`write.compression = NONE` を指定します。
  - CREATE TABLE (Parquet) – TBLPROPERTIES で、`parquet.compression = UNCOMPRESSED` を指定します。
  - CREATE TABLE (ORC) – TBLPROPERTIES で、`orc.compress = NONE` を指定します。

- 圧縮 – あるフォーマットで圧縮されたファイルを作成したが、デフォルト以外の圧縮方法を使用したときに、別の圧縮形式のファイル拡張子を使用したテキストファイルテーブルの挿入に関する問題を修正しました。
- Avro – Avro ファイルから固定タイプの小数点を読み込むときに発生する問題を修正しました。

2022 年 3 月 2 日

公開日: 2022 年 3 月 2 日

Athena は、次の機能と改善点を発表しました。

- クエリ結果の場所バケットで ACL が有効になっている場合、クエリ結果に対するフルコントロールアクセスを Simple Storage Service (Amazon S3) バケット所有者に付与できるようになりました。詳細については、「クエリ結果の場所の指定」を参照してください。
- 既存の名前付きクエリを更新できるようになりました。詳細については、「保存されたクエリの使用」を参照してください。

2022 年 2 月 23 日

公開日: 2022 年 2 月 23 日

Athena は、次の修正とパフォーマンス改善を発表しました。

- パフォーマンスを向上させ、メモリエラーを削減するためのメモリ処理の改善。
- Athena は、ストライプフッターに保存されたタイムゾーン情報で ORC タイムスタンプ列を読み込み、タイムゾーン (UTC) を持つ ORC ファイルをフッターに書き込むようになりました。これは、読み込む ORC ファイルが UTC 以外のタイムゾーン環境で作成された場合にのみ、ORC タイムスタンプ読み込みの動作に影響します。
- 十分最適でないクエリプランが発生するシンボリック リンクテーブルサイズの誤見積もりを修正しました。
- 水平分解ビューを Hive メタストア データソースから Athena コンソールでクエリできるようになりました。
- Amazon S3 の読み込みエラーメッセージが改善され、より詳細な Simple Storage Service (Amazon S3) のエラーコード情報が含まれるようになりました。
- ORC 形式の出力ファイルで Apache Hive 3.1 との互換性がなくなる問題を修正しました。

- 一部の DML および DDL クエリで、引用符付きのテーブル名が失敗する問題を修正しました。

2022 年 2 月 15 日

公開日: 2022 年 2 月 15 日

Amazon Athena はアクティブな DML クエリクォータをすべての AWS リージョンで増加させました。アクティブなクエリには、実行中のクエリとキューに登録されたクエリの両方が含まれます。この変更により、これまでよりも多くの DML クエリをアクティブ状態にできるようになりました。

Athena のサービスクォータの詳細については、「Service Quotas」を参照してください。Athena を使用するリージョンのクエリクォータについては、「AWS 全般のリファレンス」の「 Amazon Athena エンドポイントとクォータ」を参照してください。

クォータの使用状況をモニタリングするには、CloudWatch の使用状況メトリクスを使用できます。Athena は、`AWS/Usage` 名前空間に `ActiveQueryCount` メトリクスを公開します。詳細については、「Athena の使用状況に関するメトリクスのモニタリング」を参照してください。

使用状況を確認したら、Service Quotas コンソールを使用して、クォータの引き上げをリクエストできます。以前にアカウントのクォータの引き上げをリクエストしたことがある場合は、新しいデフォルトのアクティブな DML クエリクォータを超えた場合にもリクエストされたクォータが適用されます。それ以外の場合は、すべてのアカウントで新しいデフォルトが使用されます。

2022 年 2 月 14 日

公開日: 2022 年 2 月 14 日

このリリースでは、Athena GetQueryExecution API アクションの AthenaError レスポンスオブジェクトに `ErrorType` サブフィールドが追加されました。

既存の `ErrorCategory` フィールドは、失敗したクエリの一般的なソース (システム、ユーザー、またはその他) を示しますが、新しい `ErrorType` フィールドには、発生したエラーに関するより詳細な情報が表示されます。両方のフィールドの情報を組み合わせて、クエリが失敗した原因に関するインサイトを把握します。

詳細については、「Athena エラーカタログ」を参照してください。

2022 年 2 月 9 日

公開日: 2022 年 2 月 9 日

古い Athena コンソールは利用できなくなりました。Athena の新しいコンソールは、以前のコンソールの機能をすべてサポートしています。それでいて使いやすく、最新のインターフェースを備えており、新機能によりクエリの開発、データの分析、使用状況管理の経験が向上します。新しい Athena コンソールを使用するには、https://console.aws.amazon.com/athena にアクセスします。

2022 年 2 月 8 日

公開日: 2022 年 2 月 8 日

バケット所有者 — 追加のセキュリティ対策として、クエリ結果を出力する場所となる Athena のバケットの、所有者 AWS アカウント の ID を、オプションで指定できるようになりました。クエリ結果でのバケット所有者のアカウント ID が、ここで指定したアカウント ID と一致しない場合、Amazon S3 のアクセス許可エラーにより、バケットに出力できません。この設定は、クライアントレベルまたはワークグループレベルで行えます。

詳細については、「クエリ結果の場所の指定」を参照してください。

2022 年 1 月 28 日

公開日: 2022 年 1 月 28 日

Athena では、次のエンジン機能の強化点を発表しました。

- Apache Hudi — Hudi Merge on Read (MoR) テーブルに対するスナップショットクエリで、`INT64` データ型のタイムスタンプ列の読み込みが可能になりました。
- UNION クエリ — パフォーマンスが向上し、同じテーブルを何度もスキャンするような特定の `UNION` クエリの、データスキャンを減らしました。
- クエリの分離 — フィルターの各パーティション列に対して、分離値のみを含むクエリの、パフォーマンスが向上しました。
- パーティション射影の強化
  - `injected` タイプの列に対するフィルター条件として、複数の分離値が許可されるようになりました。詳細については、「挿入型」を参照してください。
  - 分離値のみがフィルターに含まれ、`CHAR` や `VARCHAR` のような文字をベースとしたタイプの列の、パフォーマンスが向上しました。

2022 年 1 月 13 日

公開日: 2022 年 1 月 13 日

Athena 用の JDBC 2.0.27 ドライバー と ODBC 1.1.15 ドライバー がリリースされました。

JDBC 2.0.27 ドライバーには、次の変更が含まれています。

- 外部カタログを取得するようにドライバが更新されました。
- 拡張ドライバのバージョン番号は、Athena API コールの一部として、`user-agent` 文字列に含まれます。

ODBC 1.1.15 ドライバーには、次の変更が含まれています。

- `SQLParamData()` への 2 回目の呼び出しに関する問題を修正しました。

これらの変更についての詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

Athena リリースノート (2021 年)

2021 年 11 月 26 日

公開日: 2021 年 11 月 26 日

Athena は、Athena ACIDトランザクションのパブリックプレビューを発表しました。Athena の SQL データ操作言語 (DML) に書込み、削除、更新、およびタイムトラベルの操作を追加しました。Athena ACID トランザクションにより、複数のユーザーが、Amazon S3 データに対して信頼性の高い行レベルの変更を、同時に行えます。Apache Iceberg テーブルフォーマット上に構築された Athena ACID トランザクションは、Amazon EMR や Apache Spark など、Iceberg テーブルフォーマットをサポートしている他のサービスやエンジンと互換性があります。

使い慣れた SQL 構文で Athena ACID トランザクションを行うことで、ビジネスや規制に関するデータの更新が簡素化されます。たとえば、データの消去リクエストに応答する場合は、SQL の `DELETE` オペレーションを実行します。手動によるレコードの修正には、単一の `UPDATE` ステートメントを使用します。また、最近削除されたデータを回復する場合は、`SELECT` ステートメントにより、タイムトラベルクエリを発行します。Athena コンソール、API オペレーション、ODBC および JDBC ドライバー経由で、Athena のトランザクションを利用できます。

詳細については、「Athena ACID トランザクションの使用」を参照してください。

2021 年 11 月 24 日

公開日: 2021 年 11 月 24 日

Athena は、ZStandard で圧縮された ORC、Parquet、およびテキストファイルのデータに対する読み込みと書き込みをサポートすると発表しました。Athena では、ZStandard で圧縮されたデータを書き込むときに、ZStandard 圧縮レベル 3 を使用します。

Athena でのデータ圧縮の詳細については、「Athena での圧縮のサポート」を参照してください。

2021 年 11 月 22 日

公開日: 2021 年 11 月 22 日

これで Amazon Athena コンソールから AWS Step Functions ワークフローを管理する際に、スケーラブルなデータを処理するパイプラインの構築、カスタムのビジネスロジックに基づくクエリの実行、管理タスクやアラートタスクの自動化などが容易になります。

アップグレードされた Athena コンソール に Step Functions が統合され、これにより Athena を起動するステートマシンのインタラクティブなワークフロー図を表示できます。開始するには、左のナビゲーションパネルから [Workflows] (ワークフロー) を選択します。Athena クエリを持つ既存のステートマシンがある場合は、ステートマシンを選択して、インタラクティブなワークフロー図を表示します。Step Functions を初めて使用する場合は、まず Athena コンソールからサンプルプロジェクトを起動し、ユースケースに合わせてカスタマイズします。

詳細については、Amazon Athena と「AWS Step Functions を使用して ETL パイプラインの構築とオーケストレーションを行う」を参照するか、「Step Functions ドキュメント」を参照してください。

2021 年 11 月 18 日

公開日: 2021 年 11 月 18 日

Athena は、新機能と改善点を発表しました。

- 次の例で示すような、DISTINCT か ORDER BY、またその両方を含む集計クエリの、ディスクへのスピルをサポートします。

```
SELECT array_agg(orderstatus ORDER BY orderstatus)
FROM orders
```

```
GROUP BY orderpriority, custkey
```

- DISTINCT を使用するクエリの、メモリ処理に関する問題を修正しました。DISTINCT クエリを使用した際に、「Query exhausted resources at this scale factor」(このスケールファクターにより、クエリがリソースを使い果たしました) のようなメッセージが表示されないようにするには、DISTINCT のカーディナリティが低い行を選択するか、クエリのデータサイズを減らします。
- 特定の列を指定しない SELECT COUNT(*) クエリで、行バッファリングなしでカウントのみを保持することで、パフォーマンスとメモリ使用率が向上しました。
- 次の文字列関数が導入されました。
  - translate(source, from, to) — from の文字列を、対応する to の文字列に置き換えた、source 文字列を返します。from の文字列が重複していた場合、最初の文字列だけが使用されます。from 文字列の中に source の文字が無い場合、source の文字は翻訳されずにコピーされます。from 文字列にあるマッチした文字のインデックスが、to 文字列よりも長い場合、その文字は結果として返される文字列から省かれます。
  - concat_ws(string0, array(varchar)) — string0 を区切り文字として使用して、配列の要素を連結し、その結果を返します。string0 が null である場合、戻り値は null です。配列内の null 値はスキップされます。
- 欠落している struct のサブフィールドにアクセスしようとすると、クエリが失敗してしまうバグを修正しました。欠落しているサブフィールドについては、クエリが null を返すようになりました。
- 小数データ型のハッシュが矛盾する問題を修正しました。
- パーティション内の列が多すぎると、リソースを使い果たしてしまう問題を修正しました。

2021 年 11 月 17 日

公開日: 2021 年 11 月 17 日

Amazon Athena がパーティションのインデックス作成をサポートし、AWS Glue Data Catalog でパーティション化されたテーブルに対するクエリを高速化しました。

パーティション化されたテーブルをクエリする場合、Athena が利用可能なテーブルパーティションを取得し、クエリに関係のあるサブセットをフィルタリングします。新しいデータとパーティションが追加されると、パーティションの処理にかかる時間が長くなり、クエリランタイムも長くなります。Athena がAWS Glue パーティションインデックスをサポートし、パーティション処理が最適化され、高度にパーティション化されたテーブルでのクエリパフォーマンスが向上しました。

詳細については、「AWS Glue パーティションのインデックス作成とフィルタリング」を参照してください。

2021 年 11 月 16 日

公開日: 2021 年 11 月 16 日

新しくなって改善された Amazon Athenaコンソールが、Athena を利用可能な AWS 商用リージョンと GovCloud リージョンで、一般利用可能になりました。Athena の新しいコンソールは、以前のコンソールの機能をすべてサポートしています。それでいて使いやすく、最新のインターフェースを備えており、新機能によりクエリの開発、データの分析、使用状況管理の経験が向上します。これで、次の操作を実行できます。

- クエリタブバーが再設計され、複数のクエリタブを並べ替え、移動し、閉じることが可能になりました。
- SQL およびテキストの書式設定が改善され、クエリの読み込みと編集がより簡単になりました。
- 完全な結果セットのダウンロードに加え、クエリ結果をクリップボードにコピーできるようになりました。
- クエリ履歴、保存されたクエリ、ワークグループを並べ替え、表示または非表示にする列を選択できるようになりました。
- シンプルなインターフェイスのおかげで、データソースとワークグループをより少ないクリック数で設定できます。
- クエリ結果、クエリ履歴、行の折り返しなどの表示に関するプリファレンスを設定できます。
- キーボードショートカットが新しく改善され、また製品ドキュメントが組み込まれたことにより、生産性を向上できます。

本日の発表で、再設計されたコンソールがデフォルトの設定になりました。あなたの体験についてお伝え頂ける場合は、コンソールの左下隅にある [Feedback] (フィードバック) を選択してください。

必要な場合は、AWS アカウント にログインし、Amazon Athena を選択してから、左のナビゲーションパネルにある [New Athena experience] (新しい Athena エクスペリエンス) の選択を解除すると、以前のコンソールを使用できます。

2021 年 11 月 12 日

公開日: 2021 年 11 月 12 日

Amazon Athena を使用して、自分以外の AWS アカウントにあるデータソースに対し、フェデレーティッドクエリを実行できるようになりました。今日までは、このデータをクエリして、データをクエリしたユーザーと同じ AWS アカウント を使用するには、データソースとそのコネクタが必要でした。

データ管理者としては、データコネクタをデータアナリストのアカウントと共有することで、クロスアカウントのフェデレーティッドクエリを有効にできます。データアナリストとしては、データ管理者が共有してくれたデータコネクタをアカウントに追加できます。発信元アカウントのコネクタに対する設定変更は、共有されたコネクタに自動的に適用されます。

クロスアカウントフェデレーティッドクエリの有効化については、「クロスアカウントのフェデレーションクエリの有効化」を参照してください。フェデレーティッドソースに対するクエリについては、「Amazon Athena 横串検索の使用」および「フェデレーティッドクエリの実行」を参照してください。

2021 年 11 月 2 日

公開日: 2021 年 11 月 2 日

Athena で EXPLAIN ANALYZE ステートメントを使用して、SQL クエリで分散された実行プランと、それぞれのオペレーションコストを表示できるようになりました。

詳細については、「Athena での EXPLAIN および EXPLAIN ANALYZE の使用」を参照してください。

2021 年 10 月 29 日

公開日: 2021 年 10 月 29 日

Athena が JDBC 2.0.25 ドライバーと ODBC 1.1.13 ドライバーをリリースし、それぞれの機能と改善点を発表します。

JDBC ドライバーおよび ODBC ドライバー

Athena 用の JDBC 2.0.25 ドライバーと ODBC 1.1.13 ドライバーがリリースされました。どちらのドライバーも、任意の SAML 2.0 プロバイダで動作するように構成可能な、ブラウザでの SAML 多要素認証をサポートしています。

JDBC 2.0.25 ドライバーには、次の変更が含まれています。

- ブラウザでの SAML 認証をサポート。ドライバーには、任意の SAML 2.0 プロバイダーで動作するように構成可能な、ブラウザの SAML プラグインが含まれています。
- AWS Glue API コールのサポート。`GlueEndpointOverride` パラメータを使用して、AWS Glue エンドポイントをオーバーライドできます。
- `com.simba.athena.amazonaws` クラスパスを、`com.amazonaws` に変更しました。

ODBC 1.1.13 ドライバーには、次の変更が含まれています。

- ブラウザでの SAML 認証をサポート。ドライバーには、任意の SAML 2.0 プロバイダーで動作するように構成可能な、ブラウザの SAML プラグインが含まれています。ODBC ドライバーでブラウザの SAML プラグインを使用する例については、「ODBC、SAML 2.0、および Okta ID プロバイダを使用したシングルサインオンの設定」を参照してください。
- ADFS、Azure AD、または Browser Azure AD を認証に使用するときに、ロールのセッション期間を設定できるようになりました。

この詳細や、その他の変更について確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

特徴と改善点

Athena は、次の機能と改善点を発表しました。

- 特定のケースにおいて、テーブルスキャンが重複して行われることを回避するために、新しい最適化ルールが導入されました。

2021 年 10 月 4 日

公開日: 2021 年 10 月 4 日

Athena は、次の機能と改善点を発表しました。

- SQL OFFSET — SQLの `SELECT` ステートメントで、`OFFSET` 句がサポートされるようになりました。詳細については、「SELECT」を参照してください。
- CloudWatch の使用状況メトリクス — Athena が `AWS/Usage` 名前空間で、`ActiveQueryCount` メトリクスを発行するようになりました。詳細については、「Athena の使用状況に関するメトリクスのモニタリング」を参照してください。

- クエリ実行計画 — まれに、クエリ実行計画でタイムアウトを引き起こす可能性のあるバグを修正しました。

2021 年 9 月 16 日

公開日: 2021 年 9 月 16 日

Athena は、次の新機能と改善点を発表しました。

機能

- CTAS において、テキストファイルの指定と、`write_compression` テーブルプロパティを使用した圧縮 JSON の指定をサポートしました。Parquet 形式および ORC 形式の場合でも、CTAS で `write_compression` プロパティを指定できます。詳細については、「CTAS テーブルのプロパティ」を参照してください。
- テキストファイルと JSON ファイルへの書き込みで、BZIP2 圧縮形式がサポートされました。Athena での圧縮形式の詳細については、「Athena での圧縮のサポート」を参照してください。

改良点

- UDF の Lambda 関数への ID 情報送信が、失敗してしまうバグを修正しました。
- 分離フィルター条件での、述語プッシュダウンの問題を修正しました。
- 小数型のハッシュの問題を修正しました。
- 不要な統計を収集していた問題を修正しました。
- 一貫性のないエラーメッセージを削除しました。
- ワーカーノードで動的パーティションプルーニングを適用し、ブロードキャストジョインのパフォーマンスが向上しました。
- フェデレーティッドクエリ
  - フェデレーティッドクエリ内で `CONSTRAINT_VIOLATION` エラーの発生を削減するために設定を変更しました。

2021 年 9 月 15 日

公開日: 2021 年 9 月 15 日

再設計された Amazon Athena コンソール (プレビュー) を使用できるようになりました。新しい Athena JDBC ドライバーがリリースされました。

Athena コンソールのプレビュー

Athena が利用可能であれば、どの AWS リージョン からでも再設計された Amazon Athena コンソール (プレビュー) を使用できるようになりました。新しいコンソールは、既存のコンソールのすべての機能をサポートしていますが、より使いやすく、最新のインターフェイスをサポートしています。

新しいコンソールに切り替えるには、AWS アカウント にログインして、Amazon Athena を選択します。AWS コンソールのナビゲーションバーから、[Switch to the new console] (新しいコンソールに切り替える) を選択します。デフォルトのコンソールに戻るには、左のナビゲーションパネルにある [New Athena experience] (新しい Athena エクスペリエンス) の選択を解除します。

新しいコンソールのご利用を開始してください。左下隅にある [Feedback] (フィードバック) から、あなたの体験についてお伝えください。

Athena JDBC ドライバー 2.0.24

Athena は、Athena 用 JDBC ドライバーのバージョン 2.0.24 の利用開始を発表しました。このリリースでは、すべての認証情報プロバイダのプロキシサポートが更新されます。ドライバーは、`NonProxyHosts` 接続プロパティでサポートされていない、すべてのホストのプロキシ認証をサポートするようになりました。

便宜上、このリリースには、AWS SDK が付属したJDBC ドライバーと、付属しないドライバーの両方のダウンロードが含まれています。この JDBC ドライバーのバージョンには、AWS-SDK と Athena JDBC ドライバーの両方がプロジェクトに埋め込まれています。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2021 年 8 月 31 日

公開日: 2021 年 8 月 31 日

Athena では、次の機能強化とバグ修正が行われました。

- Athena フェデレーション機能の強化 — Athena は、Athena Query Federation SDK の一部として、マッピングタイプのサポートと、複合型に対するサポートを強化しました。このバージョンには、メモリの強化とパフォーマンスの最適化も含まれています。

- 新しいエラーカテゴリ — エラーメッセージに、USER と SYSTEM のエラーカテゴリが導入されました。これらのカテゴリは、自分で修正が可能なエラー (USER) なのか、Athena サポートから支援を受ける必要があるエラー (SYSTEM) なのかを区別するのに役立ちます。
- フェデレーティッドクエリのエラーメッセージ — フェデレーティッドクエリ関連エラーで、USER_ERROR のカテゴリー分けを更新しました。
- JOIN — ディスクへのスピル関連のバグとメモリの問題を修正し、JOIN オペレーションにおけるパフォーマンスを向上させ、メモリエラーを減らします。

2021 年 8 月 12 日

公開日: 2021 年 8 月 12 日

Athena 用の ODBC 1.1.12 ドライバーがリリースされました。このバージョンでは、SQLPrepare()、SQLGetInfo()、および EndpointOverride 関連の問題が修正されています。

新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「ODBC を使用した Amazon Athena への接続」を参照してください。

2021 年 8 月 6 日

公開日: 2021 年 8 月 6 日

Amazon Athena では、アジアパシフィック (大阪) リージョンで Athena とその機能が利用可能になりました。

このリリースでは、アジアパシフィックでの Athena の提供が拡張され、アジアパシフィック (香港)、アジアパシフィック (ムンバイ)、アジアパシフィック (大阪)、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)、およびアジアパシフィック (東京) も含まれるようになりました。これらのリージョンや他のリージョンで利用可能な AWS のサービス の詳細な一覧については、「AWS リージョン 別のサービス表」を参照してください。

2021 年 8 月 5 日

公開日: 2021 年 8 月 5 日

UNLOAD ステートメントを使用して、SELECT クエリの出力を PARQUET、ORC、AVRO、および JSON 形式に書き込めます。

詳細については、「UNLOAD」を参照してください。

2021 年 7 月 30 日

公開日: 2021 年 7 月 30 日

Athena では、次の機能強化とバグ修正が行われました。

- 動的フィルタリングとパーティションプルーニング - 次の例のように、これらの改善によってパフォーマンスが向上し、特定のクエリでスキャンされるデータ量が削減します。

  この例では、Table_B がパーティション化されていないテーブルで、ファイルサイズが 20 MB 未満であると想定しています。このようなクエリの場合、Table_A から読み込まれるデータがより少なくなり、クエリがより迅速に完了します。

  ```
  SELECT *
  FROM Table_A
  JOIN Table_B ON Table_A.date = Table_B.date
  WHERE Table_B.column_A = "value"
  ```

- ORDER BY と LIMIT、DISTINCT と LIMIT - ORDER BY または DISTINCT に続いて LIMIT 句を使用するクエリのパフォーマンスが改善されました。
- S3 Glacier Deep Archive ファイル - Athena が S3 Glacier Deep Archive ファイルと S3 Glacier 以外のファイルが混在するテーブルをクエリする場合、S3 Glacier Deep Archive ファイルをスキップするようになりました。以前は、これらのファイルをクエリの場所から手動で移動する必要がありました。これを行わないと、クエリが失敗します。Athena を使用して S3 Glacier Deep Archive ストレージ内のオブジェクトをクエリする場合は、それらを元に戻す必要があります。詳細については、「Amazon S3 ユーザーガイド」の「アーカイブされたオブジェクトの復元」を参照してください。
- CTAS bucketed_by テーブルプロパティによって作成された空のファイルが正しく暗号化されなかったバグを修正しました。

2021 年 7 月 21 日

公開日: 2021 年 7 月 21 日

2021 年 7 月にリリースされた Microsoft Power BI Desktop では、Amazon Athena のネイティブのデータソースコネクタを使用してレポートとダッシュボードを作成できます。Amazon Athena のコネクタは、Power BI の標準コネクタとして利用できます。また、DirectQuery をサポートし、Power BI ゲートウェイを介した大規模なデータセットの分析とコンテンツの更新を可能にします。

コネクタは、既存の ODBC データソース名 (DSN) を使用して Athena に接続してクエリを実行するため、Athena ODBC ドライバーが必要です。最新の ODBC ドライバーをダウンロードする方法は、「ODBC を使用した Amazon Athena への接続」を参照してください。

詳細については、「Amazon Athena Power BI コネクタの使用」を参照してください。

2021 年 7 月 16 日

公開日: 2021 年 7 月 16 日

Amazon Athena の Apache Hudi との統合を更新しました。Hudi は、Amazon S3 データレイクの増分データ処理を簡素化するのに使用されるオープンソースのデータ管理フレームワークです。統合を更新することにより、Amazon EMR、Apache Spark、Apache Hive、またはその他の互換性のあるサービスで管理される Hudi 0.8.0 テーブルを、Athena を使用してクエリできます。さらに、Athena は、読み込み時マージ (MoR) テーブルのスナップショットクエリと、ブートストラップされたテーブルの読み込みサポートの 2 つの追加機能をサポートするようになりました。

Apache Hudi は、変更データキャプチャ (CDC) パイプラインの開発の簡素化、GDPR 駆動の更新と削除への準拠、およびデータの挿入とイベントの更新を必要とするセンサーやデバイスからのストリーミングデータの管理に役立つレコードレベルのデータ処理を提供します。0.8.0 リリースでは、データをコピーせずに、大きな Parquet テーブルを Hudi に移行しやすくなり、Athena を介してクエリおよび分析できるようになります。Athena の新しいスナップショットクエリのサポートを使用して、ストリーミングテーブルの更新をほぼリアルタイムで表示できます。

Athena での Hudi の使用の詳細については、「Athena を使用した Apache Hudi データセットのクエリ」を参照してください。

2021 年 7 月 8 日

公開日: 2021 年 7 月 8 日

Athena 用の ODBC 1.1.11 ドライバーがリリースされました。ODBC ドライバーが JSON ウェブトークン (JWT) を使用して接続を認証できるようになりました。Linux では、Workgroup プロパティのデフォルト値は Primary に設定されています。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「ODBC を使用した Amazon Athena への接続」を参照してください。

2021 年 7 月 1 日

公開日: 2021 年 7 月 1 日

2021年7月1日、プレビューワークグループの特別な取り扱いが終了しました。AmazonAthenaPreviewFunctionality ワークグループの名前は保持されますが、特別なステータスはなくなります。引き続き AmazonAthenaPreviewFunctionality ワークグループを使用して、クエリを表示、変更、整理、実行できます。ただし、以前プレビューに含まれていた機能を使用するクエリには、標準の Athena 請求条件が適用されます。請求については、「Amazon Athena の料金」を参照してください。

2021 年 6 月 23 日

公開日: 2021 年 6 月 23 日

Athena 用の JDBC 2.0.23 ドライバー と ODBC 1.1.10 ドライバー がリリースされました。どちらのドライバーも、読み込み性能とサポートが向上し、EXPLAIN ステートメントおよびパラメータ化されたクエリをサポートしています。

EXPLAIN ステートメントは、SQL クエリの論理実行計画または分散実行計画を示します。パラメータ化されたクエリを使用すると、実行時に指定された異なる値を使用して、同じクエリを複数回使用できます。

JDBC リリースでは、アクティブディレクトリフェデレーションサービス 2019 のサポートと、AWS STS 用のカスタムエンドポイント優先オプションが追加されています。ODBC リリースでは、IAM プロファイル認証情報の問題を修正しています。

詳細を確認し、新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

2021 年 5 月 12 日

公開日: 2021 年 5 月 12 日

Amazon Athena を使用して、所有するアカウント以外のアカウントから AWS Glue を登録できるようになりました。AWS Glue に必要な IAM アクセス許可を設定したら、Athena を使用してクロスアカウントクエリを実行できます。

詳細については、別のアカウントからの AWS Glue Data Catalog の登録およびAWS Glue データカタログへのクロスアカウントアクセスを参照してください。

2021 年 5 月 10 日

公開日: 2021 年 5 月 10 日

Athena 用の ODBC ドライバーバージョン1.1.9.1001 がリリースされました。このバージョンでは、Azure Active Directory (AD) を使用した際に発生する `BrowserAzureAD` 認証タイプの問題を修正しています。

新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「ODBC を使用した Amazon Athena への接続」を参照してください。

2021 年 5 月 5 日

公開日: 2021 年 5 月 5 日

横串検索で Amazon Athena Vertica コネクタを使用して、Athena から Vertica データソースをクエリできるようになりました。例えば、Vertica 上のデータウェアハウス、および Amazon S3 内のデータレイクに対して分析クエリを実行できます。

Athena Vertica コネクタをデプロイするには、AWS Serverless Application Repository の「AthenaVerticaConnector」ページを参照してください。

Amazon Athena Vertica コネクタは、Lambda 環境変数を使用して、いくつかの設定オプションを公開します。設定オプション、パラメータ、接続文字列、デプロイ、および制限事項については、「Amazon Athena Vertica コネクタ」を参照してください。

Vertica コネクタの使用に関する詳細については、「AWS ビッグデータブログ」の「Athena 横串検索 SDK を使用して Amazon Athena で Vertica データソースをクエリする」を参照してください。

2021 年 4 月 30 日

公開日: 2021 年 4 月 30 日

Athena 用の JDBC ドライバー 2.0.21 と ODBC ドライバー 1.1.9 がリリースされました。両方のリリースで、Azure Active Directory (AD) での SAML 認証と、PingFederate での SAML 認証がサポートされています。JDBC リリースでは、パラメータ化されたクエリもサポートされています。Athena でのパラメータ化されたクエリの詳細については、「パラメータ化されたクエリの使用」を参照してください。

新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

2021 年 4 月 29 日

公開日: 2021 年 4 月 29 日

Amazon Athena が、中国 (北京) および中国 (寧夏) リージョンでの Athena エンジンバージョン 2 の提供開始を発表しました。

Athena エンジンバージョン 2 の詳細については、「Athena エンジンバージョン 2」を参照してください。

2021 年 4 月 26 日

公開日: 2021 年 4 月 26 日

Athena エンジンバージョン 2 のウィンドウ値関数が `IGNORE NULLS` と `RESPECT NULLS` をサポートするようになりました。

詳細については、Presto ドキュメントの「Value Functions」を参照してください。

2021 年 4 月 21 日

公開日: 2021 年 4 月 21 日

Amazon Athena が欧州 (ミラノ) およびアフリカ (ケープタウン) リージョンでの Athena エンジンバージョン 2 の提供開始を発表しました。

Athena エンジンバージョン 2 の詳細については、「Athena エンジンバージョン 2」を参照してください。

2021 年 4 月 5 日

公開日: 2021 年 4 月 5 日

EXPLAIN ステートメント

Athena で `EXPLAIN` ステートメントを使用して、SQL クエリの実行プランを表示できるようになりました。

詳細については、Athena での EXPLAIN および EXPLAIN ANALYZE の使用および Athena EXPLAIN ステートメントの結果についてを参照してください。

SQL クエリでの SageMaker 機械学習モデル

Amazon SageMaker を使用した機械学習モデル推論が、Amazon Athena で一般公開されました。SQL クエリで機械学習モデルを使用して、SQL クエリで関数を呼び出すことによる異常検知、カスタマーコーホート分析、および時系列予測などの複合型のタスクを簡素化します。

詳細については、「Machine Learning (ML) with Amazon Athena の使用」を参照してください。

ユーザー定義関数 (UDF)

ユーザー定義関数 (UDF) が Athena で一般公開されました。UDF を使用して、単一の SQL クエリでレコードまたはレコードのグループを処理するカスタム関数を活用します。

詳細については、「ユーザー定義関数を使用したクエリ」を参照してください。

2021 年 3 月 30 日

公開日: 2021 年 3 月 30 日

Amazon Athena が、アジアパシフィック (香港) および中東 (バーレーン) リージョンでの Athena エンジンバージョン 2 の提供開始を発表しました。

Athena エンジンバージョン 2 の詳細については、「Athena エンジンバージョン 2」を参照してください。

2021 年 3 月 25 日

公開日: 2021 年 3 月 25 日

Amazon Athena が、ヨーロッパ (ストックホルム) リージョンでの Athena エンジンバージョン 2 の提供開始を発表しました。

Athena エンジンバージョン 2 の詳細については、「Athena エンジンバージョン 2」を参照してください。

2021 年 3 月 5 日

公開日: 2021 年 3 月 5 日

Amazon Athena が、カナダ (中部)、欧州 (フランクフルト)、および南米 (サンパウロ) の各リージョンでの Athena エンジンバージョン 2 の提供開始を発表しました。

Athena エンジンバージョン 2 の詳細については、「Athena エンジンバージョン 2」を参照してください。

2021 年 2 月 25 日

公開日: 2021 年 2 月 25 日

Amazon Athena 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 (ロンドン)、および欧州 (パリ) の各リージョンでの Athena エンジンバージョン 2 の一般提供開始を発表しました。

Athena エンジンバージョン 2 の詳細については、「Athena エンジンバージョン 2」を参照してください。

Athena リリースノート (2020 年)

2020 年 12 月 16 日

公開日: 2020 年 12 月 16 日

Amazon Athena が、他のリージョンで Athena エンジンバージョン 2、Athena Federated Query、および AWS PrivateLink を提供することを発表しました。

Athena エンジンバージョン 2 と Athena 横串検索

Amazon Athena が、アジアパシフィック (ムンバイ)、アジアパシフィック (東京)、欧州 (アイルランド)、および米国西部 (北カリフォルニア) の各リージョンでの Athena エンジンバージョン 2 および Athena 横串検索の一般提供開始を発表しました。Athena エンジンバージョン 2 と横串検索は、すでに米国東部 (バージニア北部)、米国東部 (オハイオ)、および米国西部 (オレゴン) の各リージョンで利用可能です。

詳細については、Athena エンジンバージョン 2および Amazon Athena 横串検索の使用を参照してください。

AWS PrivateLink

欧州 (ストックホルム) リージョンで Athena の AWS PrivateLink がサポートされるようになりました。Athena の AWS PrivateLink については、「インターフェイス VPC エンドポイントを使用して Amazon Athena に接続する」を参照してください。

2020 年 11 月 24 日

公開日: 2020 年 11 月 24 日

Athena 用の JDBC ドライバー 2.0.16 と ODBC ドライバー 1.1.6 がリリースされました。これらのリリースでは、Okta Verify 多要素認証 (MFA) をアカウントレベルでサポートします。また、Okta MFA を使用して SMS 認証と Google 認証システムの認証を要素として設定することもできます。

新しいドライバー、リリースノート、およびドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」および「ODBC を使用した Amazon Athena への接続」を参照してください。

2020 年 11 月 11 日

公開日: 2020 年 11 月 11 日

Amazon Athena が、米国東部 (バージニア北部)、米国東部 (オハイオ)、米国東部 (オハイオ)、および米国西部 (オレゴン) の各リージョンでの Athena エンジンバージョン 2 および横串検索の一般提供開始を発表しました。

Athena エンジンバージョン 2

Amazon Athena が、新しいクエリエンジンバージョンである Athena エンジンバージョン 2 の米国東部 (バージニア北部)、米国東部 (オハイオ)、および米国西部 (オレゴン) の各リージョンでの一般提供開始を発表しました。

Athena エンジンバージョン 2 には、Parquet 形式のデータに対するスキーマ進化のサポート、追加の地理空間関数、ネストされたスキーマの読み込みによるコスト削減のサポート、および JOIN と AGGATE オペレーションにおけるパフォーマンス強化などのパフォーマンス面での改善と新しい機能が含まれています。

- 改善点、最新の変更、およびバグ修正については、「Athena エンジンバージョン 2」を参照してください。
- 更新方法については、「Athena エンジンバージョンの変更」を参照してください。
- クエリのテストについては、「エンジンバージョンをアップグレードする前のクエリのテスト」を参照してください。

フェデレーティッド SQL クエリ

米国東部 (バージニア北部)、米国東部 (オハイオ)、および米国西部 (オレゴン) の各リージョンで、`AmazonAthenaPreviewFunctionality` ワークグループを使用することなく Athena のフェデレーティッドクエリを利用できるようになりました。

フェデレーティッド SQL クエリを使用して、リレーショナル、非リレーショナル、オブジェクト、カスタムの各データソース間で SQL クエリを実行します。横串検索を使用すると、ユーザーは単一の SQL クエリを送信することで、オンプレミスで実行されているか、クラウド内でホストされている複数のソースのデータをスキャンできます。

複数のアプリケーションに分散されたデータに対する分析の実行は、以下の理由により、複合型で時間がかかる場合があります。

- 分析に必要なデータは、通常、リレーショナル、キーバリュー、ドキュメント、インメモリ、検索、グラフ、オブジェクト、時系列、および台帳の各データストアに分散されています。
- これらのソース間のデータを分析するために、分析では複合型のパイプラインを構築してデータを抽出、変換し、データウェアハウスにロードすることで、データにクエリを実行できるようにします。
- さまざまなソースのデータにアクセスするには、新しいプログラミング言語とデータアクセスコンストラクトを学習する必要があります。

Athena のフェデレーティッド SQL クエリは、データの保管場所にかかわらず、ユーザーがインプレースでデータをクエリできるようにすることで、このような複合型を排除します。アナリストは、使い慣れた SQL コンストラクトを使用して複数のデータソース間でデータを `JOIN` してすばやく分析し、後で使用できるようにその結果を Simple Storage Service (Amazon S3) に保存できます。

データソースコネクタ

横串検索を処理するために、Athena は AWS Lambda で実行される Athena データソースコネクタを使用します。以下の事前構築されたオープンソースコネクタは、Athena によって記述され、テストされたものです。これらを使用して、対応するデータソースに対する SQL クエリを Athena で実行します。

- CloudWatch
- CloudWatch Metrics
- DocumentDB
- DynamoDB
- OpenSearch
- HBase
- Neptune
- Redis
- Timestream
- TPC ベンチマーク DS (TPC-DS)

カスタムデータソースコネクタ

Athena Query Federation SDK を使用することによって、デベロッパーは任意のデータソースへのコネクタを構築して、Athena がそのデータソースに対して SQL クエリを実行できるようにすることが可能です。Athena Query Federation コネクタは、AWS が提供するコネクタよりも幅広いフェデレーティッドクエリの利点を提供します。コネクタは AWS Lambda で実行されるため、インフラストラクチャを管理したり、ピーク需要に合わせてスケーリングを計画したりする必要はありません。

次のステップ

- フェデレーティッドクエリ機能の詳細については、「Amazon Athena 横串検索の使用」を参照してください。
- 既存のコネクタの使用を開始するには、「コネクタのデプロイとデータソースへの接続」を参照してください。
- Athena Query Federation SDK を使用して独自のデータソースコネクタを構築する方法については、GitHub の「Example Athena Connector」を参照してください。

2020 年 10 月 22 日

公開日: 2020 年 10 月 22 日

これで AWS Step Functions で Athena を呼び出せます。AWS Step Functions は、Amazon States Language を直接使用して特定の AWS のサービス を制御できます。Step Functions を Athena で使用して、クエリの実行の開始と停止、クエリ結果の取得、アドホックまたはスケジュールされたデータクエリの実行、Simple Storage Service (Amazon S3) 内のデータレイクからの結果の取得を行うことができます。

詳細については、「AWS Step Functions デベロッパーガイド」の「Step Functions で Athena を呼び出す」を参照してください。

2020 年 7 月 29 日

公開日: 2020 年 7 月 29 日

JDBC ドライバーバージョン 2.0.13 がリリースされました。このリリースは、Athena に登録された複数のデータカタログ、認証のための Okta サービス、および VPC エンドポイントへの接続の使用をサポートします。

新しいバージョンのドライバーをダウンロードして使用するには、「JDBC を使用した Amazon Athena への接続」を参照してください。

2020 年 7 月 9 日

公開日: 2020 年 7 月 9 日

Amazon Athena が、圧縮された Hudi データセットのクエリに対するサポートと、Athena に登録するデータカタログを作成、更新、または削除するための AWS CloudFormation の `AWS::Athena::DataCatalog` リソースを追加しました。

Apache Hudi データセットのクエリ

Apache Hudi は、増分データ処理を簡素化するオープンソースのデータ管理フレームワークです。Amazon Athena が、Amazon S3 ベースのデータレイク内にある Apache Hudi データセットの読み込み最適化ビューのクエリをサポートするようになりました。

詳細については、「Athena を使用した Apache Hudi データセットのクエリ」を参照してください。

AWS CloudFormation データカタログのリソース

Amazon Athena の横串検索機能を使用してデータソースをクエリするには、まずデータカタログを Athena に登録する必要があります。AWS CloudFormation の `AWS::Athena::DataCatalog` リソースを使用して、Athena に登録するデータカタログを作成、更新、または削除でき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::Athena::DataCatalog」を参照してください。

2020 年 6 月 1 日

公開日: 2020 年 6 月 1 日

Amazon Athena での Apache Hive メタストアのメタカタログとしての使用

Athena での AWS Glue Data Catalog に加えて、Athena を 1 つ以上の Apache Hive メタストアに接続できるようになりました。

自己ホスト型の Hive メタストアに接続するには、Athena Hive メタストアコネクタが必要です。Athena は、これに使用できるリファレンス実装コネクタを提供します。コネクタは、ユーザーのアカウントで AWS Lambda 関数として動作します。

詳細については、「外部 Hive メタストア用の Athena データコネクタの使用」を参照してください。

2020 年 5 月 21 日

公開日: 2020 年 5 月 21 日

Amazon Athena がパーティション射影のサポートを追加しました。パーティション射影を使用して、高度にパーティションされたテーブルのクエリ処理を高速化し、パーティション管理を自動化します。詳細については、「Amazon Athena でのパーティション射影」を参照してください。

2020 年 4 月 1 日

公開日: 2020 年 4 月 1 日

米国東部 (バージニア北部) リージョンに加えて、アジアパシフィック (ムンバイ)、欧州 (アイルランド)、および米国西部 (オレゴン) の各リージョンでも Amazon Athena のフェデレーティッドクエリ、ユーザー定義関数 (UDF)、機械学習推論、および外部 Hive メタストア機能をプレビューできるようになりました。

2020 年 3 月 11 日

公開日: 2020 年 3 月 11 日

Amazon Athena が、クエリの状態推移に関する Amazon EventBridge イベントをパブリッシュするようになりました。クエリの状態が推移 (例えば、「成功」または「キャンセル済」など、「実行中」から最終状態に推移) すると、Athena はクエリ状態変更イベントを EventBridge にパブリッシュします。イベントには、クエリ状態移行に関する情報が含まれます。詳細については、「Amazon EventBridge イベントで Athena クエリを監視する」を参照してください。

2020 年 3 月 6 日

公開日: 2020 年 3 月 6 日

AWS CloudFormation の `AWS::Athena::WorkGroup` リソースを使用して、Amazon Athena ワークグループを作成し、更新でき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::Athena::WorkGroup」を参照してください。

Athena リリースノート (2019 年)

2019 年 11 月 26 日

公開日: 2019 年 12 月 17 日

Amazon Athena が、リレーショナル、非リレーショナル、オブジェクト、およびカスタムの各データソース間における SQL クエリの実行、SQL クエリにおける機械学習モデルの呼び出し、ユーザー定義関数 (UDF) (プレビュー)、メタデータカタログとしての Apache Hive メタストアの Amazon Athena での使用 (プレビュー)、および 4 つの追加のクエリ関連メトリクスに対するサポートを追加しました。

フェデレーティッド SQL クエリ

フェデレーティッド SQL クエリを使用して、リレーショナル、非リレーショナル、オブジェクト、カスタムの各データソース間で SQL クエリを実行します。

Athena のフェデレーティッドクエリを使用して、リレーショナル、非リレーショナル、オブジェクト、およびカスタムの各データソースに保存されているデータをスキャンできるようになりました。横串検索を使用すると、ユーザーは単一の SQL クエリを送信することで、オンプレミスで実行されているか、クラウド内でホストされている複数のソースのデータをスキャンできます。

複数のアプリケーションに分散されたデータに対する分析の実行は、以下の理由により、複合型で時間がかかる場合があります。

- 分析に必要なデータは、通常、リレーショナル、キーバリュー、ドキュメント、インメモリ、検索、グラフ、オブジェクト、時系列、および台帳の各データストアに分散されています。
- これらのソース間のデータを分析するために、分析では複合型のパイプラインを構築してデータを抽出、変換し、データウェアハウスにロードすることで、データにクエリを実行できるようにします。
- さまざまなソースのデータにアクセスするには、新しいプログラミング言語とデータアクセスコンストラクトを学習する必要があります。

Athena のフェデレーティッド SQL クエリは、データの保管場所にかかわらず、ユーザーがインプレースでデータをクエリできるようにすることで、このような複合型を排除します。アナリストは、使い慣れた SQL コンストラクトを使用して複数のデータソース間でデータを `JOIN` してすばやく分析し、後で使用できるようにその結果を Simple Storage Service (Amazon S3) に保存できます。

データソースコネクタ

Athena は、AWS Lambda で実行される Athena データソースコネクタを使用して横串検索を処理します。これらのオープンソースのデータソース コネクタを使用して、Amazon DynamoDB、Apache HBase、Amazon Document DB、Amazon CloudWatch、Amazon CloudWatch Metrics、および JDBC 準拠のリレーショナルデータベース (MySQL や Apache 2.0 ライセンスに基づく PostgreSQL など) 間でのフェデレーティッド SQL クエリを Athena で実行します。

カスタムデータソースコネクタ

Athena Query Federation SDK を使用することによって、デベロッパーは任意のデータソースへのコネクタを構築して、Athena がそのデータソースに対して SQL クエリを実行できるようにすることが可能です。Athena Query Federation コネクタは、AWS が提供するコネクタよりも幅広いフェデレーティッドクエリの利点を提供します。コネクタは AWS Lambda で実行されるため、インフラストラクチャを管理したり、ピーク需要に合わせてスケーリングを計画したりする必要はありません。

プレビューの可用性

Athena 横串検索は、米国東部 (バージニア北部) リージョンでプレビューできます。

次のステップ

- プレビューを開始するには、「Athena プレビュー版の機能に関するよくある質問」の指示に従ってください。
- 横串検索機能の詳細については、「Amazon Athena 横串検索の使用 (プレビュー)」を参照してください。
- 既存のコネクタの使用を開始するには、「コネクタのデプロイとデータソースへの接続」を参照してください。
- Athena Query Federation SDK を使用して独自のデータソースコネクタを構築する方法については、GitHub の「Example Athena Connector」を参照してください。

SQL クエリでの機械学習モデルの呼び出し

推論用の機械学習モデルを Athena クエリから直接呼び出せるようになりました。SQL クエリで機械学習モデルを使用できる場合、異常検知、カスタマーコーホート分析、売上予測などの複合型のタスクが、SQL クエリで機能を呼び出すのと同じくらいシンプルになります。

ML モデル

Amazon SageMaker が提供する多数の組み込み型機械学習アルゴリズムを使用すると、独自のモデルをトレーニングするか、AWS Marketplace でモデルパッケージを見つけてサブスクライブして、これらのモデルを Amazon SageMaker ホスティングサービスにデプロイできます。追加の設定は不要です。これらの ML モデルは、Athena コンソール、Athena APIs、および Athena のプレビュー JDBC ドライバーから SQL クエリで呼び出せます。

プレビューの可用性

本日から米国東部 (バージニア北部) リージョンで Athena の ML 機能をプレビューできます。

次のステップ

- プレビューを開始するには、「Athena プレビュー版の機能に関するよくある質問」の指示に従ってください。
- 機械学習機能の詳細については、「Amazon Athena での機械学習 (ML) の使用 (プレビュー)」を参照してください。

ユーザー定義関数 (UDF) (プレビュー)

カスタムスカラー関数を記述して、それらを Athena クエリで呼び出せるようになりました。UDF は、Athena Query Federation SDK を使用して Java で記述できます。Athena に送信された SQL クエリで UDF が使用されていると、この UDF は AWS Lambda で呼び出され、実行されます。UDF は、SQL クエリの `SELECT` 句と `FILTER` 句の両方で使用できます。複数の UDF を同じクエリで呼び出せます。

プレビューの可用性

Athena の UDF 機能は、米国東部 (バージニア北部) リージョンでプレビューモードで使用できます。

次のステップ

- プレビューを開始するには、「Athena プレビュー版の機能に関するよくある質問」の指示に従ってください。
- 詳細については、「ユーザー定義関数を使用したクエリ (プレビュー)」を参照してください。
- UDF 実装の例については、GitHub の「Amazon Athena UDF Connector」を参照してください。
- Athena Query Federation SDK を使用して独自の関数を記述する方法については、「Lambda を使用した UDF の作成とデプロイ」を参照してください。

Amazon Athena での Apache Hive メタストアのメタカタログとしての使用 (プレビュー)

Athena での AWS Glue Data Catalog に加えて、Athena を 1 つ以上の Apache Hive メタストアに接続できるようになりました。

メタストアコネクタ

自己ホスト型の Hive メタストアに接続するには、Athena Hive メタストアコネクタが必要です。Athena は、これに使用できるリファレンス実装コネクタを提供します。コネクタは、ユーザーのアカウントで AWS Lambda 関数として動作します。詳細については、「外部 Hive メタストアの Athena データコネクターの使用 (プレビュー)」を参照してください。

プレビューの可用性

Hive メタストア機能は、米国東部 (バージニア北部) リージョンでプレビューモードで使用できます。

次のステップ

- プレビューを開始するには、「Athena プレビュー版の機能に関するよくある質問」の指示に従ってください。
- この機能の詳細については、「外部 Hive メタストアの Athena データコネクターの使用 (プレビュー)」を参照してください。

新しいクエリ関連メトリクス

Athena が、Amazon Athena のパフォーマンスを理解するうえで役立つ、追加のクエリメトリクスをパブリッシュするようになりました。Athena は、クエリ関連のメトリクスを Amazon CloudWatch にパブリッシュします。このリリースでは、Athena が以下の追加のクエリメトリクスをパブリッシュします。

- Query Planning Time – クエリを計画するためにかかった時間。これには、データソースからテーブルパーティションを取得するためにかかった時間も含まれます。
- Query Queuing Time – クエリがキュー内でリソースを待機していた時間。
- Service Processing Time – クエリエンジンが処理を完了してから結果が書き込まれるまでにかかった時間。
- Total Execution Time – Athena がクエリを実行するために費やした時間。

これらの新しいクエリメトリクスを使用するには、カスタムダッシュボードを作成する、CloudWatch のメトリクスにアラームとトリガーを設定する、または Athena コンソールから事前入力済みのダッシュボードを直接使用できます。

次のステップ

詳細については、「CloudWatch メトリクスを使用した Athena クエリのモニタリング」を参照してください。

2019 年 11 月 12 日

公開日: 2019 年 12 月 17 日

Amazon Athena が、中東 (バーレーン) リージョンで利用可能になりました。

2019 年 11 月 8 日

公開日: 2019 年 12 月 17 日

Amazon Athena が米国西部 (北カリフォルニア) リージョンと欧州 (パリ) リージョンで利用可能になりました。

2019 年 10 月 8 日

公開日: 2019 年 12 月 17 日

Amazon Athena が、Virtual Private Cloud (VPC) のインターフェイス VPC エンドポイント経由での Athena への直接接続を許可するようになりました。この機能を使用することで、VPC にインターネットゲートウェイを設定する必要なく、安全にクエリを Athena に送信できます。

Athena に接続するためのインターフェイス VPC エンドポイントを作成するには、AWS Management Console または AWS Command Line Interface (AWS CLI) を使用します。インターフェイスエンドポイントの作成については、「インターフェイスエンドポイントの作成」を参照してください。

インターフェイス VPC エンドポイントを使用すると、VPC と Athena API 間の通信がセキュア化され、AWS ネットワーク内にとどまります。この機能を使用するために、追加の Athena 料金は発生しません。インターフェイス VPC エンドポイントの料金が適用されます。

この機能の詳細については、「インターフェイス VPC エンドポイントを使用して Amazon Athena Connect する」を参照してください。

2019 年 9 月 19 日

公開日: 2019 年 12 月 17 日

Amazon Athena では、INSERT INTO ステートメントを使用して既存のテーブルに新しいデータを挿入できるようになりました。ソーステーブルで実行される SELECT クエリステートメント、またはクエリステートメントの一部として提供される値のセットに基づいて、ターゲットテーブルに新しい行を挿入できます。データ形式として Avro、JSON、ORC、Parquet、テキストファイルなどがサポートされています。

INSERT INTO ステートメントを使用して ETL プロセスを簡素化することもできます。たとえば、単一のクエリで INSERT INTO を使用することで、JSON 形式のソーステーブルからデータを選択して Parquet 形式のターゲットテーブルに書き込めます。

INSERT INTO ステートメントは、Athena での SELECT クエリの場合と同様に、SELECT フェーズでスキャンしたバイト数に基づいて課金されます。詳細については、「Amazon Athena の料金」を参照してください。

サポートされている形式、SerDes、および例などの INSERT INTO の使用に関する詳細については、Athena ユーザーガイドの「INSERT INTO」を参照してください。

2019 年 9 月 12 日

公開日: 2019 年 12 月 17 日

Amazon Athena が、アジアパシフィック (香港) リージョンで利用可能になりました。

2019 年 8 月 16 日

公開日: 2019 年 12 月 17 日

Amazon Athena が、Amazon S3 リクエスタ支払いバケット内のデータに対するクエリのサポートを追加しました。

Amazon S3 バケットがリクエスタ支払いとして設定されている場合、Amazon S3 リクエストとデータ転送のコストは、バケット所有者ではなくリクエスタが支払います。Athena では、ワークグループ管理者が、ワークグループメンバーによる S3 リクエスタ支払いバケットへのクエリを許可するようにワークグループを設定できるようになりました。

ワークグループのリクエスタ支払い設定を行う方法については、Amazon Athena ユーザーガイドの「ワークグループの作成」を参照してください。リクエスタ支払いバケットの詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「リクエスタ支払いバケット」を参照してください。

2019 年 8 月 9 日

公開日: 2019 年 12 月 17 日

Amazon Athena が、Simple Storage Service (Amazon S3) に保存されたデータ用の AWS Lake Formation で定義された、新しい、または既存のデータベース、テーブル、および列に対するきめ細かなアクセスコントロールのための AWS Glue Data Catalog ポリシーの実施をサポートするようになりました。

この機能は、米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、および欧州 (アイルランド) の各 AWS リージョン で利用できます。この機能の使用に追加料金は発生しません。

この機能の使用方法については、「Athena を使用して AWS Lake Formation に登録されたデータをクエリする」を参照してください。の詳細については、「AWS Lake Formation」を参照してください。AWS Lake Formation

2019 年 6 月 26 日

Amazon Athena が欧州 (ストックホルム) リージョンで利用可能になりました。サポートされているリージョンのリストについては、「AWS リージョン およびエンドポイント」を参照してください。

2019 年 5 月 24 日

公開日: 2019 年 05 月 24 日

Amazon Athena が AWS GovCloud (米国東部) および AWS GovCloud (米国西部) の各リージョンで利用可能になりました。サポートされているリージョンのリストについては、AWS リージョン およびエンドポイントを参照してください。

2019 年 3 月 5 日

公開日: 2019 年 03 月 05 日

Amazon Athena がカナダ (中部) リージョンで利用可能になりました。サポートされているリージョンのリストについては、AWS リージョン およびエンドポイントを参照してください。Athena ワークグループをサポートする新しいバージョンの ODBC ドライバーがリリースされました。詳細については、「ODBC ドライバーリリースノート」を参照してください。

ODBC ドライバーのバージョン 1.0.5 とそのドキュメントをダウンロードするには、「ODBC を使用した Amazon Athena への接続」を参照してください。このバージョンの詳細については、「ODBC ドライバーリリースノート」を参照してください。

ODBC ドライバーでワークグループを使用するには、次の例に示すように、接続文字列に新しい接続プロパティ `Workgroup` を設定します。

```
Driver=Simba Athena ODBC
 Driver;AwsRegion=[Region];S3OutputLocation=[S3Path];AuthenticationType=IAM
 Credentials;UID=[YourAccessKey];PWD=[YourSecretKey];Workgroup=[WorkgroupName]
```

詳細については、「ODBC ドライバーのインストールおよび設定ガイドバージョン 1.0.5」で「workgroup」を検索してください。ワークグループでタグを使用しても、ODBC ドライバーの接続文字列は変更されません。タグを使用するには、最新バージョンの ODBC ドライバー (この最新バージョン) にアップグレードしてください。

このドライバーバージョンでは、Athena API ワークグループアクションを使用してワークグループを作成および管理し、Athena API タグアクションを使用してワークグループのタグを追加、リスト化、または削除できます。使用を開始する前に、ワークグループとタグに対するアクションに関するリソースレベルの IAM 許可があることを確認してください。

詳細については、以下を参照してください。

- クエリを実行するためのワークグループの使用「」および「ワークグループのポリシーの例」。
- Athena リソースへのタグ付け「」および「タグベースの IAM アクセスコントロールポリシー」。

JDBC ドライバー、または AWS SDK を使用している場合は、最新バージョンのドライバーと SDK にアップグレードしてください。どちらにも、Athena のワークグループとタグのサポートが既に含まれています。詳細については、「JDBC を使用した Amazon Athena への接続」を参照してください。

2019 年 2 月 22 日

公開日: 2019 年 02 月 22 日

Amazon Athena のワークグループにタグのサポートを追加しました。タグは、1 つのキーと 1 つの値で構成されており、どちらもお客様側が定義します。ワークグループにタグを付けるときは、カスタムメタデータを割り当てます。AWS タグ付けのベストプラクティスを使用して、ワークグループにタグを追加してそれらを分類できます。タグを使用してワークグループへのアクセスを制限し、コ

ストを追跡できます。たとえば、コストセンターごとにワークグループを作成します。その後、これらのワークグループにタグを追加することで、各コストセンターの Athena 支出を追跡できます。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「請求へのタグの使用」を参照してください。

タグは、Athena コンソール、または API オペレーションを使用することで操作できます。詳細については、「Athena リソースへのタグ付け」を参照してください。

Athena コンソールでは、各ワークグループに 1 つ、または複数のタグを追加して、それらをタグで検索できます。ワークグループは、Athena にある IAM で管理されるリソースです。IAM では、作成したワークグループのタグを追加、削除、またはリスト化できるユーザーを制限できます。ワークグループに 1 つ以上のタグを追加するためのオプションのタグパラメータを持つ `CreateWorkGroup` API オペレーションを使用することもできます。タグを追加、削除、または一覧表示するには、`TagResource`、`UntagResource`、および `ListTagsForResource` を使用します。詳細については、「タグオペレーションの使用」を参照してください。

ワークグループの作成時にユーザーがタグを追加できるようにするには、各ユーザーに `TagResource` と `CreateWorkGroup` の API アクションの両方に対する IAM 許可を付与するようにしてください。詳細な説明と例については、「タグベースの IAM アクセスコントロールポリシー」を参照してください。

ワークグループでタグを使用しても、JDBC ドライバーは変更されません。新しいワークグループを作成して JDBC ドライバーまたは AWS SDK を使用する場合は、最新バージョンのドライバーと SDK に更新してください。詳細については、JDBC を使用した Amazon Athena への接続 を参照してください。

2019 年 2 月 18 日

公開日: 2019 年 02 月 18 日

ワークグループでクエリを実行することによってクエリコストを制御する機能が追加されました。詳細については、ワークグループを使用してクエリのアクセスとコストを制御する を参照してください。Athena で使用される JSON OpenX SerDe を改善し、Athena が `GLACIER` ストレージクラスに移行されたオブジェクトを無視しないという問題を修正しました。また、Network Load Balancer ログをクエリする例を追加しました。

以下の変更を加えました。

- ワークグループ用のサポートを追加しました。ワークグループを使用して、ユーザー、チーム、アプリケーション、またはワークロードを分離し、各クエリまたはワークグループ全体で処理で

きるデータ量に制限を設定します。ワークグループは IAM リソースとして機能するため、リソースレベルのアクセス許可を使用して特定のワークグループへのアクセスを制御できます。Amazon CloudWatch でクエリ関連のメトリクスを表示する、スキャンされるデータの量に制限を設定することによってクエリコストを管理する、しきい値を作成する、およびこれらのしきい値を超えたときに Amazon SNS アラームなどのアクションをトリガーすることもできます。詳細については、クエリを実行するためのワークグループの使用およびCloudWatch のメトリクスとイベントを使用したコストの管理とクエリのモニタリングを参照してください。

ワークグループは、IAM リソースです。IAM のワークグループ関連のアクション、リソース、および条件の完全なリストについては、「サービス認証リファレンス」の「Amazon Athena 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。新しいワークグループを作成する前に、ワークグループの IAM ポリシーと AWS 管理ポリシー: AmazonAthenaFullAccess を使用するようにしてください。

コンソール、ワークグループ API オペレーション、または JDBC ドライバーを使用して、ワークグループの使用を開始できます。大まかな手順については、「ワークグループのセットアップ」を参照してください。ワークグループサポート付きの JDBC ドライバーをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。

JDBC ドライバーでワークグループを使用する場合は、次の例のように `Workgroup` 設定パラメータを使用して接続文字列にワークグループ名を設定する必要があります。

```
jdbc:awsathena://AwsRegion=<AWSREGION>;UID=<ACCESSKEY>;
PWD=<SECRETKEY>;S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/<athena-
output>-<AWSREGION>/;
Workgroup=<WORKGROUPNAME>;
```

SQL ステートメントの実行方法やドライバーへの JDBC API 呼び出しの方法に変更はありません。ドライバーがワークグループ名を Athena に渡します。

ワークグループに導入された変更点については、「 Athena ワークグループ API」および「ワークグループのトラブルシューティング」を参照してください。

- Athena で使用される JSON OpenX SerDe を改善しました。改善点は次のとおりです。
  - `ConvertDotsInJsonKeysToUnderscores` プロパティをサポートします。`TRUE` に設定すると、SerDe はキー名のドットをアンダースコアに置き換えることができます。たとえば、JSON データセットに `"a.b"` という名前のキーが含まれている場合は、このプロパティを使用して、Athena で列名が `"a_b"` になるように定義できます。デフォルト: `FALSE`。デフォルトで、Athena は列名にドットを許可しません。

- `case.insensitive` プロパティをサポートします。デフォルトで、Athena では JSON データセットのすべてのキーに小文字が使用される必要があります。`WITH SERDE PROPERTIES ("case.insensitive"= FALSE;)` を使用すると、データ内で大文字と小文字を区別してキー名を使用できます。デフォルト: `TRUE`。`TRUE` に設定すると、SerDe はすべての大文字の列を小文字に変換します。

  詳細については、「OpenX JSON SerDe」を参照してください。
- Simple Storage Service (Amazon S3) ライフサイクルポリシーによって Glacier にアーカイブされた Amazon S3 オブジェクトを処理したときに、Athena が `"access denied"` エラーメッセージを返す問題を修正しました。この問題を解決した結果、Athena が `GLACIER` ストレージクラスに移行したオブジェクトを無視するようになりました。Athena では、`GLACIER` ストレージクラスからのデータのクエリがサポートされていません。

  詳細については、「the section called “Athena のテーブルと Amazon S3 のデータに関する要件”」および「Amazon Simple Storage Service ユーザーガイド」の「Transitioning to the GLACIER Storage Class (Object Archival)」(GLACIER ストレージクラスへの移行 (オブジェクトのアーカイブ)) を参照してください。
- Transport Layer Security (TLS) リクエストに関する情報を受け取る Network Load Balancer アクセスログのクエリ例を追加しました。詳細については、「the section called “Network Load Balancer”」を参照してください。

Athena リリースノート (2018 年)

2018 年 11 月 20 日

公開日: 2018 年 11 月 20 日

AD FS および SAML 2.0 (Security Assertion Markup Language 2.0) を使用した Athena API へのフェデレーションアクセスをサポートする新しいバージョンの JDBC および ODBC ドライバーをリリースしました。詳細については、「JDBC ドライバーリリースノート」および「ODBC ドライバーリリースノート」を参照してください。

このリリースによって、Athena へのフェデレーションアクセスが Active Directory Federation Service (AD FS 3.0) 対応になります。アクセスは、SAML 2.0 をサポートするバージョンの JDBC や ODBC ドライバーを通じて確立されます。Athena API へのフェデレーションアクセスの設定に関する詳細については、「the section called “Athena API へのフェデレーションアクセスの有効化”」を参照してください。

JDBC ドライバーのバージョン 2.0.6 とそのドキュメントをダウンロードするには、「JDBC を使用した Amazon Athena への接続」を参照してください。このバージョンの詳細については、「JDBC ドライバーリリースノート」を参照してください。

ODBC ドライバーのバージョン 1.0.4 とそのドキュメントをダウンロードするには、「ODBC を使用した Amazon Athena への接続」を参照してください。このバージョンの詳細については、「ODBC ドライバーリリースノート」を参照してください。

AWS での SAML 2.0 のサポートに関する詳細については、「IAM ユーザーガイド」の「SAML 2.0 ベースのフェデレーションについて」を参照してください。

2018 年 10 月 15 日

公開日: 2018 年 10 月 15 日

AWS Glue Data Catalog にアップグレードしている場合は、以下のサポートを提供する 2 つの新機能があります。

- データカタログメタデータの暗号化。データカタログでメタデータを暗号化することを選択する場合は、Athena に特定のポリシーを追加する必要があります。詳細については、「AWS Glue Data Catalog の暗号化されたメタデータへのアクセス」を参照してください。
- AWS Glue Data Catalog のリソースへのアクセスに対するきめ細かなアクセス許可。Athena で使用されるデータカタログからの特定のデータベースとテーブルへのアクセスを制限または許可する、アイデンティティベースの (IAM) ポリシーを定義できるようになりました。詳細については、「AWS Glue Data Catalog のデータベースとテーブルへのきめ細かなアクセス」を参照してください。

> **Note**
>
> データは Amazon S3 バケットに格納されており、それらに対するアクセスは Amazon S3 へのアクセス権 によって制御されます。データベースとテーブル内のデータにアクセスするには、データを保存する Amazon S3 バケットへのアクセスコントロールポリシーを引き続き使用します。

2018 年 10 月 10 日

公開日: 2018 年 10 月 10 日

Athena では、SELECT クエリステートメントの結果からテーブルを作成する CREATE TABLE AS SELECT がサポートされています。詳細については、「クエリ結果からのテーブルの作成 (CTAS)」を参照してください。

CTAS クエリを作成する前に、Athena ドキュメントでそれらの動作について学習しておくことが重要です。ドキュメントには、クエリ結果を保存するための Amazon S3 の場所、CTAS クエリ結果を保存するためにサポートされている形式のリスト、作成できるパーティションの数、およびサポートされている圧縮形式に関する情報が記載されています。詳細については、「CTAS クエリに関する考慮事項と制約事項」を参照してください。

以下の目的で CTAS クエリを使用します。

- 1 つのステップでクエリ結果からテーブルを作成します。
- 例を使用して、Athena コンソールで CTAS クエリを作成します。構文については「CREATE TABLE AS」を参照してください。
- クエリ結果を他のストレージ形式 (例: PARQUET、ORC、AVRO、JSON、TEXTFILE) に変換します。詳細については、CTAS クエリに関する考慮事項と制約事項および列指向ストレージ形式を参照してください。

2018 年 9 月 6 日

公開日: 2018 年 09 月 06 日

ODBC ドライバーの新しいバージョン (バージョン 1.0.3) がリリースされました。新しいバージョンの ODBC ドライバーは、ページングではなく、デフォルトで結果をストリーミングします。これによりビジネスインテリジェンスツールで大規模なデータセットを迅速に取得できます。このバージョンには改善点、バグ修正、および更新された「プロキシサーバーで SSL を使用する」ドキュメントが含まれています。詳細については、ドライバーの「リリースノート」を参照してください。

ODBC ドライバーのバージョン 1.0.3 とそのドキュメントのダウンロードについては、「ODBC を使用した Amazon Athena への接続」を参照してください。

結果のストリーミング機能は、新しいバージョンの ODBC ドライバーでのみ使用できます。JDBC ドライバーでも使用できます。結果のストリーミングの詳細については、「ODBC ドライバーのインストールおよび設定ガイド」で「UseResultsetStreaming」を検索してください。

ODBC ドライバーのバージョン 1.0.3 は、以前のバージョンのドライバーのドロップインリプレースです。最新のドライバーに移行することをお勧めします。

> **Important**
>
> ODBC ドライバーバージョン 1.0.3 を使用するには、次の要件に従ってください。
>
> - アウトバウンドトラフィックに対してポート 444 を開いたままにする。
> - Athena のポリシーのリストに `athena:GetQueryResultsStream` ポリシーアクションを追加する。このポリシーアクションは、API で直接公開されていません。結果のストリーミングサポートの一部として ODBC ドライバーおよび JDBC ドライバーとともに使用するだけです。ポリシーの例については、「AWS 管理ポリシー: AWSQuicksightAthenaAccess」を参照してください。

2018 年 8 月 23 日

公開日: 2018 年 08 月 23 日

次に示すとおり DDL に関連する機能のサポートが追加され、バグがいくつか修正されました。

- Parquet のデータの `BINARY` と `DATE` データ型、Avro のデータの `DATE` と `TIMESTAMP` データ型のサポートが追加されました。
- DDL クエリの `INT` と `DOUBLE` のサポートが追加されました。`INTEGER` は `INT` のエイリアス、`DOUBLE PRECISION` は `DOUBLE` のエイリアスです。
- `DROP TABLE` と `DROP DATABASE` のクエリのパフォーマンスが改善されました。
- データバケットが空になっているときに Simple Storage Service (Amazon S3) で `_$folder$` オブジェクトが作成されなくなりました。
- パーティション値が指定されなかった場合、`ALTER TABLE ADD PARTITION` がエラーをスローする問題を修正しました。
- ステートメントに修飾名が指定された後に、パーティションをチェックするときに、`DROP TABLE` がデータベース名を無視する問題を修正しました。

Athena でサポートされているデータ型の詳細については、「Amazon Athena のデータ型」を参照してください。

Athena、JDBC ドライバー、および Java のデータ型間でサポートされているデータ型マッピングについては、「ODBC Driver Installation and Configuration Guide」の「Data Types」セクションを参照してください。

2018 年 8 月 16 日

公開日: 2018 年 08 月 16 日

JDBC ドライバーのバージョン 2.0.5 がリリースされました。新しいバージョンの JDBC ドライバーは、ページングではなく、デフォルトで結果をストリーミングします。これによりビジネスインテリジェンスツールで大規模なデータセットを迅速に取得できます。JDBC ドライバーの以前のバージョンと比較して、次のパフォーマンスが向上しています。

- 10,000 行未満を取得するときに、パフォーマンスが約 2 倍向上します。
- 10,000 行以上を取得するときに、パフォーマンスが 5 ~ 6 倍向上します。

結果のストリーミング機能は、JDBC ドライバーでのみ使用できます。ODBC ドライバーでは使用できません。これを Athena API で使用することはできません。結果のストリーミングの詳細については、「JDBC ドライバーのインストールおよび設定ガイド」で「UseResultsetStreaming」を検索してください。

JDBC ドライバーのバージョン 2.0.5 とそのドキュメントのダウンロードについては、「JDBC を使用した Amazon Athena への接続」を参照してください。

JDBC ドライバーのバージョン 2.0.5 は、以前のバージョンのドライバー (2.0.2) のドロップインリプレースです。JDBC ドライバーバージョン 2.0.5 が必ず使用されるようにするには、Athena のポリシーのリストに `athena:GetQueryResultsStream` ポリシーアクションを追加します。このポリシーアクションは、API で直接公開されていません。結果のストリーミングサポートの一部として JDBC ドライバーとともに使用するだけです。ポリシーの例については、「AWS 管理ポリシー: AWSQuicksightAthenaAccess」を参照してください。ドライバーのバージョン 2.0.2 からバージョン 2.0.5 への移行の詳細については、「JDBC ドライバー移行ガイド」を参照してください。

ドライバーをバージョン 1.x から 2.x に移行する場合は、既存の設定を新しい設定に移行する必要があります。最新バージョンのドライバーに移行することを強くお勧めします。詳細については、「JDBC Driver Migration Guide」(JDBC ドライバー移行ガイド) を参照してください。

2018 年 8 月 7 日

公開日: 2018 年 08 月 07 日

Amazon Virtual Private Cloud フローログを Amazon S3 に GZIP 形式で直接保存できるようになりました。これらは、Athena でクエリすることができます。詳細については、「Amazon VPC フローログのクエリ」および「Amazon VPC フローログが S3 で使用可能に」を参照してください。

2018 年 5 月 6 日

公開日: 2018 年 06 月 05 日

トピック

- ビューのサポート
- エラーメッセージの改善と更新
- バグ修正

ビューのサポート

ビューのサポートが追加されました。Athena で CREATE VIEW、DESCRIBE VIEW、DROP VIEW、SHOW CREATE VIEW、および SHOW VIEWS が使用できるようになりました。ビューを定義するクエリは、クエリでビューを参照するたびに実行されます。詳細については、「ビューの使用」を参照してください。

エラーメッセージの改善と更新

- GSON 2.8.0 ライブラリを CloudTrail SerDe に含めることで、CloudTrail SerDe の問題を解決し、JSON 文字列の改正を有効にしました。
- 列の順序変更を可能にすることによって、Parquet (場合によっては ORC) に対する Athena でのパーティションスキーマ検証が強化されました。これにより、Athena では徐々に進化するスキーマの変化と AWS Glue クローラから追加されるテーブルに適切に対応できるようになりました。詳細については、「スキーマ更新の処理」を参照してください。
- `SHOW VIEWS` に解析サポートが追加されました。
- 最も一般的なエラーメッセージに次の改善が加えられました。
  - Athena クエリで SerDe が列の解析に失敗するときの「Internal Error」メッセージを、説明的なエラーメッセージに置き換えました。これまで、Athena は解析エラーの場合に内部エラーを発行していました。新しいエラーメッセージは次のようになります。「HIVE_BAD_DATA: フィールド 0: java.lang のフィールド値の解析エラー。文字列は org.openx.data.jsonserde.json.JSONObject にキャストできません」。
  - アクセス許可不足のエラーメッセージが改善されて、詳細が追加されました。

バグ修正

以下のバグを修正しました。

- REAL から FLOAT データ型への内部変換を有効にする問題が修正されました。これにより、FLOAT データ型を返す AWS Glue クローラの統合が改善されました。
- Athena が AVRO DECIMAL (論理的な型) を DECIMAL 型に変換しない問題が修正されました。
- WHERE 句が TIMESTAMP データ型の値を参照している場合に、Parquet データをクエリすると、Athena で 結果が返されないという問題を修正しました。

2018 年 5 月 17 日

公開日: 2018 年 05 月 17 日

Athena でのクエリ同時実行のクォータが 5 件から 20 件に増加されました。これによって、20 件までの DDL クエリおよび 20 件までの SELECT クエリを同時に送信して実行できます。同時実行数のクォータは、DDL および SELECT クエリで別であることに注意してください。

Athena での同時実行クォータは、サービスに対して同時に送信できるクエリの数として定義されます。同じタイプ (DDL あるいは SELECT) のクエリを同時に 20 件まで送信できます。同時実行クエリのクォータを超えるクエリを送信すると、Athena API がエラーメッセージを表示します。

Athena にクエリを送信すると、Athena は、全体的なサービス負荷と着信リクエストの量に基づいてリソースを割り当てることによってクエリを処理します。当社では継続的にサービスをモニタリングして調整を行うため、お客様のクエリはできるだけ短期間で処理されます。

詳細については、Service Quotas を参照してください。これは調整可能なクォータです。Service Quotas コンソールを使用して、同時実行クエリのクォータの引き上げをリクエストできます。

2018 年 4 月 19 日

公開日: 2018 年 04 月 19 日

配列データ型として ResultSet を返すサポート、改良点、およびバグ修正を反映した、新しいバージョンの JDBC ドライバー (バージョン 2.0.2) をリリースしました。詳細については、ドライバーの「リリースノート」を参照してください。

JDBC ドライバーの新しいバージョン 2.0.2 とそのドキュメントのダウンロードについては、「JDBC を使用した Amazon Athena への接続」を参照してください。

JDBC ドライバーの最新バージョンは 2.0.2 です。ドライバーをバージョン 1.x から 2.x に移行する場合は、既存の設定を新しい設定に移行する必要があります。最新のドライバーに移行することを強くお勧めします。

新しいバージョンのドライバーの変更点、バージョンの相違点、例については、「JDBC ドライバー移行ガイド」を参照してください。

2018 年 4 月 6 日

公開日: 2018 年 04 月 06 日

Athena コンソールでのクエリの入力に自動入力を使用します。

2018 年 3 月 15 日

公開日: 2018 年 03 月 15 日

CloudTrail コンソールから直接 CloudTrail ログファイル用の Athena テーブルを自動作成する機能を追加しました。詳細については、CloudTrail ログ用の Athena テーブルを作成するための CloudTrail コンソールの使用 を参照してください。

2018 年 2 月 2 日

公開日: 2018 年 02 月 12 日

`GROUP BY` 句を使用するメモリに大きな負担をかけるクエリのディスクに中間データを安全にオフロードする機能が追加されました。これでこのようなクエリの信頼性が向上し、「Query resource exhausted」(クエリリソースの枯渇) エラーを回避します。

2018 年 1 月 19 日

公開日: 2018 年 01 月 19 日

Athena は、オープンソースの分散型クエリエンジンである Presto を使用してクエリを実行します。

Athena には管理するバージョンがありません。Athena の基盤となるエンジンを Presto 0.172 に基づくバージョンに透過的にアップグレードしました。ユーザー操作は必要はありません。

このアップグレードにより、Athena で Presto 0.172 Lambda Expressions を含めた Presto 0.172 Functions and Operators を使用できるようになります。

このリリースの主な更新 (コミュニティ提供の修正を含む) は以下のとおりです。

- ヘッダーを無視するサポート。Athena がヘッダーを無視できるように、テーブルの定義時に `skip.header.line.count` プロパティを使用できます。これは、Grok または Regex SerDes ではなく、LazySimpleSerDe と OpenCSV SerDe を使用するクエリをサポートします。

- STRING 関数での CHAR(n) データ型のサポート。CHAR(n) の範囲は [1.255] です。VARCHAR(n) の範囲は [1,65535] です。
- 相関サブクエリのサポート。
- Presto Lambda 式および関数のサポート。
- DECIMAL 型と演算子のパフォーマンスが向上しました。
- フィルタ処理された集計のサポート (SELECT sum(col_name) FILTER である id > 0 など)。
- DECIMAL、TINYINT、SMALLINT、および REAL の各データ型のプッシュダウン述語。
- 限定比較述語のサポート (ALL、ANY、および SOME)。
- 追加された関数: arrays_overlap()、array_except()、levenshtein_distance()、codepoint()、skewness() および typeof()。
- タイムゾーン引数を使用する from_unixtime() 関数のバリアントを追加しました。
- bitwise_and_agg() 集計関数と bitwise_or_agg() 集計関数を追加しました。
- xxhash64() 関数と to_big_endian_64() 関数を追加しました。
- JSON パスの添字でバックスラッシュを使用して二重引用符やバックスラッシュをエスケープするためのサポートを json_extract() 関数と json_extract_scalar() 関数に追加しました。これまでバックスラッシュは標準の文字として扱われていたために、これによってバックスラッシュを使用した呼び出しのセマンティクスが変わります。

関数と演算子については、本ガイドの「DML クエリ、関数、および演算子」、および Presto ドキュメントの「関数と演算子」を参照してください。

Athena はすべての Presto 機能をサポートしません。詳細については、「制限」を参照してください。

Athena リリースノート (2017 年)

2017 年 11 月 13 日

公開日: 2017 年 11 月 13 日

Athena を ODBC ドライバーに接続するためのサポートを追加しました。詳細については、ODBC を使用した Amazon Athena への接続 を参照してください。

2017 年 11 月 1 日

公開日: 2017 年 11 月 01 日

地理空間データのクエリのサポートをアジアパシフィック (ソウル)、アジアパシフィック (ムンバイ)、および欧州 (ロンドン) の各リージョンに追加しました。詳細については、「地理空間データのクエリ」と「AWS リージョン とエンドポイント」を参照してください。

2017 年 10 月 19 日

公開日: 2017 年 10 月 19 日

欧州 (フランクフルト) のサポートを追加しました。サポートされているリージョンの一覧は、「AWS リージョン とエンドポイント」を参照してください。

2017 年 10 月 3 日

公開日: 2017 年 10 月 03 日

AWS CloudFormation で名前付き Athena クエリを作成します。詳細については、「ユーザーガイド」の「AWS::Athena::NamedQuery」を参照してください。

2017 年 9 月 25 日

公開日: 2017 年 09 月 25 日

アジアパシフィック (シドニー) リージョンのサポートを追加しました。サポートされているリージョンの一覧は、「AWS リージョン とエンドポイント」を参照してください。

2017 年 8 月 14 日

公開日: 2017 年 08 月 14 日

AWS Glue Data Catalog との統合を追加したほか、Athena で管理されるデータカタログから AWS Glue Data Catalog に更新するための移行ウィザードを追加しました。詳細については、「AWS Glue との統合」を参照してください。

2017 年 8 月 4 日

公開日: 2017 年 08 月 04 日

Grok SerDe のサポートを追加しました。これにより、ログなどの構造化されていないテキストファイルのレコードのパターンマッチングが容易になります。詳細については、「Grok SerDe」を参照してください。コンソールを使用してクエリ履歴をスクロールするためのキーボードショートカットを追加しました (Windows での CTRL +⇧/⇩、Mac での CMD + ⇧/⇩)。

2017 年 6 月 22 日

公開日: 2017 年 06 月 22 日

アジアパシフィック (東京) およびアジアパシフィック (シンガポール) の各リージョンのサポートを追加しました。サポートされているリージョンの一覧は、「AWS リージョン とエンドポイント」を参照してください。

2017 年 6 月 8 日

公開日: 2017 年 06 月 08 日

欧州 (アイルランド) のサポートを追加しました。詳細については、AWS リージョン およびエンドポイントを参照してください。

2017 年 5 月 19 日

公開日: 2017 年 05 月 19 日

Amazon Athena API および Athena 用の AWS CLI サポートを追加し、JDBC ドライバーをバージョン 1.1.0 に更新しました。また、さまざまな問題を修正しました。

- Amazon Athena により、Athena のアプリケーションプログラミングが可能になります。詳細については、「Amazon Athena API リファレンス」を参照してください。最新の AWS SDK には、Athena API のサポートが含まれています。ドキュメントとダウンロードのリンクについては、「Amazon Web Services のツール」の「SDK」セクションを参照してください。
- AWS CLI には、Athena 用の新しいコマンドが含まれています。詳細については、「Amazon Athena API リファレンス」を参照してください。
- 新しい JDBC ドライバー 1.1.0 が利用可能になりました。最新バージョンでは、新しい Athena API のサポート、最新の特徴、バグ修正が追加されています。ドライバーを https://downloads.athena.us-east-1.amazonaws.com/drivers/AthenaJDBC41-1.1.0.jar からダウンロードしてください。最新の Athena JDBC ドライバーにアップグレードすることをお勧めします。ただし、以前のドライバーバージョンもまだ使用できます。以前のドライバーバージョンでは Athena

API をサポートしていません。詳細については、「JDBC を使用した Amazon Athena への接続」を参照してください。

- 以前のバージョンの Athena でポリシーステートメント固有であったアクションは廃止されました。JDBC ドライバーバージョン 1.1.0 にアップグレードして、カスタマー管理の IAM ポリシーやインライン IAM ポリシーを JDBC ユーザーにアタッチする場合は、IAM ポリシーを更新する必要があります。一方、以前のバージョンの JDBC ドライバーは Athena API をサポートしていないため、廃止されたアクションは以前のバージョンの JDBC ユーザーにアタッチされているポリシーでのみ指定できます。このため、カスタマー管理の IAM ポリシーやインライン IAM ポリシーを更新する必要はありません。
- これらのポリシー固有のアクションは、Athena API のリリース前に Athena で使用されていたものです。これらの廃止されたアクションは、バージョン 1.1.0 より前の JDBC ドライバーのポリシーでのみ使用します。JDBC ドライバーをアップグレードする場合は、廃止されたアクションを許可または拒否するポリシーステートメントを、以下に示す適切な API アクションに置き換えてください。置き換えないと、エラーが発生します。

廃止されたポリシー固有のアクション	対応する Athena API アクション
`athena:RunQuery`	`athena:StartQueryExecution`
`athena:CancelQueryExecution`	`athena:StopQueryExecution`
`athena:GetQueryExecutions`	`athena:ListQueryExecutions`

改良点

- クエリ文字列の長さの上限を 256 KB に増やしました。

バグ修正

- コンソールでクエリ結果をスクロールしたときに正常に表示されない問題を修正しました。
- Amazon S3 データファイルの `\u0000` 文字列のせいでエラーが発生するという問題を修正しました。

- JDBC ドライバーを通じて行ったクエリキャンセルのリクエストが失敗するという問題を修正しました。
- 米国東部 (オハイオ) の Simple Storage Service (Amazon S3) データで AWS CloudTrail SerDe が失敗するという問題を修正しました。
- パーティションされたテーブルで `DROP TABLE` が失敗するという問題を修正しました。

2017 年 4 月 4 日

公開日: 2017 年 04 月 04 日

Amazon S3 のデータ暗号化のサポートを追加しました。また、暗号化のサポート、改良点、およびバグ修正を反映した JDBC ドライバーのアップデート (バージョン 1.0.1) をリリースしました。

機能

- 以下の暗号化機能を追加しました。
  - Amazon S3 の暗号化されたデータのクエリのサポート。
  - Athena クエリ結果の暗号化のサポート。
- 新しいバージョンのドライバーは新しい暗号化機能をサポートし、改良点とバグ修正を追加しました。
- `ALTER TABLE` を使用して列を追加、置換、変更する機能を追加しました。詳細については、Hive ドキュメントの「Alter 列」を参照してください。
- LZO 圧縮データのクエリのサポートを追加しました。

詳細については、「保管中の暗号化」を参照してください。

改良点

- JDBC のクエリパフォーマンスが向上しました。ページサイズの改善により、100 行ではなく 1,000 行が返されます。
- JDBC ドライバーインターフェイスを使用してクエリをキャンセルする機能を追加しました。
- JDBC 接続 URL で JDBC オプションを指定する機能を追加しました。最新の JDBC ドライバーについては、「JDBC を使用した Amazon Athena への接続」を参照してください。
- ドライバーでの PROXY 設定を追加しました。この設定は、AWS SDK for Java の ClientConfiguration を使用して行えます。

バグ修正

以下のバグを修正しました。

- JDBC ドライバーを使用して複数のクエリを実行すると、スロットリングエラーが発生する場合がある。
- decimal データ型の射影時に JDBC ドライバーが停止する。
- テーブルでのデータ型の定義に関係なく、JDBC ドライバーはすべてのデータ型を文字列として返す。たとえば、INT データ型として定義されている列を `resultSet.GetObject()` を使用して選択すると、INT データ型が返されずに STRING データ型が返される。
- JDBC ドライバーでの認証情報の検証が、クエリの実行時に行われずに接続時に行われる。
- スキーマを URL と共に指定すると、JDBC ドライバーを介したクエリが失敗する。

2017 年 3 月 24 日

公開日: 2017 年 03 月 24 日

AWS CloudTrail SerDe の追加、パフォーマンスの向上、およびパーティション問題の修正を行いました。

機能

- CloudTrail ログを読み込むため、Hive JSON SerDe によって置き換えられた AWS CloudTrail SerDe が追加されました。CloudTrail ログのクエリについては、「AWS CloudTrail ログのクエリ」を参照してください。

改良点

- 多数のパーティションをスキャンするときのパフォーマンスを改善しました。
- `MSCK Repair Table` オペレーションのパフォーマンスを改善しました。
- プライマリリージョン以外のリージョンに保存されている Amazon S3 データをクエリする機能を追加しました。標準の Athena 料金に加えて、Amazon S3 の標準のリージョン間データ転送料金が適用されます。

バグ修正

- パーティションがロードされないときに「テーブルが見つかりません」エラーが発生するバグを修正しました。
- `ALTER TABLE ADD PARTITION IF NOT EXISTS` クエリで例外がスローされないようにバグを修正しました。
- `DROP PARTITIONS` のバグを修正しました。

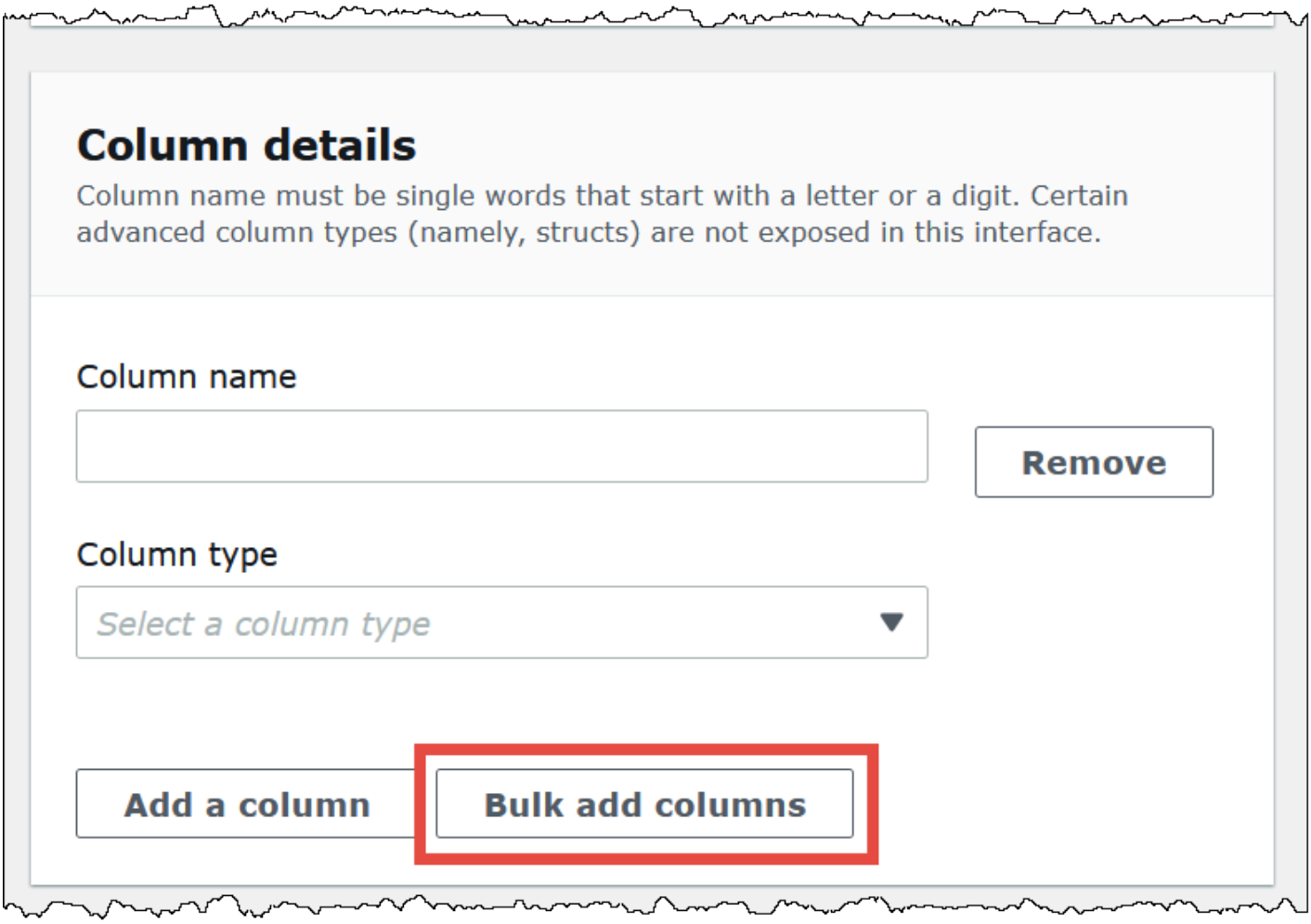
2017 年 2 月 20 日

公開日: 2017 年 02 月 20 日

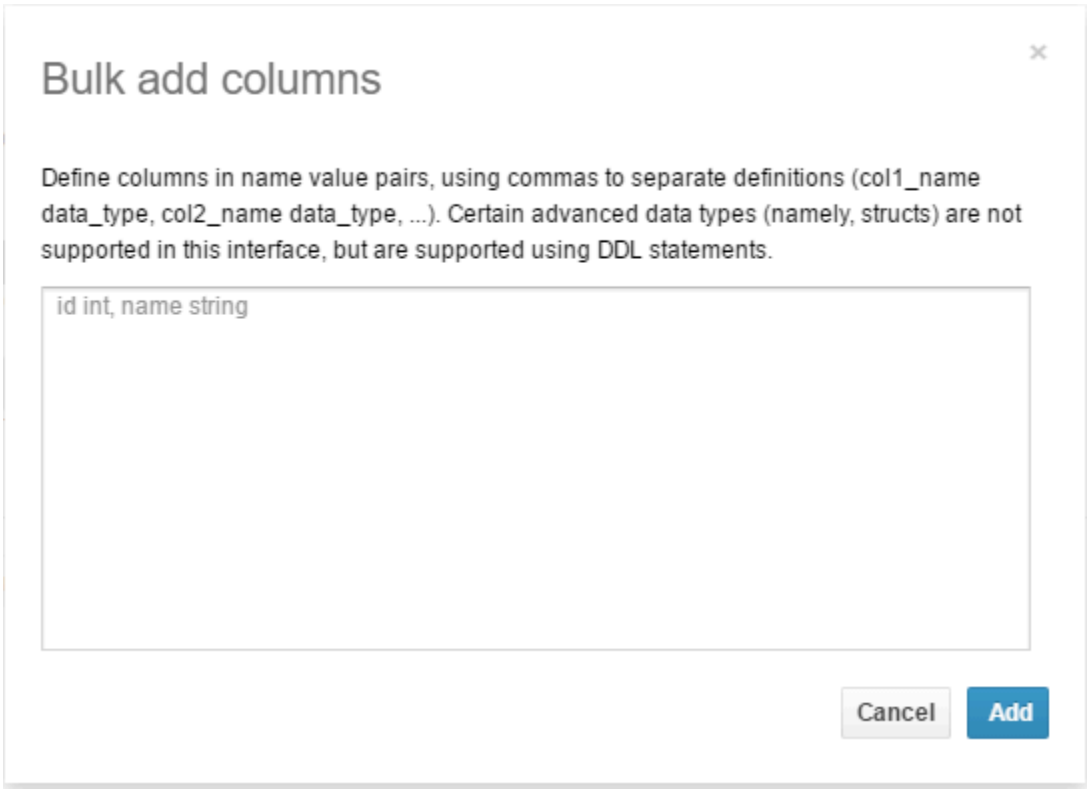
AvroSerDe と OpenCSVSerDe、米国東部 (オハイオ) リージョン、およびコンソールウィザードでの列の一括編集のサポートを追加しました。大規模な Parquet テーブルのパフォーマンスを改善しました。

機能

- 以下の新しい SerDe のサポートを導入しました。
  - Avro SerDe
  - CSV を処理するための OpenCSVSerDe
- 米国東部 (オハイオ) リージョン (us-east-2) での起動。このリージョンでクエリを実行できるようになりました。
- これで [Create Table From S3 bucket data] (S3 バケットデータからテーブルを作成) フォームを使用して、テーブルスキーマを一括で定義できるようになりました。クエリエディタで [Create] (作成)、[S3 bucket data] (S3 バケットデータ) と選択し、さらに [Column details] (列の詳細) セクションの [Bulk add columns] (列の一括追加) を選択します。



テキストボックスに名前と値のペアを入力し、[Add] を選択します。



改良点

- 大規模な Parquet テーブルのパフォーマンスを改善しました。

ドキュメント履歴

ドキュメントの最終更新日: 2024 年 5 月 28 日

お客様からのフィードバックに対応するために、ドキュメントを頻繁に更新しています。以下の表は、Amazon Athenaドキュメントへの重要な追加項目を説明するものです。すべての更新が記載されているわけではありません。

変更	説明	リリース日
AmazonAthenaFullAccess 管理ポリシーを更新しました。	datazone:ListDomains 、datazone:ListProjects 、datazone:ListAccountEnvironments のアクセス許可が AmazonAthenaFullAccess 管理ポリシーに追加されました。追加されたアクションにより、Athena ユーザーは Amazon DataZone ドメイン、プロジェクト、および環境で作業できるようになります。詳細については、「Athena で Amazon DataZone を使用する」を参照してください。	2024 年 1 月 3 日
AmazonAthenaFullAccess 管理ポリシーを更新しました。	glue:StartColumnStatisticsTaskRun 、glue:GetColumnStatisticsTaskRun 、glue:GetColumnStatisticsTaskRuns のアクセス許可を AmazonAthenaFullAccess 管理ポリシーに追加しました。追加されたアクションにより、Athena は AWS Glue を呼び出してコストベースのオプティマイザー機能の統計を取得できるようになります。詳細については、「コストベースオプティマイザーの使用」を参照してください。	2024 年 1 月 3 日
IAM アイデンティティセンターが有効な Athena ワークグループのドキュメントを追加しました。	IAM アイデンティティセンター認証モードを使用する Athena SQL ワークグループを作成できるようになりました。これらのワークグループでは、Amazon Athena や Amazon EMR Studio などの AWS サービス間で同じ ID を使用できます。詳細については、「IAM アイデンティティセンターが有効な Athena ワークグループの使用」を参照してください。	2023 年 12 月 5 日

変更	説明	リリース日
S3 Express One Zone のデータのクエリに関するドキュメントを追加しました。	Athena を使用して Amazon S3 Express One Zone ストレージクラス内のデータをクエリできます。詳細については、「S3 Express One Zone データのクエリ」を参照してください。	2023 年 11 月 28 日
Glue データカタログビューのドキュメントを追加しました。	Glue データカタログビューを使用して、Amazon Athena や Amazon Redshift などの AWS のサービス間で単一の共通ビューを提供できます。詳細については、「AWS Glue Data Catalog ビューの使用」を参照してください。	2023 年 11 月 27 日
コストベースオプティマイザー機能のドキュメントを追加しました。	AWS Glue からの統計を使用して Athena SQL のクエリを最適化できます。詳細については、「コストベースオプティマイザーの使用」を参照してください。	2023 年 11 月 17 日
Athena JDBC 3.x ドライバーの追加されたドキュメント	Athena JDBC 3.x ドライバーを使用し、Amazon S3 から直接クエリ結果を読み取ることができます。JDBC 3.x ドライバーは、JDBC 2.x ドライバーがサポートするほぼすべての認証方法をサポートします。詳細については、「Athena JDBC 3.x ドライバー」を参照してください。	2023 年 11 月 16 日
Athena で DataZone を使用するためのドキュメントを追加しました。	DataZone を使用して、Athena、AWS Glue、Lake Formation などの AWS 分析サービス全体において エクスペリエンスを簡素化できます。詳細については、「Athena で Amazon DataZone を使用する」を参照してください。	2023 年 10 月 4 日
キャパシティ予約のドキュメントを追加しました。	Amazon Athena でキャパシティ予約を使用して、フルマネージドコンピューティング性能で SQL クエリを実行できるようになりました。詳細については、「クエリ処理キャパシティの管理」を参照してください。	2023 年 4 月 28 日

変更	説明	リリース日
フェデレーテッドビューのクエリに関するドキュメントを追加しました。	Athena のフェデレーティッドデータソースでビューを作成してクエリできるようになりました。詳細については、「フェデレーションされたビューのクエリ」を参照してください。	2023 年 4 月 4 日
Amazon S3 におけるスロットリングの防止に関するドキュメントを追加しました。	詳細については、「Amazon S3 スロットリングの防止」を参照してください。	2023 年 3 月 24 日
`AmazonAthenaFullAccess` 管理ポリシーを更新しました。	AmazonAthenaFullAccess 管理ポリシーに `pricing:GetProducts` を追加しました。追加されたアクションにより、AWS Billing and Cost Management へのアクセスが提供されます。詳細については、「AWS Billing and Cost Management API リファレンス」の「GetProducts」を参照してください。	2023 年 1 月 25 日
Athena 圧縮サポートの拡張ドキュメント。	Hive テーブル圧縮、Iceberg テーブル圧縮 および ZSTD 圧縮レベル に追加された個別のトピック。詳細については、「Athena での圧縮のサポート」を参照してください。	2023 年 1 月 20 日
Amazon Athena for Apache Spark のドキュメントを追加しました。	Amazon Athena で Apache Spark アプリケーションと Jupyter 互換ノートブックをインタラクティブに作成、実行できるようになりました。詳細については、「Amazon Athena での Apache Spark の使用」を参照してください。	2022 年 11 月 30 日
Athena IBM Db2 コネクタに関するドキュメントを追加しました。	IBM Db2 用の Amazon Athena コネクタを使用して、Athena から Db2 にクエリを実行できます。詳細については、「Amazon Athena IBM Db2 コネクタ」を参照してください。	2022 年 11 月 18 日

変更	説明	リリース日
クエリ結果の再利用に関するドキュメントを追加しました。	Athena でクエリを再実行する際に、最後に保存されたクエリ結果を再利用するかをオプションで選択できるようになりました。これにより、パフォーマンスを向上させ、スキャンするバイト数のコストを削減できます。詳細については、「クエリの結果を再利用する」を参照してください。	2022 年 11 月 8 日
CloudTrail ログに関するドキュメントの更新。	CloudTrail ログをクエリするための CREATE TABLE DDL が更新され、CloudTrail SerDe の代わりに JSON SerDe が使用されるようになりました。詳細については、「AWS CloudTrail ログのクエリ」を参照してください。	2022 年 11 月 3 日
Athena エンジンバージョン 3 に関するドキュメントを追加しました。	Athena エンジンバージョン 3 の詳細については、「Athena エンジンバージョン 3」を参照してください。	2022 年 10 月 13 日
Okta プラグインを使用した ODBC 用の SSO の設定に関するチュートリアルを追加しました。	Okta の ID プロバイダを使用して、Single Sign-On (SSO) 機能用の Amazon Athena ODBC ドライバーと Okta プラグインを設定します。詳細については、「Okta プラグインと Okta ID プロバイダを使用して ODBC 用 の SSO を設定する」を参照してください。	2022 年 8 月 23 日
Athena コンソールでのクエリプランと統計情報の表示に関するドキュメントを追加しました。	Athena クエリエディタを使用して、クエリがどのように実行されるかについてのグラフィック表現や、完了したクエリがどのように実行されたかについてのグラフ、詳細、統計を表示できます。詳細については、SQL クエリの実行プランの表示および完了したクエリの統計と実行の詳細の表示を参照してください。	2022 年 7 月 21 日

変更	説明	リリース日
外部 Hive メタストアで Apache Hive ビューをクエリするためのドキュメントを追加しました。	Athena を使用して、外部 Hive メタストアで作成された Apache ビューをクエリできます。一部の Hive 関数はサポートされていないか、特別な処理が必要です。詳細については、「Hive ビューの使用」を参照してください。	2022 年 4 月 22 日
保存されたクエリのドキュメントを追加しました。	Athena の保存されたクエリ機能を使用して、クエリの保存、呼び出し、編集、名前変更を実行できます。詳細については、本ガイドの「保存されたクエリの使用」および Amazon Athena API リファレンスの UpdateNamedQuery を参照してください。	2022 年 2 月 28 日
Apache Iceberg サポートのプレビューに関するドキュメントを追加しました。	Athena では、データに Apache Parquet 形式を使用しメタストアには AWS Glue カタログを使用する Apache Iceberg テーブルについて、その読み込み、タイムトラベル、および書き込みの各クエリをサポートしています。詳細については、「Apache Iceberg テーブルの使用」を参照してください。	2021 年 11 月 26 日
クロスアカウントの横串検索に関するドキュメントを追加しました。	クロスアカウントの横串検索機能を使用すると、別のアカウントのデータソースをクエリできます。この機能を有効にするためのアクセス許可のセットアップ方法については、「クロスアカウントのフェデレーションクエリの有効化」(クロスアカウントのフェデレーティッドクエリを有効化する) を参照してください。	2021 年 11 月 12 日
Athena UNLOAD ステートメントに関するドキュメントを追加しました。	UNLOAD ステートメントを使用して、SELECT ステートメントのクエリ結果を Apache Parquet、ORC、Apache Avro、および JSON 形式に書き込みます。詳細については、「UNLOAD」を参照してください。	2021 年 8 月 5 日

変更	説明	リリース日
Athena EXPLAIN ステートメント機能に関するドキュメントを追加しました。	詳細については、Athena での EXPLAIN および EXPLAIN ANALYZE の使用およびAthena EXPLAIN ステートメントの結果についてを参照してください。	2021 年 4 月 5 日
Athena でのトラブルシューティングとパフォーマンスチューニングに関するページを追加しました。	詳細については、Athena のトラブルシューティングおよびAthena でのパフォーマンスのチューニングを参照してください。	2020 年 12 月 30 日
Athena エンジンのバージョン管理と Athena エンジンのバージョン 2 に関するドキュメントを追加しました。	詳細については、「Athena エンジンのバージョニング」を参照してください。	2020 年 11 月 11 日
一般公開リリースのために横串検索ドキュメントを更新しました。	詳細については、Amazon Athena 横串検索の使用およびAthena での CalledVia コンテキストキーの使用を参照してください。	2020 年 11 月 11 日
Athena へのフェデレーションアクセスのために Lake Formation で JDBC ドライバーを使用するためのドキュメントを追加しました。	詳細については、Athena へのフェデレーションアクセスのための Lake Formation と Athena JDBC および ODBC ドライバーの使用およびチュートリアル: Lake Formation および JDBC を使用した Okta ユーザーの Athena へのフェデレーションアクセスの設定を参照してください。	2020 年 9 月 25 日

変更	説明	リリース日
Amazon Athena の OpenSearch データコネクタに関するドキュメントを追加しました。	詳細については、「Amazon Athena OpenSearch コネクタ」を参照してください。	2020 年 7 月 21 日
Hudi データセットのクエリに関するドキュメントを追加しました。	詳細については、「Athena を使用した Apache Hudi データセットのクエリ」を参照してください。	2020 年 7 月 9 日
Amazon S3 に保存された Apache ウェブサーバーログと IIS ウェブサーバーログのクエリに関するドキュメントを追加しました。	詳細については、Simple Storage Service (Amazon S3) に保存されている Apache ログのクエリおよびSimple Storage Service (Amazon S3) に保存されているインターネットインフォメーションサーバー (IIS) ログのクエリを参照してください。	2020 年 7 月 8 日
外部 Hive メタストア用の Athena データコネクタの一般リリースに関するドキュメントを追加しました。	詳細については、「外部 Hive メタストア用の Athena データコネクタの使用」を参照してください。	2020 年 6 月 1 日
データカタログリソースのタグ付けに関するドキュメントが追加されました。	詳細については、「Athena リソースへのタグ付け」を参照してください。	2020 年 6 月 1 日

変更	説明	リリース日
パーティション射影に関するドキュメントを追加しました。	詳細については、「Amazon Athena でのパーティション射影」を参照してください。	2020 年 5 月 21 日
Athena 向けの Java コード例を更新しました。	詳細については、「コードサンプル」を参照してください。	2020 年 5 月 11 日
Amazon GuardDuty の結果のクエリに関するトピックを追加しました。	詳細については、「Amazon GuardDuty の結果のクエリ」を参照してください。	2020 年 3 月 19 日
CloudWatch Events を使用した Athena クエリの状態推移のモニタリングに関するトピックを追加しました。	詳細については、「Amazon EventBridge イベントで Athena クエリを監視する」を参照してください。	2020 年 3 月 11 日
Athena を使用した AWS Global Accelerator フローログのクエリに関するトピックを追加しました。	詳細については、「AWS Global Accelerator フローログのクエリ」を参照してください。	2020 年 2 月 6 日

変更	説明	リリース日
• パーティションされていないソースからパーティションされた宛先にデータを追加するための CTAS と INSERT INTO の使用に関するドキュメントを追加しました。 • Athena 用 ODBC ドライバーの 1.1.0 プレビューバージョンのダウンロードリンクを追加しました。 • SHOW DATABASES LIKE 正規表現の説明を修正しました。 • CTA トピックの partitioned_by 構文を修正しました。 • その他のマイナーな修正。	ドキュメントの更新には次のトピックが含まれますが、これらに限定されません。 • ETL およびデータ分析での CTAS および INSERT INTO の使用 • ODBC を使用した Amazon Athena への接続 (1.1.0 プレビュー機能は、現在 1.1.2 ODBC ドライバーに含まれています)。 • SHOW DATABASES • CREATE TABLE AS	2020 年 2 月 4 日

変更	説明	リリース日
INSERT INTO で CTAS を使用して、パーティション化されていないソースからパーティションされた宛先にデータを追加する方法に関するドキュメントが追加されました。	詳細については、「CTAS および INSERT INTO を使用して 100 パーティションの制限を回避する」を参照してください。	2020 年 1 月 22 日
クエリ結果の場所情報が更新されました。	Athena が「デフォルト」のクエリ結果の場所を作成しなくなりました。詳細については、「クエリ結果の場所の指定」を参照してください。	2020 年 1 月 20 日
AWS Glue Data Catalog のクエリに関するトピックを追加しました。Athena のサービスクォータに関する情報を更新しました。サービスクォータは「service limitations」(サービス制限) という旧称でした。	詳細については、次のトピックを参照してください。 • AWS Glue Data Catalog のクエリ • Service Quotas	2020 年 1 月 17 日
OpenCSVSerDe のトピックを修正し、TIMESTAMP タイプは UNIX 数値形式で指定する必要があることを示しました。	詳細については、「CSV を処理するための OpenCSVSerDe」を参照してください。	2020 年 1 月 15 日

変更	説明	リリース日
暗号化に関するセキュリティトピックを更新して、Athena が非対称キーをサポートしないことを追記しました。	Athena は、データの読み書きに対称キーのみをサポートします。 詳細については、「サポートされる Amazon S3 の暗号化オプション」を参照してください。	2020 年 1 月 8 日
カスタム AWS KMS キーを使用して暗号化された Amazon S3 バケットへのクロスアカウントアクセスに関する情報を追加しました。	詳細については、「カスタム AWS KMS キーで暗号化されたバケットへのクロスアカウントアクセス」を参照してください。	2019 年 12 月 13 日
フェデレーティッドクエリ、外部 Hive メタストア、機械学習、およびユーザー定義関数のドキュメントを追加しました。新しい CloudWatch メトリクスを追加しました。	詳細については、次のトピックを参照してください。 • Amazon Athena 横串検索の使用 • 使用可能なデータソースコネクタ • 外部 Hive メタストア用の Athena データコネクタの使用 • Machine Learning (ML) with Amazon Athena の使用 • ユーザー定義関数を使用したクエリ • Athena 向けの CloudWatch メトリクスとディメンションのリスト	2019 年 11 月 26 日

変更	説明	リリース日
新しい INSERT INTO コマンドと、データマニフェストファイルをサポートするための更新されたクエリ結果の場所情報のセクションを追加しました。	詳細については、INSERT INTOおよびクエリ結果、最近のクエリ、および出力ファイルの使用を参照してください。	2019 年 9 月 18 日
インターフェイス VPC エンドポイント (PrivateLink) のサポートに関するセクションを追加しました。JDBC ドライバーを更新しました。強化された VPC フローログに関する更新済みの情報です。	詳細については、インターフェイス VPC エンドポイントを使用して Amazon Athena に接続する、Amazon VPC フローログのクエリ、およびJDBC を使用した Amazon Athena への接続を参照してください。	2019 年 9 月 11 日
AWS Lake Formation との統合に関するセクションを追加しました。	詳細については、「Athena を使用して AWS Lake Formation に登録されたデータをクエリする」を参照してください。	2019 年 6 月 26 日
他の AWS のサービスとの整合性のためにセキュリティセクションを更新しました。	詳細については、「Amazon Athena のセキュリティ」を参照してください。	2019 年 6 月 26 日

変更	説明	リリース日
AWS WAF ログのクエリに関するセクションを追加しました。	詳細については、「AWS WAF ログのクエリ」を参照してください。	2019 年 5 月 31 日
Athena ワークグループをサポートする新しいバージョンの ODBC ドライバーがリリースされました。	ODBC ドライバーのバージョン 1.0.5 とそのドキュメントをダウンロードするには、「ODBC を使用した Amazon Athena への接続」を参照してください。ワークグループでタグを使用しても、ODBC ドライバーの接続文字列は変更されません。タグを使用するには、最新バージョンの ODBC ドライバー (この最新バージョン) にアップグレードしてください。 このドライバーバージョンでは、Athena API ワークグループアクションを使用してワークグループを作成および管理し、Athena API タグアクションを使用してワークグループのタグを追加、リスト化、または削除できます。使用を開始する前に、ワークグループとタグに対するアクションに関するリソースレベルの IAM 許可があることを確認してください。	2019 年 3 月 5 日
Amazon Athena のワークグループにタグのサポートを追加しました。	タグは、1 つのキーと 1 つの値で構成されており、どちらもお客様側が定義します。ワークグループにタグを付けるときは、カスタムメタデータを割り当てます。たとえば、コストセンターごとにワークグループを作成します。その後、これらのワークグループにタグを追加することで、各コストセンターの Athena 支出を追跡できます。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「請求へのタグの使用」を参照してください。	2019 年 2 月 22 日

変更	説明	リリース日
Athena で使用される JSON OpenX SerDe を改善しました。	改善点は次のとおりです。 • `ConvertDotsInJsonKeysToUnderscores` プロパティをサポートします。TRUE に設定すると、SerDe はキー名のドットをアンダースコアに置き換えることができます。たとえば、JSON データセットに "a.b" という名前のキーが含まれている場合は、このプロパティを使用して、Athena で列名が "a_b" になるように定義できます。デフォルト: FALSE。デフォルトで、Athena は列名にドットを許可しません。 • `case.insensitive` プロパティをサポートします。デフォルトで、Athena では JSON データセットのすべてのキーに小文字が使用される必要があります。`WITH SERDE PROPERTIES ("case.insensitive"= FALSE;)` を使用すると、データ内で大文字と小文字を区別してキー名を使用できます。デフォルト: TRUE。TRUE に設定すると、SerDe はすべての大文字の列を小文字に変換します。 詳細については、「OpenX JSON SerDe」を参照してください。	2019 年 2 月 18 日

変更	説明	リリース日
ワークグループ用のサポートを追加しました。	ワークグループを使用して、ユーザー、チーム、アプリケーション、またはワークロードを分離し、各クエリまたはワークグループ全体で処理できるデータ量に制限を設定します。ワークグループは IAM リソースとして機能するため、リソースレベルのアクセス許可を使用して特定のワークグループへのアクセスを制御できます。Amazon CloudWatch でクエリ関連のメトリクスを表示する、スキャンされるデータの量に制限を設定することによってクエリコストを管理する、しきい値を作成する、およびこれらのしきい値を超えたときに Amazon SNS アラームなどのアクションをトリガーすることもできます。詳細については、クエリを実行するためのワークグループの使用およびCloudWatch のメトリクスとイベントを使用したコストの管理とクエリのモニタリングを参照してください。	2019 年 2 月 18 日
Network Load Balancer からのログの分析に対するサポートが追加されました。	Network Load Balancer からのログを分析するための Athena クエリの例が追加されました。これらのログは、Network Load Balancer に送信された Transport Layer Security (TLS) リクエストに関する詳細情報を受け取ります。これらのアクセスログを使用して、トラフィックパターンを分析し、問題のトラブルシューティングを行えます。詳細については、the section called “Network Load Balancer” を参照してください。	2019 年 1 月 24 日

変更	説明	リリース日
AD FS および SAML 2.0 (Security Assertion Markup Language 2.0) を使用した Athena API へのフェデレーションアクセスをサポートする新しいバージョンの JDBC および ODBC ドライバーをリリースしました。	ドライバーのこのリリースによって、Athena へのフェデレーションアクセスが Active Directory Federation Service (AD FS 3.0) 対応になります。アクセスは、SAML 2.0 をサポートするバージョンの JDBC や ODBC ドライバーを通じて確立されます。Athena API へのフェデレーションアクセスの設定に関する詳細については、「the section called “Athena API へのフェデレーションアクセスの有効化”」を参照してください。	2018 年 11 月 10 日
Athena 内のデータベースとテーブルに対するきめ細かなアクセスコントロールのサポートを追加しました。さらに、データカタログ内のデータベースとテーブルメタデータを暗号化できるようにするポリシーを Athena に追加しました。	アイデンティティベース (IAM) ポリシーを作成するためのサポートを追加しました。これにより、Athena で使用されるデータベースやテーブルなど、AWS Glue Data Catalog のリソースに対するきめ細かなアクセスコントロールが可能になります。 さらに、特定のポリシーを Athena に追加することにより、データカタログ内のデータベースとテーブルメタデータを暗号化できます。 詳細については、「AWS Glue Data Catalog のデータベースとテーブルへのきめ細かなアクセス」を参照してください。	2018 年 10 月 15 日

変更	説明	リリース日
CREATE TABLE AS SELECT ステートメントのサポートを追加しました。 ドキュメントでいくつか記述を修正しました。	CREATE TABLE AS SELECT ステートメントのサポートを追加しました。「クエリ結果からのテーブルの作成 (CTAS)」、「CTAS クエリに関する考慮事項と制約事項」、および「CTAS クエリの例」を参照してください。	2018 年 10 月 10 日
ODBC ドライバーバージョン 1.0.3 をリリースしました。このバージョンでは、ページ単位での結果の取得ではなく、結果のストリーミングがサポートされています。 ドキュメントでいくつか記述を修正しました。	ODBC ドライバーバージョン 1.0.3 では、結果のストリーミングがサポートされ、改善、バグ修正、および「プロキシサーバーでの SSL の使用」でのドキュメントの更新も含まれています。 ODBC ドライバーのバージョン 1.0.3 とそのドキュメントのダウンロードについては、「ODBC を使用した Amazon Athena への接続」を参照してください。	2018 年 9 月 6 日

変更	説明	リリース日
JDBC ドライバーのバージョン 2.0.5 がリリースされました。これにより結果のページ単位での取得ではなく、ストリーミングがデフォルトでサポートされます。 ドキュメントでいくつか記述を修正しました。	JDBC ドライバーのバージョン 2.0.5 がリリースされました。これにより結果のページ単位での取得ではなく、ストリーミングがデフォルトでサポートされます。詳細については、JDBC を使用した Amazon Athena への接続 を参照してください。	2018 年 8 月 16 日
GZIP 形式で Amazon S3 に直接保存できる Amazon Virtual Private Cloud フローログのクエリに関するドキュメントを更新しました。 ALB ログのクエリ例を更新しました。	GZIP 形式で Amazon S3 に直接保存できる Amazon Virtual Private Cloud フローログのクエリに関するドキュメントを更新しました。詳細については、Amazon VPC フローログのクエリ を参照してください。 ALB ログのクエリ例を更新しました。詳細については、Application Load Balancer ログのクエリ を参照してください。	2018 年 8 月 7 日

変更	説明	リリース日
ビューのサポートが追加されました。さまざまなデータストレージ形式でスキーマを操作するためのガイドラインが追加されました。	ビューのサポートが追加されました。詳細については、ビューの使用 を参照してください。 さまざまなデータストレージ形式におけるスキーマの更新を処理するガイダンスでこのガイドが更新されました。詳細については、スキーマ更新の処理 を参照してください。	2018 年 5 月 6 日
デフォルトのクエリ同時実行制限が 5 件から 20 件に増加されました。	20 件までの DDL クエリおよび 20 件までの SELECT クエリを同時に送信して実行できます。詳細については、Service Quotas を参照してください。	2018 年 5 月 17 日
クエリタブと、クエリエディタで自動入力を設定する機能を追加しました。	クエリタブと、クエリエディタで自動入力を設定する機能を追加しました。詳細については、開始 を参照してください。	2018 年 5 月 8 日
JDBC ドライバーのバージョン 2.0.2 がリリースされました。	JDBC ドライバーの新しいバージョン (バージョン 2.0.2) がリリースされました。詳細については、JDBC を使用した Amazon Athena への接続 を参照してください。	2018 年 4 月 19 日
Athena コンソールでのクエリ入力時における自動入力機能を追加しました。	Athena コンソールでのクエリ入力時における自動入力機能を追加しました。	2018 年 4 月 6 日

変更	説明	リリース日
CloudTrail コンソールから直接 CloudTrail ログファイル用の Athena テーブルを作成する機能を追加しました。	CloudTrail コンソールから直接 CloudTrail ログファイル用の Athena テーブルを自動作成する機能を追加しました。詳細については、CloudTrail ログ用の Athena テーブルを作成するための CloudTrail コンソールの使用 を参照してください。	2018 年 3 月 15 日
GROUP BY を使用したクエリ用に中間データを安全にディスクにオフロードできるようになりました。	GROUP BY 句を使用するメモリに大きな負担をかけるクエリのディスクに中間データを安全にオフロードする機能が追加されました。これでこのようなクエリの信頼性が向上し、「Query resource exhausted」(クエリリソースの枯渇) エラーを回避します。詳細については、「2018 年 2 月 2 日」のリリースノートを参照してください。	2018 年 2 月 2 日
Presto バージョン 0.172 のサポートが追加されました。	Amazon Athena の基盤となるエンジンを Presto 0.172 に基づくバージョンにアップグレードしました。詳細については、「2018 年 1 月 19 日」のリリースノートを参照してください。	2018 年 1 月 19 日
ODBC ドライバーのサポートを追加しました。	Athena を ODBC ドライバーに接続するためのサポートを追加しました。詳細については、「ODBC を使用した Amazon Athena への接続」を参照してください。	2017 年 11 月 13 日

変更	説明	リリース日
アジアパシフィック (ソウル)、アジアパシフィック (ムンバイ)、および欧州 (ロンドン) の各リージョンのサポートを追加しました。地理空間データのクエリのサポートを追加しました。	地理空間データのクエリと、アジアパシフィック (ソウル)、アジアパシフィック (ムンバイ)、および欧州 (ロンドン) の各リージョンのサポートを追加しました。詳細については、「地理空間データのクエリ」および「AWS リージョン とエンドポイント」を参照してください。	2017 年 11 月 1 日
欧州 (フランクフルト) のサポートを追加しました。	欧州 (フランクフルト) のサポートを追加しました。サポートされているリージョンのリストについては、「AWS リージョン およびエンドポイント」を参照してください。	2017 年 10 月 19 日
AWS CloudFormation を使用した名前付き Athena クエリのサポートを追加しました。	AWS CloudFormation を使用した名前付き Athena クエリの作成のサポートを追加しました。詳細については、「ユーザーガイド」の「AWS::Athena::NamedQuery」を参照してください。	2017 年 10 月 3 日
アジアパシフィック (シドニー) リージョンのサポートを追加しました。	アジアパシフィック (シドニー) リージョンのサポートを追加しました。サポートされているリージョンのリストについては、「AWS リージョン およびエンドポイント」を参照してください。	2017 年 9 月 25 日

変更	説明	リリース日
本ガイドに、AWS のサービス のログや、その他の型のデータ (マッピング、配列、ネストされたデータ、JSON を含むデータなど) に対するクエリについてのセクションを追加しました。	Athena での「AWS のサービス ログのクエリ」および異なる型のデータのクエリに関する例を追加しました。詳細については、Amazon Athena を使用した SQL クエリの実行 を参照してください。	2017 年 9 月 5 日
AWS Glue Data Catalog のサポートが追加されました。	AWS Glue Data Catalog との統合を追加したほか、Athena で管理されるデータカタログから AWS Glue Data Catalog に更新するための移行ウィザードを追加しました。詳細については、「AWS Glue との統合」および「AWS Glue」を参照してください。	2017 年 8 月 14 日
Grok SerDe のサポートを追加しました。	Grok SerDe のサポートを追加しました。これにより、ログなどの構造化されていないテキストファイルのレコードのパターンマッチングが容易になります。詳細については、「Grok SerDe」を参照してください。コンソールを使用してクエリ履歴をスクロールするキーボードショートカットを追加しました。	2017 年 8 月 4 日
アジアパシフィック (東京) のサポートを追加しました。	アジアパシフィック (東京) およびアジアパシフィック (シンガポール) の各リージョンのサポートを追加しました。サポートされているリージョンのリストについては、「AWS リージョン およびエンドポイント」を参照してください。	2017 年 6 月 22 日
欧州 (アイルランド) のサポートを追加しました。	欧州 (アイルランド) のサポートを追加しました。詳細については、「AWS リージョン およびエンドポイント」を参照してください。	2017 年 6 月 8 日

変更	説明	リリース日
Amazon Athena API および AWS CLI サポートを追加しました。	Amazon Athena API、および Athena に対する AWS CLI サポートを追加しました。JDBC ドライバーがバージョン 1.1.0 に更新されました。	2017 年 5 月 19 日
Amazon S3 データ暗号化のサポートを追加しました。	Amazon S3 のデータ暗号化のサポートを追加し、暗号化サポート、改良点、およびバグ修正を反映した JDBC ドライバー更新 (バージョン 1.0.1) をリリースしました。詳細については、「保管中の暗号化」を参照してください。	2017 年 4 月 4 日
AWS CloudTrail SerDe を追加しました。	AWS CloudTrail SerDe の追加、パフォーマンスの向上、およびパーティション問題の修正を行いました。 • AWS CloudTrail SerDe は CloudTrail ログを読み込むための Hive JSON SerDe に取って代わられました。CloudTrail ログのクエリについては、「AWS CloudTrail ログのクエリ」を参照してください。 • 多数のパーティションをスキャンするときのパフォーマンスを改善しました。 • `MSCK Repair Table` オペレーションのパフォーマンスを改善しました。 • プライマリリージョン以外のリージョンに保存されている Amazon S3 データをクエリする機能を追加しました。標準の Athena 料金に加えて、Amazon S3 の標準のリージョン間データ転送料金が適用されます。	2017 年 3 月 24 日
米国東部 (オハイオ) のサポートを追加しました。	Avro SerDe と CSV を処理するための OpenCSVSerDe、米国東部 (オハイオ)、およびコンソールウィザードでの列の一括編集のサポートを追加しました。大規模な Parquet テーブルのパフォーマンスを改善しました。	2017 年 2 月 20 日
	「Amazon Athena ユーザーガイド」の初回リリース。	2016 年 11 月

AWS 用語集

AWS の最新の用語については、「AWS の用語集 リファレンス」の「AWS 用語集」を参照してください。